Radio Shack
TRS-80
MODEL II
MICRO
COMPUTER

COBOL Development System

CUSTOM MANUFACTURED IN USA BY RADIO SHACK A DIVISION OF TANDY CORPORATION

Cat. No.
26-4703

Important Note For Model II RSCOBOL Users (Catalog Number 26-4703)

It is important to note that it isn't possible to REWRITE on a Sequential File. (See page 23 of **Use** section and pages 185 and 209 of the **RSCOBOL** section in your manual.)

If it is necessary for you to do a REWRITE, create a Relative File instead. (See page 211 of the **RSCOBOL** section.)

Also note that a Sequential File cannot be opened (the OPEN Mode) with an Input-Output statement. (Refer to page 184-5 of the **RSCOBOL** section.)

Thank-You!

Radio Shack®

A DIVISION OF TANDY CORPORATION

8759120-681

IMPORTANT NOTICE

For User's of the Radio Shack Model II
COBOL Runtime (RUNCOBOL)
26-4703 and 26-4704
TRSDOS 1.2 Only

We strongly recommend that you do **not** execute COBOL applications programs inside a TRSDOS DO-file. The reason is, if you press BREAK during DO-file processing, certain COBOL data file types will be destroyed. (ISAM files are destroyed, since RUNCOBOL is not able to update the index before TRSDOS closes the file.)

This caution,only applies to the use of RUNCOBOL inside DO-files. Pressing BREAK during normal execution of COBOL programs (not in DO-files) will not cause this problem. Furthermore, if the RUNCOBOL command is the **last** command in a DO-file, pressing BREAK will not cause this problem.

Thank You
Radio Shack Technical Publications
Fort Worth, Texas 76102

8759044 **PRINTED IN U.S.A.**

An Overview of the Model II COBOL Documentation Package

This binder contains the information you need to use the Radio Shack COBOL system. It assumes you are familiar with the general operation of the Computer, including use of the TRSDOS operating system. The package includes three manuals.

System User's Guide

Provides general information, start-up procedures, compiler commands, creation and use of a minimal-system runtime diskette, sample programs and sample session.

CEDIT User's Guide

Describes how to create and edit COBOL source files, using the COBOL editor CEDIT. Also includes instructions on using the text printout utility COBPRT.

RSCOBOL Language Reference Manual

A complete description of the Radio Shack version of the COBOL programming language. Newcomers to COBOL should consult a standard COBOL textbook for tutorial material.

USE

Radio Shack®

TRS-80™ Model II RSCOBOL System User's Guide

General Information, Compiler Use, Start-up and Sample Programs and Sample Session

CUSTOM MANUFACTURED FOR RADIO SHACK A DIVISION OF TANDY CORPORATION

TRS-80 Model II

COBOL USER'S GUIDE

MARCH, 1980

PREFACE

This document contains the information required to compile, run and debug COBOL language programs on the Radio Shack TRS-80 Model II Microcomputer under the TRSDOS Disk Operating System.

It assumes the reader is familiar with the COBOL Language, the general operation of the TRS-80 Model II Microcomputer, and the TRSDOS Operating System. The reader is specifically referred to the following publications:

TRS-80 Model II COBOL Language Manual
TRS-80 Model II Operation Manual
TRS-80 Model II Disk Operating System Reference Manual

This guide is organized such that each chapter fully describes a particular operational procedure. While the experienced user need only refer to the appropriate chapter, it is recommended that the first-time user read the complete guide prior to operation of the COBOL system.

COPYRIGHT NOTICES

NOTICE TO PROGRAMMERS

By your purchase of the software product described in this book, you have obtained a license to duplicate TRSDOS and Model II COBOL only as necessary for your personal use.

If you intend to sell COBOL applications programs you have written for the TRS-80 Model II, you must follow the procedure described below to avoid violation of this license and of the copyright laws.

The complete Radio Shack COBOL Development System (26-4703) includes the TRSDOS (TM) operating system, the RSCOBOL compiler, the RUNCOBOL runtime and numerous auxiliary files.

RSCOBOL produces an intermediate code which can only be executed by the runtime system RUNCOBOL. Therefore your compiled program will require that the user have TRSDOS and RUNCOBOL from Radio Shack.

Since you may not duplicate TRSDOS or RUNCOBOL for resale, you have two options for selling a copy of your own program:

A. Purchase a RUNCOBOL/TRSDOS runtime system diskette (Catalog Number 26-4704) from Radio Shack. Copy your compiled program onto this diskette, and sell this diskette to your customer. The copyright notices affixed to that diskette must not be removed or hidden from view. For each copy of your program you sell in this manner, you must purchase the 26-4704 diskette and copy your program onto it.

B. Sell your compiled program without TRSDOS and without the COBOL runtime. Instruct your customer to purchase a RUNCOBOL/TRSDOS runtime from Radio Shack.

TABLE OF CONTENTS

CHAPTER 1

THE COBOL COMPILER

1.1 Compiler Overview

The COBOL Compiler operates on a 64K byte TRS-80 Model II Microcomputer under the TRSDOS Operating System.

Once executed, the Compiler makes a single pass on the source program, generating object and listing files concurrently. Upon completion it reports compilation results on the console and returns control to TRSDOS.

Compilation always proceeds to the end of the program, regardless of the number of source errors found.

A listing of the program is generated showing the original COBOL source statements, error information, data allocation, Interactive Debug information and, optionally, a Cross Reference of all program labels and data items. This listing can be directed to the Console, the Printer and/or a disk file.

The generated object file is in a form ready for immediate execution by the COBOL Runtime. Object code is produced such that an attempt to execute an erroneous statement will terminate execution with an appropriate error message.

1.2 Device Assignments

All communication between the Compiler and the User is through the system console.

During operation, the Compiler will require one or more of the following devices:

Console	compiler command input and compiler messages
Disk	source input file
Disk	listing file (optional)
Disk	object file (optional)
Disk	COPY input file (optional)
Console	listing display (optional)
Printer	listing print (optional)

1.3 Executing the Compiler

To compile a COBOL source program, issue the following command to TRSDOS:

```
RSCOBOL filespec {options} comment
```

where:

filespec

is the file specification of the COBOL source file to be compiled; of the form:

```
filename/ext.password:d(diskette name)
```

'filename' is required.

'/ext' is an optional name-extension. When omitted, the default '/CBL' is used.

'.password' is an optional password. Note: If the file was created with a nonblank password, '.password' becomes a required field.

':d' is an optional drive specification. When omitted, the system does an automatic search, starting with drive 0.

'(diskette name)' is optional. When omitted, no disk name checking is performed.

options

allows the user to specify compiler and/or file options. Each option must be specified as shown below, separated by spaces. The left and right braces are required if any comments are present.

When no options are specified, the compiler will automatically generate an object file but no listing output.

1.3.1 Compiler Options

A=Y A=N

'A=' tells the compiler whether or not the line printer generates an automatic line-feed after each line is printed.

Specifying 'A=Y' indicates 'Yes', the line printer does generate an automatic line-feed; specifying 'A=N' indicates 'No', the line printer does not generate an automatic line-feed. Incorrect specification will cause either overprinting or double spacing.

The default is 'A=Y', the normal operating mode of Radio Shack Line Printers.

D

'D' instructs the compiler to compile all COBOL "Debug" source lines, identified by a "D" in column 7. This allows the user selective compilation of COBOL source statements.

This option has no relationship to the COBOL Runtime Interactive Debug facility and need not be specified to allow such debugging.

The default is to treat such lines as comments.

E

'E' instructs the compiler to generate an 'Error Only' listing instead of a full listing. This option is effective only when a listing has been specified (L, P and/or T options).

The listing generated will contain the page heading information, all source lines in error with their appropriate undermarks and messages, plus all summary information.

The default is not to generate an error listing.

L L=d

'L' indicates that the compiler listing is to be written to a disk file with the name of the source file and a filename-extension of '/LST'. The first available disk is used.

Specifying a drive number (L=d) indicates that the listing file is to be written to disk 'd'.

The default is not to generate a listing file.

O O=d O=N

'O' indictes that the Compiler object output is to be written to a disk file with the name of the source file and a filename-extension of '/COB'. The first available disk is used.

Specifying a drive number (O=d) indicates that the object file is to be written to disk 'd'. When omitted the first available disk is used.

'O=N' indicates that no object file is to be generated.

The default is to generate an object file on the first available disk.

P

'P' indicates that the listing is to be printed on the printer.

The default is not to print the listing.

T

'T' indicates the listing is to be displayed on the CRT (console).

The default is not to display the listing.

X

'X' indicates a cross-reference of COBOL Procedure and Data Division names is to be produced. This option is effective only when a listing has been specified (L, P or T options).

The default is not to generate a cross-reference.

1.3.2 Compiler Messages

Messages which report the compiler's status, or its ability to complete the compilation process are reported on the system console as they are detected.

```
TRS-80 Model II COBOL Compiler (RM/COBOL ver v.r)
Copyright 1980 by Tandy Corp.  Licensed from Ryan-McFarland Corp.
```

Indicates that the compiler has been loaded and has begun to compile the specified program. 'ver v.r' identifies the version (v) and revision (r) level of the compiler.

```
COMPILATION COMPLETE: eeee ERRORS, wwww WARNINGS
```

Indicates that the compilation has been completed. The values of 'eeee' and 'wwww' indicate the number of errors and warnings, respectively, identified in the source program. This message is repeated on the listing.

```
PARAMETER ERROR AT: vvvvvvvv
```

Indicates that an unrecoverable error was detected on the command to execute the compiler. 'vvvvvvvv' will identify the offending field.

The user should reenter the command with the necessary corrections.

```
COMPILATION CANCELLED
```

Compiler cancelled by operator with BREAK key.

COMPILER ERROR, NO: nnnn

An internal error has occurred which prevents continued compilation. The value of 'nnnn' identifies the condition which caused the error.

0001	Pointer overflow The user program has exceeded internal compiler pointers. Segment the program and recompile. If this problem still exists, separate programs into main program with multiple subroutines.
0002	Roll memory overflow The user program has exceeded available work space. Segment the program and recompile.
0010	Unable to locate or load a compiler overlay. Install the RSCBLnvr program overlays as described in the chapter on 'Installation Procedures.'

1.3.3 Examples

```
RSCOBOL PAYROLL {P X}
```

locates and compiles the source program PAYROLL/CBL, producing an object file (PAYROLL/COB) on the first available disk and a listing, with cross-reference, on the printer.

```
RSCOBOL MORTGAGE/SRC:1 {L=2 O=N}
```

compiles the source program MORTGAGE/SRC located on the disk in drive 1, producing a listing file (MORTGAGE/LST) on the disk in drive 2, and no object file.

1.4 The Program Listing

The compiler outputs 'source', 'allocation', and 'summary' listings if a listing device or file is specified (L, P or T options). When the 'X' option is specified, a 'cross-reference' listing is also produced.

The source listing includes a sequential line number, sentence address, source image, and interspersed diagnostics.

The allocation listing includes the address, size, order, type, and name of each identifier. The identifier names are indented to show the record structure. (The order of an identifier is the number of subscripts it requires).

The summary listing includes the number of errors, the number of warnings, and the size of the program.

The cross-reference listing includes all identifier names in alphabetical order, and the line number of each declaration, source, and destination reference. The line number is surrounded by slashes if the reference is a declaration; asteriks if the reference is a possible modification. References to all paragraphs and sections are included.

In all listings, numbers in decimal are represented as ddd...d, numbers in hexadecimal are represented as >dd...d.

1.4.1 Listing Diagnostics

Source constructs are checked for syntax and semantic errors as they are scanned. Errors may cause interruption in scanning. In this case, text is ignored until a recovery point is found and a resume message is printed. Recovery points are chosen to minimize the amount of unanalyzed text without producing irrelevent error messages. In any case, the constructs at fault are undermarked and error messages listed when the source line is printed. The error message includes either E's or W's indicating error or warning. For example:

```
004030  02  STOCK  PIC  9(16)PPP COMPUTATIONAL.
                                $
***** 1)PICTURE  *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
```

Indicates a semantic number size error but

```
005040  02  PART  PIC X(4BX(5)     SYNC.
                         $         $
***** 1)SYNTAX *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
***** 2)SCAN RESUME  *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
```

indicates a syntax error at the first undermark and a recover at the second undermark.

The number preceding the error message is the undermark number, counting from left to right. More than one message may refer to the same undermark.

Global errors such as undefined paragraph names and illegal control transfers are listed with the program summary at the end of the source listing.

1.4.2 Diagnostic Messages

ACCESS CLASH
: Nonsequential access given for sequential file.

BLANK WHEN ZERO
: BLANK WHEN ZERO clause given for nonnumeric or group item.

CLASS
: The referenced identifier is not valid in a class condition.

COPY
: COPY statement failed because of permanent error associated with the undermarked file-name.

CORRESPONDING
: The CORRESPONDING phrase cannot be used with the referenced identifier.

DATA OVERFLOW
: The data area (working-storage and literals) is larger than 65535 bytes in length.

DATA TYPE
: Context does not allow data type of the referenced identifier.

DEVICE CLASH
: Random characteristics given to nonrandom device.

DEVICE TYPE
: OPEN or CLOSE mode inconsistent with device type.

DOUBLE DECLARATION
: Multiple declaration of a file or identifier attribute.

DOUBLE DEFINITION
: Multiple definition of an identifier.

DUPLICATE
Warning only. Multiple USE procedure declared for same function or file.

FILE DECL ERROR
The referenced file-name is SELECTed and has an invalid or missing file description (FD).

FILE NAME ERROR
The referenced file-name has an invalid external file name declaration.

FILE NAME REQUIRED
File name not given as referenced in I/O verb.

FILE RECORD KEY ERROR
The referenced file-name has a RECORD KEY which is incorrectly qualified or is not defined as a data item of the category alphanumeric within a record description entry associated with that file name.

FILE RECORD SIZE ERROR
The referenced file-name has a declared record size which conflicts with the actual data record descriptions or is a relative organization file with variable length records.

FILE RELATIVE KEY ERROR
The referenced file-name has a RELATIVE KEY which is incorrectly qualified, is defined in a record description associated with that file-name, or is not defined as an unsigned integer.

FILE STATUS ERROR
The referenced file-name has a status item which is incorrectly qualified, is not defined in the WORKING-STORAGE SECTION, or is not a two-character alphanumeric item.

FILE TYPE
Access or organization of file conflicts with undermarked statement.

FILLER LEVEL
A non-elementary FILLER item is declared.

GROUP CLASH
USAGE or VALUE clause of group member conflicts with same clause for group.

GROUP VALUE CLASH
Warning Only. An item subordinate to a group with the VALUE IS clause is described with the SYNCHRONIZED, JUSTIFIED, or USAGE (other than USAGE IS DISPLAY) clause.

IDENTIFIER
Identifier reference is incorrectly constructed or the identifier has an invalid or double definition.

ILLEGAL ALTER
An ALTER statement references an unalterable paragraph or violates the rules of segmentation.

ILLEGAL PERFORM
A PERFORM statement reference undefined or incorrectly qualified paragraph or the reference violates the rules of segmentation.

INVALID ID
The referenced identifier was not successfully defined.

INVALID PARAGRAPH
Context does not allow section name.

JUSTIFY
JUSTIFY clause given in conflict with other attributes.

KEY REQUIRED
Relative key not declared for random access relative file or record key not declared for indexed file.

LABEL
Presence or absence of label record conflicts with device standards.

LEVEL
Level-number given is invalid either intrinsically or because of position within a group.

LINKAGE
An identifier in the USING clause of the PROCEDURE title is not a linkage item or a statement references a linkage item not subordinate to an identifier in the USING clause of the PROCEDURE title.

LITERAL VALUE
Literal value given is incorrect in context.

MOVE
Operands of MOVE verb specify an invalid move.

MUST BE INTEGER
Context requires decimal integer.

MUST BE PROCEDURE
Context requires procedure name either as reference or definition, or the reference must be a nondeclarative procedure-name.

MUST BE SECTION
Context requires procedure-name to be section.

NESTING
Illegal nesting of condition that is not an IF condition.

NOT IN REDEFINE
VALUE IS clause given in REDEFINES item.

OCCURS
OCCURS clause given at invalid level or after three have been given for the same item.

OCCURS DEPENDING ERROR
The referenced object of a DEPENDING phrase has not been defined correctly.

OCCURS-VALUE CLASH
VALUE IS and OCCURS in effect for the same item.

PICTURE
Invalid PICTURE syntax.

PICTURE-BWZ CLASH
Zero suppression and BLANK WHEN ZERO cannot be in effect for the same item.

PICTURE-USAGE CLASH
USAGE clause or implied usage conflicts with usage implied by picture.

PROCEDURE INDEPENDENCE
PERFORM given for procedures in independent segments not in the current segment.

PROGRAM OVERFLOW
The instruction area is larger than 32767 bytes in length.

RECORD KEY
Record key declared for other than an indexed organization file or a START statement KEY phrase references a data item not aligned on the declared key's leftmost byte.

RECORD REQUIRED
Context requires record name.

REDEFINES
REDEFINES given within an OCCURS or not redefining the last allocated item.

REDEFINES ERROR

The referenced data-name redefines an item which does not have the same number of character positions and is not level 01.

REFERENCE INVALID

Reference given is not valid in context.

RELATION

Operands of relation test are incompatible.

RELATIVE KEY

Relative key declared for other than a relative organization file or a START statement KEY pharase references a data item other than the declared key.

RESERVED WORD CONFLICT

A COBOL reserved word or symbol is given where a user word is required. In the summary this is only a warning about an ANSI COBOL reserved word that is not an implemented COBOL reserved word.

SCAN RESUME

Warning only. Scanning was terminated at previous error message and resumes at undermarked character.

SECTION CLASH

A VALUE IS clause appears in the FILE or LINKAGE section.

SEGMENT

Warning only. Segment number given in an independent segment is not the same as the current segment or the number of a new independent segment. The current segment number is used.

SEPARATOR

Warning only. Redundant punctuation or a separator is not followed by the required space.

SIGN

SIGN clause given in conflict with usage and picture.

SIZE

Warning only. Size of data referenced not correct for context.

SIZE ERROR

Declared size of record conflicts with present reference.

SUBSCRIPT

Incorrect number of subscripts or indices for a reference.

SYNC
: Synchronized clause given for a group item

SYNTAX
: Incorrect character or reserved word given for context.

UNDEFINED
: File referenced in FD entry was not defined.

UNDEFINED DECLARATIVE PROCEDURE
: A declarative statement references a procedure not defined within the DECLARATIVES.

UNDEFINED PROCEDURE
: A GO TO statement references an undefined or incorrectly qualified paragraph.

USE REQUIRED
: A DECLARATIVES section must begin with a USE statement.

USING COUNT
: Warning only. The item count in the USING list of a CALL statement is different from that of the first reference to the same program name.

VALUE ERROR
: Value given in VALUE IS required truncation of nonzero digits.

VALUE
: VALUE IS clause given in conflict with other declared attributes.

VARIABLE RECORD
: Warning only. The INTO phrase is not allowed with variable size records.

CHAPTER 2

THE COBOL RUNTIME

2.1 Runtime Overview

The COBOL runtime operates on a 64K byte TRS-80 Model II Microcomputer under the TRSDOS Operating System.

Once invoked, the runtime loads and executes the compiled object program, automatically loading any required segments. Concurrently, it allocates memory for file buffers, and CALLed COBOL and Assembly Language subprograms. Upon completion appropriate messages are displayed and control is returned to the operating system.

2.2 Device Assignments

All communication between Runtime and the User is through the system console.

During operation the Runtime will require one or more of the following devices:

Console	runtime command input, Interactive Debug command input, and runtime messages.
Console	ACCEPT and DISPLAY, and Interactive Debug display.
Printer	PRINT output, if required.

NOTE: For PRINT output, the device name "PRINTER" must be specified in the SELECT statement; i.e,

```
SELECT filename, ASSIGN to PRINT, "PRINTER".
```

2.3 Executing the Compiled Program

To execute a compiled COBOL object program, issue the following command to TRSDOS:

```
RUNCOBOL filespec {options} comment
```

where:

filespec

is the specification of the compiled COBOL object file to be executed of the form:

```
filename/ext.password:d(diskette name)
```

'filename' is required.

'/ext' is an optional name-extension. When omitted the default '/COB' is used.

'.password' is an optional password. Note: If the file was created with a nonblank password, '.password' becomes a required field.

':d' is an optional drive specification. When omitted the system does an automatic search, starting with drive 0.

'(diskette name)' is optional. When omitted no disk name checking is performed.

options

allows the user to specify runtime options. Each option must be specified as shown below, separated by spaces. The left and right braces are required if any comments are present.

When no options are specified, the runtime will execute the User's program without Interactive Debug, with all switches set to 0, using all of available memory.

2.3.1 Runtime Options

A=Y A=N

'A=' tells the runtime whether or not the line printer generates an automatic line-feed after each line is printed (see WRITE... ADVANCING ZERO LINES... statement below).

Specifying 'A=Y' indicates 'Yes', the line printer does generate an automatic line-feed; specifying 'A=N' indicates 'No'; the line printer does not generate an automatic line-feed. Incorrect specification will cause either overprinting or double spacing.

The default is 'A=Y', the normal operating mode of Radio Shack Line Printers.

D

'D' invokes the RSCOBOL Interactive Debug package. See RSCOBOL Interactive Debug discussion, below, for operating instructions.

The default is not to invoke Interactive Debug.

S=nn..n

'S' sets (or resets) the value of SWITCHES in the COBOL program.

Each 'n' is a switch value, 0 for off, 1 for on, numbered 1 to 8, left to right. Trailing zeroes need not be specified.

The default is to set all switches off (0).

T=hhhh

'T' sets the top of available memory to a value different from the highest available address. This is used to protect assembly language user subroutines, all of which must be created to load above the hexadecimal address 'hhhh'.

The default is to use all available memory.

2.3.2 Runtime Messages

Messages which report the runtime's status, or its ability to execute the COBOL program, are reported on the system console as they are detected.

```
TRS-80 Model II COBOL Runtime (RM/COBOL ver v.r)
Copyright 1980 by Tandy Corp. Licensed from Ryan-McFarland Corp.
```

Indicates that the runtime has been loaded and has begun to execute the specified program. 'ver v.r' identifies the version (v) and revision (r) level of the runtime.

```
COBOL STOP RUN AT xxyyyy IN nnnnnn
```

This is the normal termination message of a program.

'xxyyyy' identifies the overlay (xx) and statement address (yyyy) where the program terminated. 'nnnnnn' are the first six characters of the PROGRAM-ID.

If Debug was invoked on the command line, an 'S' Debug command may be used to cause Debug to exit to the operating system.

```
COBOL STOP literal AT xxyyyy IN nnnnnn
CONTINUE (Y/N)?
```

This message indicates that a STOP 'literal' statement has been encountered. 'xxyyyy' identifies the overlay (xx) and statement address (yyyy) where the program terminated. 'nnnnnn' are the first six characters of the PROGRAM-ID.

Responding with a 'Y' will be the equivalent of a "pause" statement, returning control to the next COBOL statement.

An 'N' response will cause all program files to be closed and control will be returned to the operating system.

2.3.3 Examples

```
RUNCOBOL PAYROLL {S=1011}
```

locates, loads, and executes the compiled COBOL program PAYROLL/COB; and sets the value of SWITCHES 1, 3, and 4 'on', all others 'off'.

```
RUNCOBOL MORTGAGE/TST:2 {D}
```

loads the compiled COBOL program MORTGAGE/TST from drive 2 along with the Interactive Debug package. Control is passed directly to Debug.

2.4 Runtime Diagnostics

Diagnostic messages are displayed on the console if an internal error occurs, or if an I/O error occurs that was not, or could not, be processed by an appropriate USE procedure.

If Debug was invoked, Debug will be entered to allow examination of program data values; otherwise, control will return to the operating system.

COBOL error AT xxyyyy IN nnnnnn

> Indicates an internal error condition has occurred, where 'error' identifies the error condition. 'xxyyyy' identifies the overlay (xx) and statement address (yyyy) where the program terminated. 'nnnnnn' are the first six characters of the PROGRAM-ID.

COBOL filename IO ERROR = cc AT xxyyyy IN nnnnnn

> Identifies that an abnormal I/O condition, 'cc' has caused the program to be aborted. 'xxyyyy' identifies the overlay (xx) and statement address (yyyy) where the program terminated. 'nnnnnn' are the first 6 characters of the PROGRAM-ID.
>
> The I/O error 'cc' has a different meaning depending on whether the file's organization is sequential, relative or indexed.
>
> Sequential Files:
>
> 10 AT END.
> The sequential READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.
>
> 30 PERMANENT ERROR.
> The input-output statement was unsuccessfully executed as the result of an input-output error, such as data check parity error, or transmission error. May also indicate attempted execution of an instruction not implemented in the runtime (REWRITE to a variable length record (VLR) file; CLOSE REEL).
>
> 34 PERMANENT ERROR BOUNDARY VIOLATION.
> The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file.

90 INVALID OPERATION.
An attempt has been made to execute a READ, WRITE, or REWRITE statement that conflicts with the current open mode or a REWRITE statement was not preceded by a successful READ statement.

91 FILE NOT OPENED.
An attempt has been made to execute a DELETE, READ, START, UNLOCK, WRITE, REWRITE or CLOSE statement on a file which is not currently open.

92 FILE NOT CLOSED.
An attempt has been made to execute an OPEN statement on a file which is currently open.

93 FILE NOT AVAILABLE.
An attempt has been made to execute an OPEN statement for a file closed with LOCK.

94 INVALID OPEN.
An attempt has been made to execute an OPEN statement for a file with no external correspondence or a file having inconsistent parameters.

95 INVALID DEVICE.
An attempt has been made to execute a CLOSE REEL statement, or to execute an OPEN statement for a file which is assigned to a device in conflict with the externally assigned device. Valid combinations are:

Program Assignment	External Assignment
RANDOM	Disk
INPUT	Disk
OUTPUT	Disk
PRINT	Disk, line printer
INPUT-OUTPUT	Disk

96 UNDEFINED CURRENT RECORD POINTER STATUS.
An attempt has been made to execute a READ statement after the occurrence of an unsuccessful READ statement without an intervening successful CLOSE and OPEN.

97 INVALID RECORD LENGTH.
An attempt has been made to execute a REWRITE statement when the new record length is different from that of the record to be rewritten, or to OPEN a file that was defined with a maximum record length different from the externally defined maximum record length, or to execute a WRITE statement that specifies a record with a length smaller than the minimum or larger than the maximum record size.

Relative and Indexed Files:

10 AT END.
The Format 1 READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

21 SEQUENCE ERROR FOR A SEQUENTIALLY ACCESSED INDEXED FILE.
The ascending sequence requirement of successive record key values has been violated or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

22 DUPLICATE KEY VALUE.
An attempt has been made to WRITE a record that would create a duplicate key on a file that does not allow duplicates.

23 NO RECORD FOUND.
An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

24 BOUNDARY VIOLATION.
An attempt has been made to WRITE beyond the externally-defined boundaries of a file.

30 PERMANENT ERROR.
The input-output statement was unsuccessfully executed as the result of an input-output error, such as data check, parity error, or transmission error.

90 INVALID OPERATION.
An attempt has been made to execute a DELETE, READ, REWRITE, START, or WRITE statement which conflicts with the current open mode of the file or a sequential access DELETE or REWRITE statement not preceded by a successful read statement.

91 FILE NOT OPENED.
An attempt has been made to execute a CLOSE, DELETE, READ, REWRITE, START, UNLOCK, or WRITE statement on a file which is not in an open mode.

92 FILE NOT CLOSED.
An attempt has been made to execute an OPEN statement on a file that is currently open.

93 FILE NOT AVAILABLE.
An attempt has been made to execute an OPEN statement on a file closed with LOCK.

94 INVALID OPEN. OR PROGRAM opened as I-O & No file exists OR EXISTS in WRONG FORMAT OR PROGRAM ABORTED + DIDN'T CLOSE file.
An attempt has been made to execute an OPEN statement for a file with no external correspondence or a file having inconsistent parameters.
OR Simply means File Name must have format XX-XX/XXX

95 INVALID DEVICE.
An attempt has been made to execute an OPEN statement on a file whose device description conflicts with the externally assigned device. The device must be RANDOM and the external correspondence must be a disk.

96 UNDEFINED CURRENT RECORD POINTER.
An attempt has been made to execute a Format 1 READ statement when the current record pointer has an undefined state. This can occur only as the result of a preceding unsuccessful READ or START statement.

97 INVALID RECORD LENGTH.
An attempt has been made to execute a REWRITE statement and the new record length is different from that of the record to be rewritten, or to OPEN a file that was defined with a maximum record length different from the externally defined maximum record length, or to execute a WRITE statement that specifies a record with a length smaller than the minimum or larger than the maximum record size.

98 INVALID INDEX.
An input-output statement on an indexed organization file was unsuccessful as a result of invalid data in the index. This can result if the externally assigned file is not an index organization file or if an undetected input-output error has occurred.

2.5 File System Considerations

Three types of files are supported by COBOL: sequential, relative (random), and indexed sequential. These files exist on the disk as standard Model II TRSDOS disk files, consisting of either fixed length records (FLR), or variable length records (VLRs). While the user will not typically need this information to execute COBOL programs, he is referrred to the Technical Information Section of the TRS-80 Model II Disk Operating System Reference Manual if further information is desired.

Files are specified in the user's program SELECT statement in a manner consistent with the TRSDOS filespec, of the form:

```
filename/ext.password:d(diskette name)
```

where:

'filename' is required.

'/ext' is an optional name-extension.

'.password' is an optional password. Note: If the file was created with a nonblank password, '.password' becomes a required field.

':d' is an optional drive specification. When omitted the system does an automatic search, starting with drive 0.

'(diskette name)' is optional. When omitted no disk name checking is performed.

2.5.1 COBOL Sequential Files

COBOL sequential files consist of a serially accessible set of 'logical' records. These 'logical' records may exist on the disk as either variable length (VLR) or fixed length (FLR) records.

COBOL sequential files that are 'created' by a COBOL program (i.e., do not already exist), are created as variable length (VLR) records. Each 'logical' record within the file can have a maximum length of 255 bytes.

COBOL sequential files that were 'created' by other than a COBOL program can have either fixed length (FLR) or variable length (VLR) records. In this case, the COBOL Runtime will process the records as presented. Each 'logical' record can have a maximum length of 255 bytes (VLR) or 254 bytes (FLR).

NOTE: The REWRITE statement is not valid for variable length records (VLR's), and will generate an appropriate error message if executed.

2.5.2 COBOL Relative Files

COBOL relative files are addressable randomly by 'logical' record number. These files can only exist on the disk as fixed length (FLR) records.

COBOL relative file 'logical' records are internally formatted, and can be created and/or accessed only by COBOL programs. Each 'logical' record can have a maximum length of 254 bytes.

COBOL relative files are dynamically allocated or extended as required by TRSDOS. If the user desires to preallocate the file, allocate it using the CREATE program of TRSDOS, with options shown below:

```
TYPE   = F       (fixed length records)

LRL    = 256     (record length = 256 bytes)

                 (record length +2) * (max # records)
NRECS =          ------------------------------------
                                256
```

2.5.3 COBOL Indexed Files

COBOL indexed files are created and maintained by the COBOL runtime; implemented on the disk using TRSDOS fixed length (FLR) records.

COBOL indexed files are internally formatted, and can be created and/or accessed only by COBOL programs. Each 'logical' record can have a maximum length of 4096 bytes.

Indexed files contain an index structure for each key specified interspersed with the data records. The use of ALTERNATE KEYS can cause a geometric increase in the time required to create the file; however, access time will be relatively constant throughout the file.

COBOL indexed files are dynamically allocated or extended as required by TRSDOS. If the user desires to preallocate the file, allocate it as shown below: By using TRSDOS "CREATE" COMMAND

```
          TYPE = F         (fixed length records)

          LRL = 256        (record length = 256 bytes)

                           (L + (2 * T)) * R
          NRECS =          -----------------
                                   8
```

where:

```
          L = number of 32 byte records required to cover the
              maximum length record

          K = 8 * number of key fields in record

          S = sum total of all key field lengths

          T = number of 32 byte records required to cover (K+S)

          R = maximum number of records expected in file
```

Notes: This calculation provides an approximation for preallocating the file. TRSDOS will extend the file as necessary to the physical limits of the disk.

2.5.4 COBOL Label Processing

The COBOL language allows the specification of the existance, and processing, of Label records on file type devices.

TRSDOS provides automatic maintenance and validation of file specifications by name and file type. No additional Label processing is performed unique to COBOL programs or files.

References to Label processing in the file description entry (FD), OPEN statement, and CLOSE statement, are checked for correct syntax by the compiler. They are largely ignored by the runtime except that appropriate error codes will be returned, and any applicable USE procedures will be executed.

2.6 Runtime Memory Usage

The TRSDOS Operating System occupies lower memory from location 0000H to 02800H. The COBOL Runtime is loaded starting at 02800H. The remaining memory is allocated as follows:

> The main COBOL object program is loaded immediately behind the COBOL Runtime. Space for COBOL overlays (SECTIONS greater than 50) are included in this area.
>
> Additional COBOL programs are loaded behind this main program as they are CALLed (See the CALL statement below).
>
> Assembly Language programs are loaded in high memory at the address they were assigned at 'DUMP' times (See Runtime 'T=hhhh' option).
>
> File buffers are dynamically allocated from high memory downward, when OPENed, deallocated (space recovered for use by other files) when CLOSEd.

CHAPTER 3

INTERACTIVE DEBUG

3.1 Debug Overview

COBOL Interactive Debug is dynamically loaded when the user specifies the 'D' option on the RUNCOBOL statement. Debug is then given control and supervises the execution of the user's program.

Interactive Debug is loaded directly behind COBOL Runtime, requiring approximately 1000 bytes.

3.2 User Interaction and Display

All Debug commands, and all resultant displays, are through the system console.

Debug will request command input by a prompt of the form

```
     nnnnnn xxyyyy
```

where 'nnnnnn' are the first 6 characters of PROGRAM-ID, 'xx' is the overlay number, and 'yyyy' is the hexadecimal location within the specified overlay that will be executed next.

The values of 'xx' and 'yyyy' are taken directly from the Debug column in the source listing for program 'nnnnnn'.

3.3 Debug Commands

All commands are specified by a single character, optionally followed by one or more arguments. Optional fields are shown surrounded by brackets; the brackets are never entered. All numeric arguments are in hexadecimal unless otherwise noted.

Invalid commands will be rejected with 'ERROR' displayed; corrected input will be requested with a reprompt.

A[xx]yyyy[,nnnnnn]   Address stop.

Executes object instructions until overlay number 'xx' and location 'yyyy' in program nnnnnn is to be executed. Debug will regain control immediately prior to the execution of the specified COBOL sentence, and request further command input.

If 'xx' is specified, 'yyyy' must be fully four hexadecimal digits; if 'xx' is not specified, then leading zeros are not required for 'yyyy'. If 'nnnnnn' is omitted, it is assumed to be the first six characters of the program-id of the currently executing program.

S[n] Single step sentence.

Execute 'n' COBOL sentences and return to the debug monitor.

The decimal argument 'n' specifies the number of COBOL sentences to be executed before returning the Debug.

Dxxxx,yyyy[,tttt] Dump by type.

Display the COBOL data item starting at hexadecimal location 'xxxx' of decimal length 'yyyy' and type 'tttt'. The values for 'xxxx', 'yyyy', and 'tttt' are directly from the first three columns of the allocation map. 'tttt' may be one of the following:

NSU	NPS
NSS	ABS
NCU	ANS
NCS	GRP
NBS	ANSE
NSE	HEX (hexadecimal)

Dump Display has the format:

```
xxxx tttt dddd....
```

where dddd = data in the specified format

Note: Only items in the currently executing program can be displayed. This does not include linkage items.

Q Quit Execution.

Terminate Debug and force an immediate STOP RUN. Enter 'S' to return to TRSDOS.

E Exit

Exit the Debugger. Continue normal execution as if the ebugger had not been invoked on the command line.

CHAPTER 4

SYSTEM CONSIDERATIONS

4.1 The ACCEPT and DISPLAY Statements

The ACCEPT and DISPLAY statements support the transfer of data between the console and the User's program data area. These statements allow the specification of general phrases which may not be supported on every CRT.

Phrases which are not supported will compile correctly, but will be ignored at runtime, causing no operation to take place. The phrases which are not supported under the Tandy Model II are:

ACCEPT....HIGH, LOW, BLINK.

DISPLAY....HIGH, LOW, BLINK.

The ON EXCEPTION phrase of the ACCEPT statement is executed when an invalid character is entered. Invalid characters include the valid control characters (CNTR/n) below 020H, and non-ASCII characters above and including 080H.

When an invalid character is entered, its ASCII equivalent is placed in the specified data-name and the ON EXCEPTION phrase is executed. To determine which control character was entered, define the data-name as USAGE COMPUTATIONAL-1 and compare for its ASCII value.

Certain keys affect the operation of the ACCEPT statement, including:

BACKSPACE	Erases the current character and moves the cursor back one position.
ESC	Backspace to the beginning of the field, erasing all characters in the field.

4.2 The CALL Statement

When 'CALLed' the first time, COBOL and Assembly Language programs are loaded by Runtime and entered at their initial location. These 'called' programs remain in memory as long as the 'calling' program is active; i.e., has not EXITed. Therefore, subsequent CALLs from the 'calling' program will enter the 'called' program directly, without requiring the 'called' program to be reloaded.

Once the 'calling' program has EXITed, all related 'called' programs are discarded and will be reloaded if subsequently CALLed by any program, including the previous 'calling' program. Regardless of the sequence of 'called' and 'calling' programs, all related files not explicitly closed are forced closed by the interface upon EXIT from a given 'called' program.

COBOL programs that are to be CALLed must have been previously compiled. The default filename-extension for a program name in a CALL statement is '/COB'. A compiled COBOL program will have the required extension. If the extension used is not '/COB' , then it must be specified in the CALL statement.

Assembly language programs that are to be CALLed must be in TRSDOS LOAD command format as created by DUMP, with a filename extension other than '/COB'. Assembly language programs must reside in high memory, and the 'T=nnnn' option must be specified on the Runtime command line to protect all memory required by the routine. The user is responsible for ensuring that the assembler programs do not interfere with each other.

Assembly language programs are loaded and reused while the 'calling' program resides in memory. If the COBOL 'calling' program is reloaded in memory, then the assembler program will again be reloaded when it is called.

At entry time to an assembly-language routine register IX points to the parameter list defined by the USING clause of the CALL statement. The first word on the list contains the number of bytes in the list. Subsequent words are addresses of the USING arguments: e.g., if the length word specifies 6 bytes, there are 2 addresses following the length word. For example:

```
     (IX) =>  DW  Argument List Length (n * 2 + 2)
              DW  USING Argument 1
              DW  USING Argument 2
              .
              .
              DW  USING Argument n
```

The format of each argument depends on its dataname PICTURE definition; see the COBOL Language Manual, 'the PICTURE Clause'.

At exit time from an assembler routine, register A may be set non-zero to request a STOP RUN.

4.3 The COPY Statement

The COPY statement provides the facility to copy (include) COBOL source text from a user-specified file into the source program. The complete file is copied into the program, without change, at the location of the COPY statement.

The file to be copied is identified in the COBOL program by the statement

```
        COPY  filename
```

or

```
        COPY  "filename/ext.password:d(diskette name)"
```

where:

'filename' is required.

'/ext' is an optional name-extension. When omitted the default '/CBL: is used.

'.password' is an optional password. Note: If the file was created with a nonblank password, '.password' becomes a required field.

':d' is an optional drive specification. When omitted the system does an automatic search, starting with drive 0.

'(diskette name)' is optional. When omitted no disk name checking is performed.

A filename consisting only of letters and numbers (first character must be letter) can be written without surrounding quotes. All other forms must be surrounded by quotes.

Examples:

```
    IDENTIFICATION DIVISION.
        COPY     STDID.
    ENVIRONMENT DIVISION.
        COPY     "STDENVIR/TST".
    DATA DIVISION.
        COPY     "STDDATA/CBL:1".
```

4.4 The WRITE...ADVANCING ZERO...Statement

The sequential WRITE statement allows control of the vertical positioning of each line on the printed page with the ADVANCING phrase.

The ... ADVANCING ZERO LINE(s) ... phrase allows overprinting on those print devices which support this feature. In all cases, the phrase will compile correctly, but may operate as though ...ADVANCING 1 LINE... was specified.

Standard Radio Shack Line Printers automatically advance after each line is printed. Therefore, the ...ADVANCING ZERO LINES... phrase will execute as ... ADVANCING 1 LINE. The Compiler and Runtime defaults (see 'A=Y' options) to standard Radio Shack Line Printer operation. For use with line printers that allow overprinting, specify the 'A=N' option on the command line when executing the compiler or runtime.

CHAPTER 5

INSTALLATION PROCEDURES

Installation of RSCOBOL requires only that the object modules be copied from the Software Distribution Disk to the appropriate user diskette. NOTE: 'nn' indicates the current release level, i.e., release 1.2 will be '12'.

The modules required to compile COBOL programs are:

```
RSCOBOL
RSCBL2nn/OBJ
RSCBL3nn/OBJ
RSCBL4nn/OBJ
```

The modules required to execute compiled COBOL programs are:

```
RUNCOBOL
RSCBLDnn/OBJ
```

As with all Software Distribution Disks, the user should save it in a secure location in case re-creation is required.

APPENDIX A

SAMPLE PROGRAMS

Both of the following two programs are included on the RSCOBOL diskette.

The first program, CALCXMPL, demonstrates keyboard input, video output, and basic arithmetic operation under RSCOBOL. It is a working program.

The second program, ERRXMPL, is included solely to illustrate how the compiler handles various errors. It is not a working program.

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                3/28/80  16.28.56  PAGE     1
SOURCE FILE: CALCXMPL                              OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B............................................................ID...

   1         000000 IDENTIFICATION DIVISION.
   2         000010 PROGRAM-ID.
   3         000020     CALCULATOR.
   4         000030
   5         000040 ENVIRONMENT DIVISION.
   6         000050 CONFIGURATION SECTION.
   7         000060 SOURCE-COMPUTER.      RMC.
   8         000070 OBJECT-COMPUTER.      RMC.
   9         000080
  10         000090 DATA DIVISION.
  11         000100 WORKING-STORAGE SECTION.
  12         000110 77  RESULT              PICTURE S9(9)V9(9) VALUE ZERO.
  13         000120 77  OPERAND-1           PICTURE S9(9)V9(9).
  14         000130 77  OPERAND-2           PICTURE S9(9)V9(9).
  15         000140 77  WAIT-CHAR           PICTURE  X.
  16         000150 01  GREETING.
  17         000160     02  FILLER          PICTURE X(18)
  18         000170           VALUE "CALCULATOR PROGRAM".
  19         000180 01  OPERATION-MESSAGE.
  20         000190     02  FILLER          PICTURE X(37)
  21         000200           VALUE "CHOOSE YOUR OPERATION (+,-,*,/) = ".
  22         000210 01  OPERATOR            PICTURE X(2).
  23         000220 01  RESULT-MESSAGE.
  24         000230     02  FILLER          PICTURE X(12)
  25         000240           VALUE "RESULT IS = ".
  26         000250     02  RESULT-EDITED   PICTURE -(9)9.9(9).
  27         000260     02  FILLER          PIC X(4) VALUE SPACES.
  28         000270     02  OVERFLOW-FIELD  PIC X(8) VALUE SPACES.
  29         000280 01  WAIT-MESSAGE.
  30         000290     02  FILLER          PICTURE X(36)
  31         000300           VALUE "HIT NEWLINE TO CONTINUE (Q TO QUIT) ".
  32         000310 01  OPERAND-1-MESSAGE.
  33         000320     02  FILLER          PICTURE X(12)
  34         000330           VALUE "OPERAND-1 = ".
  35         000340 01  OPERAND-2-MESSAGE.
  36         000350     02  FILLER          PICTURE X(12)
  37         000360           VALUE "OPERAND-2 = ".
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)             3/28/80  16.28.56  PAGE     2
SOURCE FILE: CALCXMPL                           OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B.....................................................ID.....

  38        000370/    EJECT
  39        000380 PROCEDURE DIVISION.
  40  >0000 000390 RESIDENT SECTION 1.
  41  >0000 000400 NOT-START.
  42  >0000 000410     GO TO DISPLAY-GREETING.
  43  >0004 000420 RE-TRY.
  44  >0004 000430     DISPLAY OPERATION-MESSAGE, LINE 2, ERASE.
  45  >000C 000440     ACCEPT OPERATOR, POSITION 0, PROMPT, ECHO.
  46  >0014 000450     IF OPERATOR EQUAL  "+ " GO TO ADDITION.
  47  >001C 000460     IF OPERATOR EQUAL  "- " GO TO SUBTRACTION.
  48  >0024 000470     IF OPERATOR EQUAL  "* " GO TO MULTIPLICATION.
  49  >002C 000480     IF OPERATOR EQUAL  "/ " GO TO DIVI-SION.
  50  >0034 000490     IF OPERATOR EQUAL  "Q " GO TO END-RUN.
  51  >003C 000500     GO TO RE-TRY.
  52  >003E 000510 DISPLAY-RESULT.
  53  >003E 000520     MOVE RESULT TO RESULT-EDITED.
  54  >0042 000530     DISPLAY RESULT-MESSAGE.
  55  >0046 000540     MOVE ZERO TO RESULT.
  56  >004A 000550     MOVE SPACES TO OVERFLOW-FIELD.
  57  >0050 000560 WAIT-ENTRY.
  58  >0050 000570     DISPLAY WAIT-MESSAGE.
  59  >0054 000580     ACCEPT WAIT-CHAR, POSITION 0, PROMPT, ECHO.
  60  >005C 000590     IF WAIT-CHAR EQUAL  "Q" GO TO END-RUN.
  61  >0064 000600     GO TO RE-TRY.
  62  >0066 000610 GET-OPERANDS.
  63  >0066 000620     DISPLAY OPERAND-1-MESSAGE, LINE 4.
  64  >006C 000630     ACCEPT OPERAND-1, LINE 4, POSITION 13, SIZE 10,
  65        000640         PROMPT, CONVERT.
  66  >0078 000650     MOVE OPERAND-1 TO RESULT-EDITED.
  67  >007C 000660     DISPLAY RESULT-EDITED, LINE 4, POSITION 13.
  68  >0084 000670     DISPLAY OPERAND-2-MESSAGE.
  69  >0088 000680     ACCEPT OPERAND-2 , LINE 5, POSITION 13, SIZE 10,
  70        000690         PROMPT, CONVERT.
  71  >0094 000700     MOVE OPERAND-2 TO RESULT-EDITED.
  72  >0098 000710     DISPLAY RESULT-EDITED, LINE 5, POSITION 13.
  73  >00A2 000720 END-RUN.
  74  >00A2 000730     EXIT PROGRAM.
  75  >00A6 000740 STOP-RUN.
  76  >00A6 000750     STOP RUN.
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)              3/28/80  16.28.56  PAGE     3
SOURCE FILE: CALCXMPL                             OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B.......................................................ID....

  77         000760/    EJECT
  78>0100A8 000770 OVERLAY-ADDITION SECTION 51.
  79>0100A8 000780 ADDITION.
  80>0100A8 000790     PERFORM GET-OPERANDS.
  81>0100AA 000800     ADD OPERAND-1    OPERAND-2 GIVING RESULT
  82        000810         ON SIZE ERROR MOVE "OVERFLOW" TO OVERFLOW-FIELD.
  83>0100B8 000820     GO TO DISPLAY-RESULT.
  84        000830
  85>0200A8 000840 OVERLAY-SUBTRACTION SECTION 52.
  86>0200A8 000850 SUBTRACTION.
  87>0200A8 000860     PERFORM GET-OPERANDS.
  88>0200AA 000870     SUBTRACT OPERAND-2 FROM OPERAND-1 GIVING RESULT
  89        000880         ON SIZE ERROR MOVE "OVERFLOW" TO OVERFLOW-FIELD.
  90>0200B8 000890     GO TO DISPLAY-RESULT.
  91        000900
  92>0300A8 000910 OVERLAY-MULTIPLICATION SECTION 53.
  93>0300A8 000920 MULTIPLICATION.
  94>0300A8 000930     PERFORM GET-OPERANDS.
  95>0300AA 000940     MULTIPLY OPERAND-1 BY OPERAND-2 GIVING RESULT
  96        000950         ON SIZE ERROR MOVE "OVERFLOW" TO OVERFLOW-FIELD.
  97>0300B8 000960     GO TO DISPLAY-RESULT.
  98        000970
  99>0400A8 000980 OVERLAY-DIVISION SECTION 54.
 100>0400A8 000990 DIVI-SION.
 101>0400A8 001000     PERFORM GET-OPERANDS.
 102>0400AA 001010     DIVIDE OPERAND-1 BY OPERAND-2 GIVING RESULT ROUNDED
 103        001020         ON SIZE ERROR MOVE "OVERFLOW" TO OVERFLOW-FIELD.
 104>0400BA 001030     GO TO DISPLAY-RESULT.
 105        001040
 106>0500A8 001050 OVERLAY-DISPLAY-GREETING SECTION 98.
 107>0500A8 001060 DISPLAY-GREETING.
 108>0500A8 001070      DISPLAY GREETING.
 109>0500AC 001080      GO TO WAIT-ENTRY.
 110        001090
 111        001100 END PROGRAM.
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                  3/28/80  16.28.56 - PAGE      4
SOURCE FILE: CALCXMPL                                 OPTION LIST: (L O T X

 ADDRESS  SIZE DEBUG ORDER TYPE                       NAM

  >0004     19  NSS     0   NUMERIC SIGNED            RESULT

  >0018     19  NSS     0   NUMERIC SIGNED            OPERAND-1

  >002C     19  NSS     0   NUMERIC SIGNED            OPERAND-2

  >0040      1  ANS     0   ALPHANUMERIC              WAIT-CHAR

  >0042     18  GRP     0   GROUP                     GREETING

  >0054     37  GRP     0   GROUP                     OPERATION-MESSAGE

  >007A      2  ANS     0   ALPHANUMERIC              OPERATOR

  >007C     44  GRP     0   GROUP                     RESULT-MESSAGE
  >0088     20  NSE     0   NUMERIC EDITED             RESULT-EDITED
  >00A0      8  ANS     0   ALPHANUMERIC               OVERFLOW-FIELD

  >00A8     36  GRP     0   GROUP                     WAIT-MESSAGE

  >00CC     12  GRP     0   GROUP                     OPERAND-1-MESSAGE

  >00D8     12  GRP     0   GROUP                     OPERAND-2-MESSAGE

READ ONLY BYTE SIZE =         >01BE

READ/WRITE BYTE SIZE =        >00EC

OVERLAY SEGMENT BYTE SIZE = >002E

TOTAL BYTE SIZE =             >02D8

    0 ERRORS

    0 WARNINGS
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                    3/28/80  16.28.56  PAGE     5
SOURCE FILE: CALCXMPL                                  OPTION LIST: (L O T X

CROSS REFERENCE                 /DECL/ *DEST

ADDITION                         0046  /0079/
DISPLAY-GREETING                 0042  /0107/
DISPLAY-RESULT                  /0052/  0083   0090   0097   0104
DIVI-SION                        0049  /0100/
END-RUN                          0050   0060  /0073/
GET-OPERANDS                    /0062/  0080   0087   0094   0101
GREETING                        /0016/  0108
MULTIPLICATION                   0048  /0093/
NOT-START                       /0041/
OPERAND-1                       /0013/ *0064*  0066   0081  *0088*  0095   0102
OPERAND-1-MESSAGE               /0032/  0063
OPERAND-2                       /0014/ *0069*  0071   0081   0088  *0095*  0102
OPERAND-2-MESSAGE               /0035/  0068
OPERATION-MESSAGE               /0019/  0044
OPERATOR                        /0022/ *0045*  0046   0047   0048   0049   0050
OVERFLOW-FIELD                  /0028/ *0056* *0082* *0089* *0096* *0103*
OVERLAY-ADDITION                /0078/
OVERLAY-DISPLAY-GREETING        /0106/
OVERLAY-DIVISION                /0099/
OVERLAY-MULTIPLICATION          /0092/
OVERLAY-SUBTRACTION             /0085/
RESIDENT                        /0040/
RESULT                          /0012/  0053  *0055* *0081* *0088* *0095* *0102*
RESULT-EDITED                   /0026/ *0053* *0066*  0067  *0071*  0072
RESULT-MESSAGE                  /0023/  0054
RE-TRY                          /0043/  0051   0061
STOP-RUN                        /0075/
SUBTRACTION                      0047  /0086/
WAIT-CHAR                       /0015/ *0059*  0060
WAIT-ENTRY                      /0057/  0109
WAIT-MESSAGE                    /0029/  0058
TRS-80 Model II COBOL (RM/COBOL 1.2)                4/ 3/80  00.08.38  PAGE     1
SOURCE FILE: ERRXMPL/CBL                           OPTION LIST: (P X

LINE  DEBUG PG/LN  A...B............................................................ID......

   1        000000 IDENTIFICATION DIVISION.
   2        000010 IDENTI
                   $
*****    1) SYNTAX   *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
   3        000020 PROGRAM-ID.
                   $
*****    1) SCAN RESUME   *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
   4        000030     ERROR-EXAMPLES.
   5        000040     ER
                       $
*****    1) SYNTAX   *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
   6        000050 ENVIRONMENT DIVISION.
                   $
*****    1) SCAN RESUME   *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
   7        000060 CONFIGURATION SECTION.
   8        000070 SOURCE-COMPUTER.  RMC-MINI.
   9        000080 OBJECT-COMPUTER.  RMC-MINI.
  10        000090 INPUT-OUTPUT SECTION.
  11        000100 FILE-CONTROL.
  12        000110     SELECT INPUT-FILE
  13        000120         ASSIGN TO INPUT, INPUT-NAME;
  14        000130         FILE STATUS IS INPUT-STATUS.
  15        000140     SELECT OUTPUT-FILE
  16        000150         ASSIGN TO OUTPUT, OUTPUT-NAME;
  17        000160         FILE STATUS IS OUTPUT-STATUS.
  18        000170
  19        000180 DATA DIVISION.
  20        000190 FILE SECTION.
  21        000200 FD  INPUT-FILE
  22        000210     RECORD CONTAINS 80 CHARACTERS
  23        000220     LABEL RECORD IS OMITTED.
  24        000230 01  INPUT-REC.
  25        000240     05  FILLER           PIC  X(06).
  26        000250     05  INPUT-FLD        PIC  X(66).
  27        000260     05  AREA-FLDS        REDEFINES INPUT-FLD.
  28        000270         10  AREA-C       PIC  X(01).
  29        000280         10  AREA-A       PIC  X(04).
  30        000290         10  AREA-B       PIC  X(61).
  31        000300     05  FILLER           PIC  X(08).
  32        000310 FD  OUTPUT-FILE
  33        000320     RECORD CONTAINS 80 CHARACTERS,
  34        000330     LABEL RECORD IS OMITTED.
  35        000340 01  OUTPUT-REC.
  36        000350     05  SEQ-FLD          PIC  9(06).
  37        000360     05  OUTPUT-FLD       PIC  X(66).
  38        000370     05  FILLER           PIC  X(08).
  39        000380 WORKING-STORAGE SECTION.
  40        000390 77  INPUT-NAME           PIC  X(28).
  41        000400 77  OUTPUT-NAME          PIC  X(28).
  42        000410 77  COUNT                PIC  9(06) VALUE 0.
  43        000420 77  LARGE-VALUE          PIC  X(04) VALUE "ERROR".
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)              4/ 3/80  00.08.38  PAGE     2
SOURCE FILE: ERRXMPL/CBL                          OPTION LIST: (P X

LINE  DEBUG PG/LN  A...B...........................................................ID......

  44        000430 77  PIC-ERROR            PIC  *(05).*9.
                                                        $
*****   1) PICTURE  *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
*****   1) SCAN RESUME  *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
  45        000440 77  INPUT-STATUS         PIC  X(04).
  46        000450 77  OUTPUT-STATUS        PIC  X(02).
  47        000460 01  SEQ-VALUE            PIC  9(06).
  48        000470 01  SE
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)              4/ 3/80  00.08.38  PAGE.    3
SOURCE FILE: ERRXMPL/CBL                         OPTION LIST: (P X

LINE   DEBUG PG/LN  A...B............................................................ID......

            000480/
  50        000490 PROCEDURE DIVISION.
                   $                 $
*****   1) SYNTAX    *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
*****   2) SCAN RESUME   *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
  51        000500 0100.
                   $
*****   1) LEVEL   *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
  52        000510     DISPLAY "COBOL PROGRAM SEQUENCER",
  53        000520         LINE 1 POSITION 30 ERASE.
  54        000530     DISPLAY SPACES LINE 2.
  55        000540     DISPLAY "INPUT FILE:  ".
  56        000550     MOVE 3.5 TO OUTPUT-STATUS.
  57        000560     ACCEPT INPUT-NAME POSITION 0 PROMPT ECHO.
  58        000570     DISPLAY "OUTPUT FILE:  ".
  59        000580     ACCEPT OUTPUT-NAME POSITION 0 PROMPT ECHO.
  60        000590     OPEN INPUT INPUT-FILE.
  61        000600     OPEN OUTPUT OUTPUT-FILE.
  62        000610     MOVE SPACES TO OUTPUT-REC.
  63        000620     MOVE 0 TO SEQ-VALUE.
  64        000630     DISPLAY "SEQUENCING BEGUN".
  65        000640 0200.
  66        000650     READ INPUT-FILE AT END
  67        000660         GO TO 0300.
  68        000670     PERFORM INPUT-CHECK.
  69        000680     ADD 10 TO SEQ-VALUE.
  70        000690     MOVE SEQ-VALUE TO SEQ-FLD.
  71        000700     MOVE INPUT-FLD TO OUTPUT-FLD.
  72        000710     WRITE OUTPUT-REC.
  73        000720     ADD 1 TO COUNT.
  74        000730     GO TO 0200.
  75        000740 0300.
  76        000750     DISPLAY COUNT,
  77        000760         " RECORDS SEQUENCED AND COPIED" POSITION 0.
  78        000770     CLOSE INPUT-FILE, OUTPUT-FILE.
  79        000780     STOP RUN.
  80        000790     GO TO 0150.
  81        000800 END PROGRAM.
                   $
*****   1) SCAN RESUME   *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
*****   1) SYNTAX    *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
*****   1) SCAN RESUME   *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
*****   1) SYNTAX    *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
  82        ZZZZZZ END PROGRAM.                        *** END OF FILE ***
```

```
 ADDRESS  SIZE DEBUG ORDER TYPE                     NAME

             0              FILE                    INPUT-FILE
  >0000     80  GRP     0   GROUP                    INPUT-REC
  >0006     66  ANS     0   ALPHANUMERIC              INPUT-FLD
  >0006     66  GRP     0   GROUP                     AREA-FLDS
  >0006      1  ANS     0   ALPHANUMERIC               AREA-C
  >0007      4  ANS     0   ALPHANUMERIC               AREA-A
  >000B     61  ANS     0   ALPHANUMERIC               AREA-B

             0              FILE                    OUTPUT-FILE
  >0050     80  GRP     0   GROUP                    OUTPUT-REC
  >0050      6  NSU     0   NUMERIC UNSIGNED          SEQ-FLD
  >0056     66  ANS     0   ALPHANUMERIC              OUTPUT-FLD

  >00A4     28  ANS     0   ALPHANUMERIC            INPUT-NAME

  >00C0     28  ANS     0   ALPHANUMERIC            OUTPUT-NAME

  >00DC      6  NSU     0   NUMERIC UNSIGNED        COUNT

  >00E2      4  ANS     0   ALPHANUMERIC            LARGE-VALUE

  >00E6    166  ANS     0   ALPHANUMERIC            PIC-ERROR

  >00EE      4  ANS     0   ALPHANUMERIC            INPUT-STATUS

  >00F2      2  ANS     0   ALPHANUMERIC            OUTPUT-STATUS

  >00F4      6  NSU     0   NUMERIC UNSIGNED        SEQ-VALUE
RESERVED WORD CONFLICT *W*W*W*W*W*W*W*W*W*W*W*W*W  COUNT

VALUE ERROR *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E LARGE-VALUE

FILE STATUS ERROR *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E INPUT-FILE


READ ONLY BYTE SIZE =        >0042

READ/WRITE BYTE SIZE =       >0134

OVERLAY SEGMENT BYTE SIZE = >0000

TOTAL BYTE SIZE =            >0176

    9 ERRORS

    7 WARNINGS
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                   4/ 3/80  00.08.38  PAGE     5
SOURCE FILE: ERRXMPL/CBL                              OPTION LIST: (P X

CROSS REFERENCE                /DECL/ *DEST*

AREA-A                         /0029/
AREA-B                         /0030/
AREA-C                         /0028/
AREA-FLDS                      /0027/
COUNT                          /0042/
INPUT-FILE                     /0012/ /0021/
INPUT-FLD                      /0026/ /0027/
INPUT-NAME                     *0013* /0040/
INPUT-REC                      /0024/
INPUT-STATUS                   *0014* /0045/
LARGE-VALUE                    /0043/
OUTPUT-FILE                    /0015/ /0032/
OUTPUT-FLD                     /0037/
OUTPUT-NAME                    *0016* /0041/
OUTPUT-REC                     /0035/
OUTPUT-STATUS                  *0017* /0046/
PIC-ERROR                      /0044/
SEQ-FLD                        /0036/
SEQ-VALUE                      /0047/
```

TRS-80 (TM) MODEL II COBOL
SAMPLE SESSION

This section will take you through a compilation and execution session, starting with a COBOL source file. We will use the sample program, CALCXMPL/CBL, included with your COBOL diskette.

We will also create a runtime diskette, i.e., a diskette that contains only those COBOL files necessary to executing a compiled program.

STEP ONE. Create the source file.

In this session, we'll use one of the sample programs included on the diskette, namely, CALCXMPL/CBL. To create your own source file, follow the instructions in the COBOL Editor User's Guide.

STEP TWO. Compile.

With the COBOL diskette in drive zero, type under TRSDOS READY:

```
  RSCOBOL CALCXMPL {T}
```

The T option causes a listing to be displayed at the console. For other options available, see section 1.3.1 of this manual.

This command creates an object file that can be executed by the COBOL runtime. The file will have the name CALCXMPL/COB.

STEP THREE. Execute.

Under TRSDOS READY, type:

```
    RUNCOBOL CALCXMPL
```

The runtime will execute the program CALCXMPL/COB.

(Continued)

STEP FOUR (OPTIONAL). Create a runtime diskette.

A runtime diskette contains only those COBOL files necessary for executing a pre-compiled COBOL program. In this example, we will assume you have a one-drive system and are going to run the CALCXMPL program.

4.1 First make an extra backup copy of the diskette containing all COBOL files, including your source program.

4.2 Using the TRSDOS PURGE command, eliminate the following COBOL files (do not delete RUNCOBOL):

```
  RSCOBOLnnn/OBJ          (nnnn = version/release numbers)
  RSCBLnnn/OBJ
  RSCBLDnn/OBJ*
  CALCXMPL/CBL
  CALCXMPL/LST
  ERRXMPL/CBL
  ERRXMPL/LST
  ERRXMPL/COB
  COBPRT
  CEDIT
```

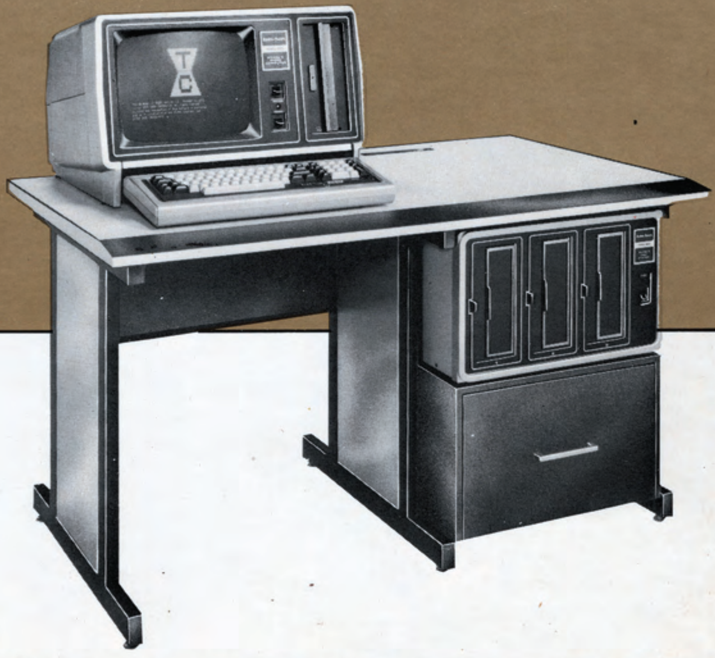
*Do not delete RSCBLDnn/OBJ if you want to use the runtime debug feature.

Note: You can also purge unnecessary system files. See "Creating a minimal system diskette," Section 0 of the TRSDOS Reference Manual.

Radio Shack®

TRS-80™ Model II RSCOBOL CEDIT User's Guide

Using CEDIT to Create and Edit COBOL Source Files.
Using the COBPRT Print Utility.



CEDIT

CUSTOM MANUFACTURED FOR RADIO SHACK A DIVISION OF TANDY CORPORATION

TRS-80 (TM)

TRS-80 (TM) MODEL II COBOL

C E D I T — SOURCE PROGRAM EDITOR

C O B P R T — PRINTOUT UTILITY

USER'S GUIDE

Radio Shack®

TABLE OF CONTENTS

INTRODUCTION

CEDIT lets you create and edit COBOL source files (the files that are input to the COBOL Compiler).

Capabilities and features:
. Allows you to load in ("chain") multiple source files.
. Single-key abbreviations for many commands
. Powerful intra-line editing mode like the edit mode in Model II Interpreter BASIC
. "M" command informs you of memory used/free at any time
. Global string find/change commands
. Each line can contain up to 128 characters of text (COBOL compiler accepts lines of up to 80 characters.
. Editor provides line numbers in the range 0-65535
. Line renumbering command, and special auto-renumber feature
. Execute any TRSDOS library command without exiting the editor
. <TAB> skips over to the next four-column boundary--to match standard COBOL field boundaries

SOURCE FILE FORMAT

Source files are written to disk in the format required by the COBOL compiler, as follows:
1. Files are variable-length record (VLR) type, as described in the TRSDOS Reference Manual, page 4/5.
2. Each record in the file corresponds to one line of source program. The first six data bytes (after the length-byte) in a record represent the line number in ASCII form. The carriage return (<ENTER>) used to terminate the line during line insertion is not stored.
3. Text is stored exactly as it is displayed on the video, e.g., spaces are stored as spaces, not as a tab character.
4. No end-of-text code is stored in the data file.

Note: Although the editor will allow lines of up to 128 characters, the COBOL compiler will not accept more than 80 characters per line.

TO START THE EDITOR

The editor program is included on the COBOL package diskette. It has the file name CEDIT.

To use the editor, put the COBOL diskette into one of your drives (drive 0 for single-drive users), and under TRSDOS READY, type:

```
    CEDIT
```

The editor will start up with the prompt:

```
    MODEL II COBOL EDITOR VERSION v.r
    OK
    >
```

Where v is the version and r is the release number. The > indicates you are in the command mode.

MODES OF OPERATION

There are three modes of operation:
. COMMAND, for entering the editor commands
. INSERT, for entering your text lines
. EDIT, for interactive editing of a line of text

COMMAND MODE
The > prompt followed by the blinking cursor indicates the editor is waiting for you to type in a command. Every command must be completed by pressing <ENTER>. To cancel a command, press <ESC> or <BREAK>.

INSERT MODE
You enter text one line at a time; a line consists of up to 128 characters. The editor numbers each line with a reference number from 000000 to 065535 (the leftmost digit is always zero).

The I command puts you in the insert mode. When you start inserting a line, the editor displays the six-digit line number followed by the blinking cursor. Your text can begin in column seven. (See the COBOL Language Reference Manual for column-field uses in COBOL source programs.)

To store the current line, press <ENTER>. The editor will display the next line number, and you can begin inserting into that line. To cancel the current line and return to the command mode, press <ESC> or <BREAK>. See I Command for details.

EDIT MODE
There are many powerful edit sub-commands--identical in most cases to those in Model II BASIC's Edit Mode. There is also a sub-edit insertion mode in which the keys you type are inserted into the line at the current cursor position.

To start editing a line, use the E command. After editing the line, press <ENTER> to save the corrected line and return to the command mode. To cancel all changes made and return to the command mode, press <Q>. For further details, see E Command.

USING THE COMMAND MODE

Special terms used in the command descriptions:

"text", "text buffer", "text area"
All refer to the COBOL source program currently in RAM.

"current line"
The line most recently inserted, displayed or referenced in a command. When there is no text in RAM, current line is set to 100. Immediately after a file is loaded, the current line is set to the beginning of the text.

"increment"
The value which is added to the current line number whenever the editor needs to compute a new line number. After startup, loading a new file, and when there is no text in RAM, the increment is set to 10.

"line-reference"
Either an actual line number from 0 to 65535, or one of the following special abbreviations:

Symbol	Meaning
#	Beginning line of text (lowest-numbered line)
.	Current line
*	Last line of text (highest-numbered line)

"line-range"
This can be either a single-line reference or a pair of line-references separated by a colon:

Sample Command	Meaning
P100	Prints line 100 only
P100:300	Prints all lines from 100 to 300
P#:.	Prints all lines from beginning to current

"delimiter"
A special character used to delimit (mark the beginning and end of) a string. Any of the following characters can be used:
! " # $ % & ' ( ) * + , - . / : ; < = > ?
These are the ASCII characters in the ranges <X'21', X'2F'> and <X'3A', X'3F'>

Whichever character is used to mark the beginning of a string must also be used to mark the end of the string.

```
    Sample use...            To find this string...
    -------------            ------------------------
    F'THIS " MARK'            THIS " MARK
    F/X'8000'/                X'8000'
    F&~~~~~~~&                ~~~~~~~  (seven blanks)
```

(The "~" symbol represents a blank space. It is used only where necessary for emphasis or illustration.)

The F (find) command is explained later on.

SPECIAL KEYS IN THE COMMAND MODE

<ESC> or <BREAK>
Press either key to cancel the command you are entering, or to abort a command which is currently being executed.

<TAB>
Advances the cursor to the next four-column boundary (boundaries in the command mode are at columns 4, 8, 12, 16, ...

<ENTER>
Pressing this key at the beginning of a command line displays the current line.

<up-arrow>
Pressing this key at the beginning of a command line displays the line which precedes the current line.

<down-arrow>
Pressing this key at the beginning of a command line displays the next line after the current line.

<-
Erases the command you are entering.

<HOLD>
Pauses H and P commands. Press any key to continue.

COMMANDS

Note: Spaces are not significant in command lines. For example,

```
    P 1 : 5
```

has the same effect as

```
    P1:5
```

Print lines found in the range 1 to 5. (See P command for details).

A

Enables automatic line-renumbering. Whenever a line-number collision occurs in the insert mode, the editor will automatically renumber the text lines, using the current line number as start-line and the current increment as increment. See N commmand for details on renumbering.

Note: The A command does not put you in the insert mode; only the I and R commands do that. The A command simply sets an auto-renumbering "switch".

This function is disabled when you execute the N or L command.

B

Displays the beginning line (first line in the text area).

C/search-string/replacement-string/n

Finds, changes, and displays the first n lines that contain search-string. In each of these lines, search-string is changed to replacement-string. ONLY THE FIRST OCCURRENCE OF search-string IN A SINGLE LINE IS COUNTED AND CHANGED. If the end of text is reached before n finds, the message "SEARCH FAILS" will be displayed.

Upon completion of the command, the current line is set to the line of the last find, or to the first line of text when "SEARCH FAILS" is displayed.

/search-string/ is a sequence of characters delimited by
    a matched pair of characters from the set:
    ! " # $ % & ' ( ) * + , - . / : ; < = > ?
     replacement-string/ is a sequence of characters terminated by
    the same character used to delimit search-string.

n   Tells the maximum number of "changes" you want. n can be a number or an asterisk. The asterisk means change and list all occurrences. If n is omitted, only the first occurrence is changed and listed.

Sample Commands	Notes
C/VAR=/NET=/	Changes the first occurrence of "VAR=" to "NET=" in the first line that contains it.
C"VAR="NET="	Same as above.
C/RETRY/R/4	Changes the first occurrence of "RETRY" to "R" in the first four lines that contain it.
C/MISPELING/MIS-SPELLING/*	Changes the first occurrence of "MISPELING" to "MIS-SPELLING" in every line that contains it.
C/EXTRA//*	Changes the first occurrence of "EXTRA" to "" (null string) i.e., deletes the first "EXTRA" in every line that contains it.

D line-range

Deletes lines in the specified range. If line-range is omitted, the current line is deleted.

Sample Commands	Notes
D. or D	Deletes the current line.
D2	Deletes line number 2.
D98:115	Deletes lines found in the range 98 to 115.
D1000:*	Deletes all lines numbered 1000 or higher to end of text.

E line-reference

Starts edit mode using the specified line. If line-reference is omitted, the current line is used.

Edit sub-commands:

<ENTER>	Ends editing and returns to command mode.
<F1>	Causes escape from sub-edit insertion (X, I, and H sub-commands) and returns to edit mode.

n <SPCBAR>	Advances cursor n columns. If n is omitted, 1 is used.
<BKSPC>	Backspaces the cursor one column without erasing.
L	Lists working copy of the line and starts a new working copy.
X	Extends line: positions cursor to end of line and enters sub-edit insertion mode. Use <F1> to escape to edit mode.
I	Enters sub-edit insertion mode at the current cursor position; use <F1> to escape. to edit mode.
A	Cancels changes and starts a new working copy of the line.
E	("End") Saves edited line and exits to command mode, > prompt.
Q	("Quit") Cancels changes and returns to command mode, > prompt.
H	"Hacks" remainder of line beginning at current cursor position and enters sub-edit insertion mode. Use <F1> to escape to edit mode.
nD	Delete n characters beginning at current cursor position. If n is omitted, 1 is used. The deletion is not echoed; use <L> to see the line with characters deleted.
nC	Change next n characters from the current cursor position, using the next n characters typed. If n is omitted, 1 is used.
nSc	("Search") Move cursor to nth occurence of character c. Search starts at next character after the cursor. If n is omitted, 1 is used.
nKc	Delete all characters from current cursor position up to nth occurence of character c, counting from current cursor position. If n is omitted, 1 is used. The deletion is not echoed; use <L> to see the line with characters deleted.

See Sample Session for examples.

F/search-string/n

Finds and displays the first n lines which contain search-string, starting at the current line. ONLY THE FIRST OCCURRENCE OF search-string IN A SINGLE LINE IS COUNTED. If the end of text is reached before n finds, the message "SEARCH FAILS" will be displayed.

Upon completion of the command, the current line is set to the line of the last find, or to the first line of text when "SEARCH FAILS" is displayed.

/search-string/ is a sequence of characters delimited by a matched pair of delimiters chosen from the set:
! " # $ % & ' ( ) * + , - . / : ; < = > ?
These are the ASCII characters in the ranges <X'21', X'2F'> and <X'3A', X'3F'>

n Tells the maximum number of "finds" you want. n can be a number or an asterisk. The asterisk means find and list all occurrences. If n is omitted, only the first occurrence is listed.

Sample Commands	Notes
F/VAR=/	Finds and displays the first line that contains the string "VAR=".
F"VAR="	Same as above.
F/RETRY/4	Finds and displays the first four lines containing at least one occurrence of "RETRY".
F/MISPELING/*	Finds and displays every line containing at least one occurrence of "MISPELING".

H line-range

("Hard-copy") Lists to the printer all lines found in the specified range. If line-range is omitted, only the current line is printed.

The printer should be initialized (with FORMS) before you execute this command.

Sample Commands	Notes
H#:*	Lists all lines to the printer.
H7020	Lists line 7020 to the printer.
H672:800	Lists all lines found in the range 672 to 800.

I start-line, increment

Starts the insert mode.

start-line is a line-reference telling the editor where to begin inserting into the text. If omitted, the current line is used.

,increment is a number telling the editor how to compute successive line numbers. If omitted, the current increment is used.

If start-line is already in use, the editor will start with the next line number (start-line + increment).

Special Keys in the Insert Mode

<TAB>	Advances the cursor to the next four-column boundary (8, 12, 16, 20, etc.).
<-	Erases the line and starts over.
<BKSPC>	Backspaces the cursor and erases the character.
<ENTER>	Marks the end of the current line. The editor will store the current line and start a new one, using increment to generate the next line number.

Line-Collisions
If the next line number is already in use (this is referred to as a "collision"), the editor will display the message:

```
    NO ROOM BETWEEN LINES
```

and return to the command mode. To allow further insertion at this point in your program, either renumber the text or try inserting with a smaller increment.

Note: If the automatic-renumbering function is enabled when a line collision occurs, the editor will renumber the text automatically before displaying the next line number. See A command.

Sample Commands	Notes
I	Start inserting at current line number, using current increment.
I,1	Start inserting at current line number, using 1 as an increment. If current line number is in use, start with current line plus 1.
I45,2	Start inserting at line 45 with an increment of 2. If line 45 is in use, start with line 47.
I100	Start inserting at line 100, using the current increment. If line 100 is in use, start with 100 plus increment.

L filespec

Loads a source file from disk. If there is already text in RAM, the editor will ask whether you want to chain the new text onto the end of the old, or clear out the old first.

filespec is a TRSDOS file specification for a VLR text file. The file may have been created by this COBOL editor or by another means. However, it must be in the COBOL source file format. (See Source File Format.)

If no extension is provided in filespec, the editor assumes an extension of /CBL. To load a file with a blank extension, insert "/~~~" after the file name ("~" represents a blank space).

The L command also disables the automatic renumbering function (see A command).

Immediately after chaining a file, you must use the N command to renumber the text. This will resolve any duplicate line numbers.

Sample Commands	Notes
L DEMO:1	Load DEMO/CBL from drive 1.
L XDATA/~~~	Load XDATA (blank extension).

M

Prints number of characters in the source text (excluding the editor's line numbers) and the amount of memory free for text storage.

Sample Command	Notes
M	A typical response in a 64K system might might look like this: 000440- TEXT 046145- MEMORY Meaning you have 440 bytes of text, and 46145 free bytes of memory available.

N start-line,increment

Renumbers the entire text.

start-line becomes the lowest line number when the text is renumbered. If start-line is omitted, the current line number is used.

increment is used in computing successive line numbers. If omitted, the current increment is used.

After renumbering, the current line is set to the highest line number in the renumbered text.

This command disables the automatic renumbering function (see A command).

Sample Commands	Notes
N	Renumbered text will start with current line; successive lines computed with current increment.
N100	Renumbered text will start with line 100; successive lines computed with the current value of increment.
N100,25	As above; line numbers at increments of 25.
N,100	Renumbered text will start with current line number; line numbers at increments of 100.

P line-range

Prints the specified lines to the display. If line-range is omitted, 20 lines starting at the current line are displayed.

Sample Commands	Notes
P	Prints 20 lines starting at current line.
P233	Prints line 233.
P.	Prints the current line.
P*	Prints the last line.
P140:615	Prints the lines within the specified range. Lines 140 and 615 don't have to be existing line numbers.

Q

Terminates session and returns to TRSDOS. The source text is not written to disk.

R line-reference, increment

Replaces contents of the specified line and continue in insert mode. If line-reference is omitted, the current line is used. If increment is omitted, the current increment is used.

The R command is equivalent to the D (delete) command followed by the I (insert) command. When you enter the command, the editor deletes the specified line and puts you into the insert mode, starting with the line just deleted.
After you press <ENTER>, the editor will contine in the insert mode, prompting you to enter the text of the next line number. To escape from the insert mode, press <ESC> or <BREAK>.

Sample Commands	Notes
R125,3	Prompts you to insert replacement text for line 125. Subsequent line numbers will be generated with an increment of 3.
R*	Prompts you to insert replacement text for the highest numbered line in the text area; subsequent lines will be generated using the current increment.

S lib-command

Allows you to enter any TRSDOS library command, and return to the editor command mode upon completion. For a list of library commands, try the TRSDOS LIB command.

Sample Command	Notes
S FORMS W=80	Executes the FORMS command, setting the paper width to 80 characters. When FORMS has completed execution, the edit command prompt > is displayed.

W filespec

Writes the text in RAM into the specified file.

filespec is a TRSDOS file specification. If file already exists, its previous contents will be lost.

If no extension is provided in filespec, the editor assumes an extension of /CBL. To save a file with a blank extension, insert "/~~~" after the file name ("~" stands for a blank space).

Sample Commands	Notes
W DEMO:1	Save DEMO/CBL from drive 1.
W XDATA/~~~	Save XDATA (blank extension).

X/search-string/replacement-string/n

This command is exactly like the C (Change) command, except that it displays the line to be changed and queries you (Change?) each time it finds search-string. If you answer Y, the line will be changed; any other answer leaves the line unchanged. In either case, the process continues until all first occurrences have been found.

```
    Sample
    Command            Notes
    ---------          -----
    X/MISPELING/MSP/*
                       Changes the first occurrence of
                       "MISPELING" to "MSP"
                       in every line that contains it, but asks
                       you to confirm each change before it
                       is made.
```

SAMPLE SESSION

This section will show a simple editing session, including:

. Inserting text lines
. Inserting lines in-between existing lines
. Inserting lines in-between existing lines with
  auto-renumbering enabled
. Intra-line editing
. Performing a global search/change

Start the editor as explained on page 4. Type as shown under the DIALOG column. The lowercase Notes are for explanation only.

The symbol "~" means a space (press space-bar). The computer's prompts are underlined--do not type these in. Also note that characters inside < > are keys you press that are not echoed on the display.

```
DIALOG                   NOTES
------                   -----

OK                       Start in command mode
--
>I <ENTER>               Begin insert mode.
-
000100<TAB>DEMONSTRATION OF COBOL EDITOR <ENTER>
------
000110<TAB><TAB>TWO TABS START THIS LINE <ENTER>
------
000120<TAB><TAB><TAB>THREE TABS START THIS LINE <ENTER>
------
000130 <ESC>             Back to command mode.
------
OK
--
>P#:* <ENTER>
-
000100 DEMONSTRATION OF COBOL EDITOR
------------------------------------
000110     TWO TABS START THIS LINE
-----------------------------------
000120         THREE TABS START THIS LINE
-----------------------------------------
>N 10,5 <ENTER>          Renumber text
-
ALL LINES RE-NUMBERED
---------------------
```

```
DIALOG                   NOTES
------                   -----

>P #:* <ENTER>           Display renumbered text
-
000010 DEMONSTRATION OF COBOL EDITOR
-------------------------------------
000015     TWO TABS START THIS LINE
------------------------------------
000020         THREE TABS START THIS LINE
-----------------------------------------
>I 5                     Insert at 5 using old increment (5)
-
000005THISHASNOSPACES <ENTER>
------
NO ROOM BETWEEN LINES      Collision at line 10
---------------------
>P <ENTER>
-
000005THISHASNOSPACES
---------------------
000010 DEMONSTRATION OF COBOL EDITOR
-------------------------------------
000015     TWO TABS START THIS LINE
------------------------------------
000020         THREE TABS START THIS LINE
-----------------------------------------
```

```
DIALOG                  Notes
------                  -----

>A <ENTER>              Turn on auto renumbering
-
>I 11,1 <ENTER>         Insert in-between lines
-
000011<TAB>THESE FOUR LINES ARE INSERTIONS <ENTER>
------
000012<TAB>WHEN A LINE COLLISION OCCURS, RENUMBERING <ENTER>
------
000013<TAB><TAB>WILL OCCUR AUTOMATICALLY. YOU WON'T <ENTER>
------
000014<TAB><TAB>GET A MESSAGE UNTIL RETURNING TO THE <ENTER>
------
000015<TAB><TAB>COMMAND MODE <ENTER>
------
000016<ESC>
------
OK
--
ALL LINES RE-NUMBERED
---------------------
>P #:* <ENTER>
-
000010THISHASNOSPACES
---------------------
000011 DEMONSTRATION OF COBOL EDITOR
------------------------------------
000012 THESE FIVE LINES ARE INSERTIONS
--------------------------------------
000013 WHEN A LINE COLLISION OCCURS, RENUMBERING
------------------------------------------------
000014     WILL OCCUR AUTOMATICALLY. YOU WON'T
----------------------------------------------
000015     GET A MESSAGE UNTIL RETURNING TO THE
-----------------------------------------------
000016     COMMAND MODE
-----------------------
000017     TWO TABS START THIS LINE
-----------------------------------
000018         THREE TABS START THIS LINE
-----------------------------------------
```

```
DIALOG                       NOTES
------                       -----
>E 15 <ENTER>                Start editing line 15
-
000015<L> GET A MESSAGE UNTIL RETURNING TO THE
------    ------------------------------------
                             Search for "G", delete 3 and list:
000015<S><G><3><D><L> A MESSAGE UNTIL RETURNING TO THE
------                ------------------------------
                             Search for "A", insert "RECEIVE ":
000015<S><A><I>RECEIVE~<F1><ENTER>
------
>P 15 <ENTER>                Display edited line.
-
000015     RECEIVE A MESSAGE UNTIL RETURNING TO THE
-----------------------------------------------------
                             Set current line = beginning:
>B <ENTER>
-
000010THISHASNOSPACES
---------------------
                             Find and change:
>C/TAB/TAB-CHARACTER/* <ENTER>
-
000017     TWO TAB-CHARACTERS START THIS LINE
---------------------------------------------
000018         THREE TAB-CHARACTERS START THIS LINE
---------------------------------------------------
SEARCH FAILS
------------
>P <ENTER>
-
000010THISHASNOSPACES
---------------------
000011 DEMONSTRATION OF COBOL EDITOR
------------------------------------
000012 THESE FIVE LINES ARE INSERTIONS
--------------------------------------
000013 WHEN A LINE COLLISION OCCURS, RENUMBERING
------------------------------------------------
000014     WILL OCCUR AUTOMATICALLY. YOU WON'T
----------------------------------------------
000015     RECEIVE A MESSAGE UNTIL RETURNING TO THE
---------------------------------------------------
000016     COMMAND MODE
-----------------------
000017     TWO TAB-CHARACTERS START THIS LINE
---------------------------------------------
000018         THREE TAB-CHARACTERS START THIS LINE
---------------------------------------------------
>
```

TRS-80 (TM) MODEL II COBOL

C O B P R T
COBOL SOURCE FILE PRINT UTILITY

USER'S GUIDE

The Model II COBOL Package includes a special file-print utility, COBPRT. This program allows you to output COBOL source and list files to the display and to a printer if one is available. Note: "source files" are created by the COBOL editor; "list files" are created by the COBOL compiler.

The listing will always go to the display; if you have initialized a printer before starting the program, the program will also attempt output to the printer. If you want a printout, put the printer online before starting the program.

To use COBPRT:
Under TRSDOS READY, type:

```
    COBPRT filespec <ENTER>
```

where filespec is a TRSDOS file specification for a COBOL source or list file. If you omit filespec, COBPRINT will prompt you to enter the listing file specification.

The program will read in the file and output a copy to the display and line printer if available.

For example, assume you have a COBOL listing file named CALCXMPL/CBL. Then type:

```
    COBPRT CALCXMPL/CBL <ENTER>
```

Radio Shack®

TRS-80™ Model II RSCOBOL Language Reference Manual

A Description of the RSCOBOL Programming Language

CUSTOM MANUFACTURED FOR RADIO SHACK A DIVISION OF TANDY CORPORATION

RSCOBOL

TRS-80 Model II

COBOL LANGUAGE MANUAL

MARCH, 1980

PREFACE

This reference document describes the COBOL Language as implemented on the Radio Shack TRS-80 Model II Microcomputer under the TRSDOS Disk Operating System.

It assumes the reader is familiar with the COBOL Language, the general operation of the TRS-80 Model II Microcomputer, and the TRSDOS Operating System. The reader is specifically referred to the following publications:

TRS-80 Model II COBOL User's Guide
TRS-80 Model II Operation Manual
TRS-80 Model II Disk Operating System Reference Manual

COPYRIGHT NOTICES

ACKNOWLEDGEMENT

Much of the material in this manual is extracted from the ANSI X3.23-1974 COBOL Standard. Accordingly, the following acknowledgement is made as required in that document.

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

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

The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28- 8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

TABLE OF CONTENTS

I

INTRODUCTION

What is COBOL?

COBOL (COmmon Business Oriented Language) is an English oriented programming language designed primarily for developing business applications on computers. It is described as English oriented because its free form enables a programmer to write in such a way that the final result can be read easily and the general flow of the logic can be understood by persons not necessarily as closely allied with the details of the problem as the programmer himself.

Because COBOL is a programming language it can be translated to serve as communication between the programmer and the computer. The COBOL program (the source program) which has been written by the programmer is input to the COBOL compiler. The COBOL compiler then translates the COBOL program into a machine readable form (the object program).

Although each computer has its own unique COBOL compiler program, an industry-wide COBOL effort has resulted in a degree of compatibility so that a COBOL source program can be exchanged among different computers of one manufacturer or among computers of different manufacturers.

A COBOL program is both a readable document and an efficient computer program. Throughout the study of the COBOL language, it is important to keep these two basic capabilities of COBOL in mind and to observe the close relationship between them.

The readability factor of the COBOL language facilitates communication not only between programmer and management, but also among programmers, with a minimum of additional documentation. The readability factor need not affect the other equally important capability of constituting an efficient computer program. It is precisely here that the attention of a good COBOL programmer is centered. He can produce a solution in the form of a well-integrated COBOL program by combining the following: knowledge of the problem, programming technique, capability of the equipment, and familiarity with the available elements of the COBOL language.

The History of COBOL

Development of the COBOL programming language is a continuing process performed by the Programming Language Committee (PLC) of the COnference on DAta SYstems Languages (CODASYL). This committee is made up of representatives of computer manufacturers and computer users.

The first version of the COBOL programming language to be published by CODASYL was called COBOL-60. The second version, called COBOL-61, contained changes in the organization of the Procedure Division and thus was not completely compatible with COBOL-60.

In 1963 the third version, called COBOL-61 Extended, was released. It was basically COBOL-61 with the addition of the sort feature, the addition of the report writer feature, and the modification of the arithmetics to include multiple receiving fields and the CORRESPONDING option.

The fourth version of the COBOL programming language, COBOL-65, consists of COBOL-61 Extended with the inclusion of a series of options to provide for the reading, writing, and processing of mass storage files and the addition of table handling features.

Beginning in 1968 the CODASYL COBOL Programming Language Committee began to report its developmental work in a Journal of Development. The first report to be published was the CODASYL COBOL Journal of Development -- 1968. This journal is the official report of the CODASYL COBOL Programming Language Committee and it documents the developmental activities of CODASYL through July 1968. COBOL-68 is based on COBOL-65 with certain additions and deletions.

Additional COBOL Journal of Development reports were published in 1969, 1970 and 1973. Each documented the developmental activities of CODASYL from the previous report, resulting in continually varying COBOL definitions.

The Standardization of COBOL

In September 1962 the American National Standards Institute (ANSI) set up a committee to work on the definition of a standard COBOL programming language. This standardization effort was based on the technical content of COBOL as defined by CODASYL. In August 1968 an American National Standard COBOL was approved which was based upon the developmental work of CODASYL through January 1968. This first version was called American National Standard COBOL 1968.

In May 1974 a revision of American National Standard COBOL was approved. This revision, called American National Standard COBOL 1974, is based upon the developmental work of CODASYL through December 1971. The COBOL programming language and compiler described in this document is based on the American National Standard COBOL 1974.

CONVENTIONS USED IN THIS MANUAL

This manual presents the language definition and capabilities of COBOL in a generally accepted syntax consistent with the 1974 American National Standard COBOL document. As a result, COBOL Syntax is specified by formats employing special notation.

Words

All underlined uppercase words are key words and are required when the functions of which they are a part are used. Uppercase words which are not underlined are optional and may or may not be present in the source program. Uppercase words, whether underlined or not, must be spelled correctly.

Lowercase words are generic terms used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied by the user. When generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion.

Brackets and Braces

When a portion of a general format is enclosed in brackets, [], that portion may be included or omitted at the user's choice. Braces, {}, enclosing a portion of a general format means a selection of one of the options contained within the braces must be made. In both cases, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies. If an option within braces contains only reserved words that are not key words, then the option is a default option (implicity selected unless one of the other options is explicitly indicated).

Ellipsis

The ellipsis (...) represents the position at which repetition may occur at the user's option.

Punctuation

The punctuation characters comma and semicolon are shown in some formats. Where shown in the formats, they are optional and may be included or omitted by the user. In the source program these two punctuation characters are interchangeable and either may be used anywhere one of them is shown in the formats. Neither one may appear immediately preceding the first clause of an entry or paragraph.

If desired, a semicolon or comma may be used between statements in the Procedure Division.

Paragraphs within the Identification and Procedure Divisions, and the entries within the Environment and Data Divisions must be terminated by the separator period.

Special Characters

The characters '+', '-', '>','<', '=', when appearing in formats, although not underlined, are required when such formats are used.

System Dependent Information

Selected features in ANSI COBOL are intended for definition by the implementor, to accomodate the capabilities and restrictions of the host system. These system dependent items are summarized in the COBOL Users Guide.

II

THE STRUCTURE OF THE COBOL LANGUAGE

THE LANGUAGE STRUCTURE

The smallest element in the COBOL language is the character. A character is a digit, a letter of the alphabet, or a symbol. A COBOL word is one possible result obtained when one or more COBOL characters are joined in a sequence of contiguous characters. Just as English words are determined by rules of spelling, so COBOL words are formed by following a specific set of rules.

Using the COBOL rules of grammar, the COBOL words and COBOL punctuation characters are combined into statements, sentences, paragraphs, and sections. When writing normal English, a failure to follow the rules of grammar and sentence structure may cause misunderstanding; the same is true when writing COBOL. It must be emphasized that a thorough knowledge of the rules of COBOL structure is a prerequisite to writing a workable COBOL program.

Character Set

The COBOL character set consists of fifty-one characters:

Digits		0 through 9
Letters		A through Z
Punctuation		Blank (or space)
	,	Comma
	;	Semicolon
	.	Period
	"	Quote
	(	Left parenthesis
	)	Right parenthesis
Special	>	Greater than
	<	Less than
	+	Plus
	-	Minus (or hyphen)
	*	Asterisk
	/	Slash (or Stroke)
	=	Equal
	$	Currency

These characters determine the structure of a COBOL program. In some constructs, such as comments, other characters may be used but they have no grammatical meaning.

Characters are combined to form either a separator or a character-string.

The COBOL character set is a proper subset of the ASCII character code set native to the computer. The complete character set may be used only within non numeric literals and comments. The chart below gives the hexadecimal and decimal codes for the complete character set.

Character	Hexadecimal Value	Decimal Value	Character	Hexadecimal Value	Decimal Value
Space	20	32	@	40	64
!	21	33	A	41	65
"	22	34	B	42	66
#	23	35	C	43	67
$	24	36	D	44	68
%	25	37	E	45	69
&	26	38	F	46	70
'	27	39	G	47	71
(	28	40	H	48	72
)	29	41	I	49	73
*	2A	42	J	4A	74
+	2B	43	K	4B	75
,	2C	44	L	4C	76
-	2D	45	M	4D	77
.	2E	46	N	4E	78
/	2F	47	O	4F	79
0	30	48	P	50	80
1	31	49	Q	51	81
2	32	50	R	52	82
3	33	51	S	53	83
4	34	52	T	54	84
5	35	53	U	55	85
6	36	54	V	56	86
7	37	55	W	57	87
8	38	56	X	58	88
9	39	57	Y	59	89
:	3A	58	Z	5A	90
;	3B	59	[	5B	91
<	3C	60	\	5C	92
=	3D	61	]	5D	93
>	3E	62	^	5E	94
?	3F	63	_	5F	95

Separators

A separator is a string of one or more punctuation characters.

Punctuation characters belong to the following set:

	Space
,	Comma
=	Equal sign
(	Left parenthesis
.	Period
"	Quotation mark (double)
)	Right parenthesis
;	Semicolon

Separators are formed according to the following rules:

1. A space is a separator. Anywhere a space is used as a separator, more than one space may be used.

2. Comma, semicolon, and period are separators when immediately followed by a space. These separators may appear only when explicitly permitted.

3. Parentheses are separators which may appear only in balanced pairs of left and right parentheses delimiting subscripts, indices, arithmetic expressions or conditions.

   Left parentheses must be preceded by a separator space or left parenthesis.

   Right parenthesis must be followed by one of the separators: space, period, semicolon, comma or right parenthesis.

4. Quotes are separators which may appear only in balanced pairs delimiting the nonnumeric literals except when the literal is continued.

   An opening quotation mark must be immediately preceded by a space or left parenthesis.

   A closing quotation mark must be immediately followed by one of the separators: space, comma, semicolon, period or right parenthesis.

5. The separator space may optionally immediately precede all separators except:

   As specified by reference format rules.

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

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

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

These rules do not apply to the characters within nonnumeric literals, picture strings, or comments.

Character-Strings

A character-string is a sequence of one or more characters that form a COBOL word, literal, picture string, or comment. A character-string is delimited by separators.

COBOL Words

A COBOL word is a character-string of not more than 30 characters which form either a user word or a reserved word. All words are one or the other.

User Words

User words are composed of the alphabetic characters, the numbers, and the hyphen character. A user word must not begin or end with a hyphen. With the exception of paragraph-name, section-name, level-number and segment-number, all user-defined words must contain at least one alphabetic character. There are twelve types of user words:

program-name	condition-name
file-name	index-name
record-name	alphabet-name
data-name	text-name
paragraph-name	level-number
section-name	segment-number

Program-Name

The program-name identifies the COBOL source and object program. The name must contain at least one alphabetic character. Only the first 6 characters are associated with the object program.

File-Name

File-names are the internal names for files accessed by the source program. They are not necessarily the same as the external names given to the files. File-names must contain at least one alphabetic character and must be unique.

Record-Name

Record-names are used to name data records within a file. They must contain at least one alphabetic character and, if not unique, must be made unique by qualification with the file name.

Data-Name

A group of contiguous characters or a word of binary data treated as a unit of data is called a data item, named by a data-name. A data-name must contain at least one alphabetic character. References to data items must be made unique by qualification or the appending of subscripts (or indices) or both. Complete unique references to data items are called identifiers.

Paragraph-Name

A paragraph-name is a procedure name that identifies the beginning of a set of COBOL procedural sentences. If not unique, a paragraph-name must be made unique by qualification with a section-name.

Section-Name

A section-name is a procedure name that identifies the beginning of a set of paragraphs. Section-names must be unique.

Condition-Name

A condition-name may be defined in the SPECIAL-NAMES paragraph within the Environment Division or in a level-number 88 description within the Data Division.

A SPECIAL-NAMES condition-name is assigned to ON STATUS or OFF STATUS of one of eight system software switches.

A level-number 88 condition-name is assigned to a specific value, set of values, or range of values within a complete set of values that a data item may assume. The data item itself is called a conditional variable.

A condition-name is used only in conditions as an abbreviation for the relation condition which assumes that the associated switch or conditional variable is equal to one of the set of values to which that condition-name is assigned.

Index-Name

An index-name names an index associated with a specific table. It must contain at least one alphabetic character and must be unique.

Alphabet-Name

An alphabet-name is used to specify a character code set. It must contain at least one alphabetic character and must be unique.

Text-Name

A text-name is the name of a COBOL library text file. It must correspond exactly to a valid file access-name as described in the operating system documentation.

Level-Number

A level-number is used to specify the position of a data item within a data hierarchy. A level-number is a one- or two-digit number in the range 01-49, 66, 77 or 88.

Level-numbers 66, 77 and 88 identify special properties of a data description entry.

Segment-Number

A segment-number specifies the segmentation classification of a section. It is a one- to two-digit number in the range 01-99.

Reserved Words

The structure of COBOL governs the use of certain COBOL words called reserved words. Reserved words, recognized by the COBOL compiler, aid the compiler in determining how to generate a program. A programmer cannot devise a reserved word for a COBOL program; he must use the word designated by the format of the language. A reserved word must not appear as a user-defined word within a program. A list of all reserved words recognized by the compiler is shown in Appendix B.

Five kinds of reserved words are recognized by the compiler:

Key words
Optional words
Connectives
Figurative constants
Special-characters

Key Words

Key words are required elements of COBOL formats. Their presence indicates specific compiler action.

Optional Words

Optional words are optional elements of COBOL formats. Their presence has no effect on the object program.

Connectives

The connectives OF and IN are used interchangeably to connect qualifiers to a user word. The words AND and OR are logical connectives and are used in the formation of conditions.

Figurative Constants

Figurative constants identify commonly used constant values. These constant values are generated by the compiler according to the context in which the references occur. Note that figuratives represent values, not literal occurrences. Thus QUOTE cannot be used to delimit a nonnumeric literal, SPACE is not a separator, and so forth. Singular and plural forms of figuratives are equivalent and may be used interchangeably.

ZERO
ZEROS
ZEROES

Represents the value 0 or one or more zero (0) characters, depending on context.

SPACE
SPACES

Represents one or more space ( ) characters.

HIGH-VALUE
HIGH-VALUES

Represents one or more of the highest characters in the collating sequence (hexadecimal FF).

LOW-VALUE
LOW-VALUES

Represents one or more of the lowest characters in the collating sequence (hexadecimal 00).

QUOTE
QUOTES

Represents one or more quote (") characters.

ALL literal

Represents one or more of the characters comprising the literal. The literal must be either a nonnumeric literal or a figurative constant. When a figurative constant is used, the word ALL is redundant.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

1. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character-by-character on the right until the size of the resultant string is equal to the size in characters of the associated data item. This is done prior to and independent of the application of any JUSTIFIED clause that may be associated with the data item.

2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY or STOP statement, the length of the string is one character.

A figurative constant may be used wherever a literal appears in a format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

Each reserved word which is used to reference a figurative constant value is a distinct character-string with the exception of the construction 'ALL literal' which is composed of two distinct character-strings.

Special Characters

The special character words are the arithmetic operators and relation characters:

+	Plus sign (indexing)
-	Minus sign (indexing)
>	Greater than
<	Less than
=	Equal to

Literals

A literal is a character-string whose form determines its value. Literals are either nonnumeric or numeric.

Nonnumeric Literals

A nonnumeric literal is a character-string enclosed in quotes. Any characters in the COBOL character set may be used. Quote characters within the string are represented by two contiguous quotes. The value of the literal is the string itself excluding the delimiting quotes and one of each contiguous pair of imbedded quotes. The value of the literal may contain from 1 to 2047 characters.

Examples:

Literal	Value
"AGE?"	AGE?
"""TWENTY""?"	"TWENTY"?
""""""	illegal (odd number of quotes)

Numeric Literals

A numeric literal represents a numeric value, not a character-string. Numeric literals are composed according to the following rules:

1. The literal must contain from 1 to 18 digits.

2. The literal may contain a single plus or minus sign if it is the first character.

3. The literal may contain a single decimal point if it is not the last character. The decimal point must be represented with a comma if the DECIMAL-POINT IS COMMA phrase is specified in the SPECIAL-NAMES paragraph.

Examples:

```
 1234
+1234
-1.234
 .1234
+.1234
```

Picture String

A picture string consists of certain combinations of characters from the COBOL character set used as symbols. Any punctuation character appearing as part of a picture string is considered to be a symbol, not a punctuation character.

Comment-Entry

A comment-entry is an entry in the Identification Division that may contain any characters from the computer's character set.

System Names

System names identify certain hardware or software system components. System names consist of device-names and switch-names.

Device-Names	Component
PRINT	printer or print file
INPUT	input only device
OUTPUT	output only device
INPUT-OUTPUT	input-output device
RANDOM	disc

Switch-Names	Component
SWITCH-1	
.	
.	software switches
.	
SWITCH-8	

THE PROGRAM STRUCTURE

Source Format

COBOL programs are accepted as a sequence of formatted lines (or records) of 80 characters or less. Each line is divided into five areas:

Columns	Area
1-6	sequence number
7	indicator
8-11	A
12-72	B
73-80	identification

The sequence number and identification areas are used for clerical and documentation purposes. They are ignored by the compiler.

The indicator area is used for denoting line continuation, comments, and debugging.

Areas A and B contain the actual program according to the following rules:

1. Division headers, section headers, paragraph headers, section-names, and paragraph-names must begin in area A.

2. The Data Division level indicator FD and level-numbers 01 and 77 must begin in area A. Other level-numbers may begin in area A or area B, although B is preferable.

3. The key word DECLARATIVES and the key words END DECLARATIVES, precede and follow, respectively, the declaratives portion of the Procedure Division. Each must appear on a line by itself and each must begin in area A and be followed by a period and a space.

4. Any other language element must begin in area B unless it immediately follows, on the same line, an element in area A.

Continuation of Lines

Whenever a sentence, entry, phrase, or clause requires more than one line, it may be continued by starting subsequent line(s) in area B. These subsequent lines are called the continuation line(s). The line being continued is called the continued line. Any word or literal may be broken in such a way that part of it appears on a continuation line, according to the following rules:

1. A hyphen in the indicator area of a line indicates that the first nonblank character in area B of the current line is the successor of the last nonblank character of the preceding line without any intervening space. However, if the continued line contains a nonnumeric literal without closing quotation mark, the first nonblank character in area B on the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. All spaces at the end of the continued line are considered part of the literal. Area A of continuation line must be blank.

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

Blank Lines

A blank line is one that is blank in the indicator, A and B areas. A blank line can appear anywhere in the source program, except immediately preceding a continuation line with a hyphen in the indicator area.

Comment Lines

A comment line is any line with an asterisk (*) in the indicator area of the line. A comment line can appear as any line in a source program after the Identification Division header. Any combination of characters from the computer's character set may be included in area A and area B of that line. The asterisk and the characters in area A and area B will be produced on the listing but serve as documentation only.

Successive comment lines are allowed. Continuation of comment lines is permitted, except that each continuation line must contain an asterisk in the indicator area.

A special form of comment line represented by a back slash (\) in the indicator area of the line causes page ejection prior to printing the comment.

Debugging Lines

A debugging line is any line with a D in the indicator area of the line. Any debugging line that consists solely of spaces from area A to the identifier area is considered to be a blank line.

A program that contains debugging lines must be syntactically correct with or without the debugging lines.

A debugging line will be considered to have all the characteristics of a comment line if the debug option is not specified at compiler invocation.

Successive debugging lines are allowed. Continuation of debugging lines is permitted, except that each continuation line must contain a D in the indicator area, and character strings may not be broken across two lines.

Statements

COBOL statements always begin with a key word called a verb. There are three kinds of statements: directive, conditional, and imperative.

A directive statement specifies action to be taken by the compiler during compilation. The directive statements are:

The COPY and USE 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. The conditional statements are:

An IF statement.

A READ statement with the AT END or INVALID KEY phrase.

A DELETE, REWRITE or START statement with the INVALID KEY phrase.

A WRITE statement with the INVALID KEY phrase.

An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) with the SIZE ERROR phrase.

An imperative statement specifies an unconditional action to be taken by the object program. The imperative statements are:

A READ statement without the AT END or INVALID KEY phrase.

A DELETE, REWRITE or START statement without the INVALID KEY phrase.

A WRITE statement without the INVALID KEY phrase.

An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT) without the ON SIZE ERROR phrase.

An ACCEPT, ALTER, CLOSE, DISPLAY, EXIT, GO, INSPECT, MOVE, OPEN, PERFORM, SET or STOP statement.

Whenever the term imperative-statement appears in the format of a COBOL verb, it refers to one or more consecutive imperative statements. The sequence ends with a period separator or an ELSE associated with an IF verb.

Sentences

A sentence is a sequence of one or more statements terminated by the period separator. There are three kinds of sentences: directive, conditional, and imperative.

A directive sentence may contain only a single directive statement.

A conditional sentence is a conditional statement, optionally preceded by a sequence of imperative statements, terminated by a period followed by a space.

An imperative sentence is one or more imperative statements terminated by a period separator.

Clauses and Entries

An entry is an item of descriptive or declaratory nature composed of consecutive clauses. Each clause specifies an attribute of the entry. Clauses are separated by space, comma, or semicolon separators. The entry is terminated by a period separator.

Paragraphs

A paragraph is a sequence of an arbitrary number, which may be zero, of sentences or entries. In the Identification and Environment Divisions, each paragraph begins with a reserved word called a paragraph header. In the Procedure Division, each paragraph begins with a user-defined paragraph-name.

Sections

A section is a sequence of an arbitrary number, which may be zero, of paragraphs in the Environment and Procedure Divisions and a sequence of an arbitrary number, which may be zero, of entries in the Data Division. In the Environment and Data Divisions, each section begins with reserved words called a section header. In the Procedure Division, each section begins with a user-defined section-name.

Divisions

Each COBOL program consists of four divisions; each is composed of paragraphs or sections. These are the Identification, Environment, Data, and Procedure divisions, in that order. All divisions are required. Each division begins with a group of reserved words called a division header.

THE COPY STATEMENT

The COPY statement provides the facility for copying text from user-specified files into the source program. Text is copied from the file without change. The effect of the interpretation of the COPY statement is to insert text into the source program, where it will be treated by the compiler as part of the source program.

COBOL library text is placed on the COBOL library as a function independent of the COBOL program and according to operating system techniques.

FORMAT

COPY text-name.

The COPY statement must be preceded by a space and terminated by the separator period. There must not be any additional text in area B following the separator period.

Text-name is the external identification of the file containing the text to be copied. Its format conforms to the rules for filename (or pathname) construction of the host operating system. If the external identification contains any characters that are not letters or digits, or if the first character is not a letter, then the text-name must be written as a nonnumeric literal and enclosed in quotation marks.

A COPY statement may occur in the source program anywhere a characterstring or separator may occur except that a COPY statement must not occur within a COPY statement.

The compilation of a source program containing COPY statements is logically equivalent to processing all COPY statements prior to the processing of 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 reserved word COPY and ending with the punctuation character period, inclusive.

The library text is copied unchanged.

Debugging lines are permitted within library text. If a COPY statement is specified on a debugging line, then the COPY statement will be processed only if the debug option has been specified in the compiler invocation options.

The text produced as a result of processing a COPY statement may not contain a COPY statement.

The syntactic correctness of the library text cannot be independently determined. The syntactic correctness of the entire COBOL source cannot be determined until all COPY statements have been completely processed.

Library text must conform to the rules for COBOL source format.

COPY Examples:

```
FILE-CONTROL.
     COPY  FLCTRL.

PROCEDURE DIVISION.
     COPY  "INPUTP.COBOL".
```

III

IDENTIFICATION DIVISION

INTRODUCTION

The Identification Division must be included in every COBOL source program. This division identifies both the source program and the resultant object program. In addition, the user may include other commentary information.

FORMAT

```
IDENTIFICATION DIVISION.
-------------- --------

PROGRAM-ID. program-name.
----------

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

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

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

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

PROGRAM IDENTIFICATION

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

Paragraph headers identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional and may be included at the user's choice, in the order of presentation shown.

The PROGRAM-ID Paragraph

The PROGRAM-ID paragraph, containing the program-name, identifies the source program, the object program, and all listings pertaining to a particular program. A program-name is a user-defined word made up of only those characters from the word set.

A program-name cannot exceed 8 characters in length, and must contain at least one alphabetic character located in any position within the program-name. Each program-name must be unique.

The AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY Paragraphs

The AUTHOR, INSTALLATION, DATE-WRITTEN, and SECURITY paragraphs are optional. The programmer may use these paragraphs to document information pertaining to the paragraph header.

The comment-entry may be any combination of characters from the computer's character set. The continuation of the comment-entry by the use of the hyphen in the indicator area is not permitted; however, the comment-entry may be contained on one or more lines.

IV

ENVIRONMENT DIVISION

INTRODUCTION

The Environment Division describes the hardware configuration of the compiling computer (source computer) and the computer on which the object program is run (object computer). It also describes the relationship between the files and the input/output media.

The Environment Division must be included in every COBOL source program.

There are two sections in the Environment Division: the Configuration Section and the Input-Output Section.

FORMAT

```
ENVIRONMENT DIVISION.
--------------------

CONFIGURATION SECTION.
---------------------

SOURCE-COMPUTER.  computer-name.
---------------

OBJECT-COMPUTER.  computer-name.
---------------

[SPECIAL-NAMES.  special-names-entry].
 -------------

[INPUT-OUTPUT SECTION.
 ------------ -------

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

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

CONFIGURATION SECTION

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into three paragraphs:

> the SOURCE-COMPUTER paragraph, which describes the computer configuration on which the source program is compiled
>
> the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run
>
> the SPECIAL-NAMES paragraph, which relates names used by the compiler to user-names in the source program.

The SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled.

FORMAT

<u>SOURCE-COMPUTER</u>. computer-name.

Computer-name is a user-defined word and is only commentary.

The OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed.

FORMAT

```
OBJECT-COMPUTER. computer-name
---------------

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

  [, PROGRAM COLLATING SEQUENCE IS alphabet-name].
                       --------
```

Computer-name is a user-defined word and is only commentary.

The MEMORY SIZE definition is treated as commentary.

The PROGRAM COLLATING SEQUENCE clause specifies the program collating sequence to be used in determining the truth value of any nonnumeric comparisons. The Program Collating Sequence clause is treated as commentary; the collating sequence is always ASCII.

The SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates names used by the compiler to user-names in the source program.

```
[SPECIAL-NAMES. [,switch-name
 -------------

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

  [,alphabet-name IS {STANDARD-1}]...
                      ----------
                     {NATIVE    }
                      ------

  [,CURRENCY SIGN IS literal-1]
    --------      --

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

Switch-name may be SWITCH-1, ..., SWITCH-8.

At least one condition-name must be associated with each switch-name given. The status of the switch is specified by condition-names and interrogated by testing the condition-names.

The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. The alphabet-name definition is treated as commentary; the collating sequence is always ASCII.

The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one 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 '*', '+', '-', ',', '.', ';', '(', ')', '"', '/', '='.

If this clause is not present, only the currency sign ($) is used in the PICTURE clause.

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

INPUT-OUTPUT SECTION

The INPUT-OUTPUT section names the files and external media required by an object program and provides information required for transmission and handling of data during execution of the object program. This section is divided into two paragraphs:

> the FILE-CONTROL paragraph which names and associates the files with external media.
>
> the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

FORMAT

```
[INPUT-OUTPUT SECTION.
 ------------ -------

FILE-CONTROL.
-------------

  {file-control-entry} ...

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

The FILE-CONTROL Paragraph

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

FORMAT

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

The content of the file-control-entry is dependent upon the organization of the file named.

The Sequential File Control Entry

FORMAT

```
SELECT file-name
------

   ASSIGN TO device-type, {"external-file-name"}
   ------                 {data-name-1         }

   [;ORGANIZATION IS SEQUENTIAL]
     ------------    ----------

   [;ACCESS MODE IS SEQUENTIAL]
     ------         ----------

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

The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.

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

Device-type must be one of the device names INPUT, INPUT-OUTPUT, OUTPUT, PRINT, or RANDOM according to the operations to be performed.

External-file-name specifies the file access name. It can be from one to thirty characters in length and must be enclosed in quotation marks. A name longer than thirty characters will be diagnosed as an error. The name may contain any sequence of characters supported by the operating system for file access names.

Data-name-1 must be defined in the Data Division as a data item of category alphanumeric and must not be defined in the Linkage Section. Its value at the time of an OPEN statement execution will be used as the file access name. Data-name-1 may be qualified.

The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

Records in the file are accessed in the sequence dictated by the file organization. This sequence is specified by predecessor-successor record relationships established by the execution of WRITE statements when the file is created or extended.

When the ORGANIZATION clause is not specified, ORGANIZATION IS SEQUENTIAL is implied.

The ACCESS MODE clause specifies the order in which records are read or written.

If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is implied.

When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-2 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement.

Data-name-2 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section. Data-name-2 may be qualified.

The Relative File Control Entry

FORMAT

```
SELECT file-name
------

ASSIGN TO RANDOM, {"external-file-name"}
------    ------  {data-name-1         }

  ;ORGANIZATION IS RELATIVE
   ------------    --------

  [;ACCESS MODE IS { SEQUENTIAL [,RELATIVE KEY IS data-name-2]}]
    ------           ----------   --------
                   {{RANDOM }    ,RELATIVE KEY IS data-name-2 }
                     ------       --------
                   {{DYNAMIC}                                 }
                     -------

  [;FILE STATUS IS data-name-3].
         ------
```

The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

Each file described in the Data Divison must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the Data Division.

The ASSIGN TO RANDOM clause specifies the association of the file referenced by file-name to a storage medium.

External-file-name specifies the file access name and must be enclosed in quotation marks. It can be from one to thirty characters in length. A name longer than thirty characters will be diagnosed as an error. The name may contain any characters supported by the operating system for file access names.

Data-name-1 must be defined in the Data Division as a data item of category alphanumeric and must not be defined in the Linkage Section. Its value at the time of an OPEN statement execution will be used as the file access name. Data-name-1 may be qualified.

The ORGANIZATION IS RELATIVE clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

All records stored in a relative file are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of one (1), and subsequent logical records have relative record numbers of 2, 3, 4, . . . n.

The ACCESS MODE clause specifies the order in which records are to be accessed.

When the ACCESS MODE IS SEQUENTIAL, records in the file are accessed in the sequence dictated by the file organization. This sequence is the order of ascending relative record numbers of existing records in the file.

If the ACCESS MODE IS RANDOM, the value of the RELATIVE KEY data item indicates the record to be accessed.

If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be specified for that file.

When the ACCESS MODE IS DYNAMIC, records in the file may be accessed sequentially and/or randomly.

Data-name-2 must not be defined in a record description entry associated with that file-name. The data item referenced by data-name-2 must be defined as an unsigned integer. Data-name-2 may be qualified.

If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is implied.

When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-3 after the execution of every statement that references that file either explicitly or implicitly. This value indicates that status of execution of the statement.

Data-name-3 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section.

The Indexed File Control Entry

FORMAT

```
SELECT file-name
------

ASSIGN TO RANDOM, {"external-file-name"}
------    ------  {data-name-1         }

   [;ORGANIZATION IS INDEXED
     ------------    -------

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

   ;RECORD KEY IS data-name-2
    ------

     [; ALTERNATE RECORD KEY IS data-name-3 [WITH DUPLICATES]]...
        --------- ------                         ----------

   [;FILE STATUS IS data-name-4].
         ------
```

The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

Each file described in the Data Division must be named once and only once as file-name in the FILE-CONTROL paragraph.

Each file specified in the file control entry must have a file description entry in the Data Division.

The ASSIGN TO RANDOM clause specifies the association of the file referenced by file-name to a storage medium.

External-file-name specifies the file access name and must be enclosed in quotation marks. It can be from one to thirty characters in length. A name longer than thirty characters will be diagnosed as an error. The name may contain any characters supported by the operating system for file access names.

Data-name-1 must be defined in the Data Division as a data item of category alphanumeric and must not be defined in the Linkage Section. Its value at the time of an OPEN statement execution will be used as the file access name. Data-name-1 may be qualified.

The ORGANIZATION IS INDEXED clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

The ACCESS MODE clause specifies the order in which records are to be accessed.

When the ACCESS MODE IS SEQUENTIAL, records in the file are accessed in the sequence dictated by the file organization. For indexed files this sequence is the order of ascending record key values within a given key of reference.

If the ACCESS MODE IS RANDOM, the value of the RECORD KEY data item indicates the record to be accessed.

When the ACCESS MODE IS DYNAMIC, records in the file may be accessed sequentially and/or randomly.

If the ACCESS MODE clause is not specified, ACCESS MODE IS SEQUENTIAL is implied.

The RECORD KEY clause specifies the record key that is the prime record key for the file. This prime record key provides an access path to records in an indexed file.

An ALTERNATE RECORD KEY clause specifies a record key that is an alternate record key for the file. This alternate record key provides an alternate access path to records in an indexed file.

The data description of data-name-2 and data-name-3 as well as their relative locations within a record must be the same as that used when the file was created. The number of alternate keys for the file must also be the same as that used when the file was created.

The data items referenced by data-name-2 and data-name-3 must each be defined as a data item of the category alphanumeric within a record description entry associated with that file-name.

Neither data-name-2 nor data-name-3 can describe an item whose size is variable.

Data-name-3 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-2 or by any other data-name-3 associated with this file.

Data-name-2 and data-name-3 may be qualified.

The WITH DUPLICATES phrase specifies that the value of the associated alternate record key may be duplicated within any of the records in the file. If the WITH DUPLICATES phrase is not specified, the value of the associated alternate record key must not be duplicated among any of the records in the file.

When the FILE STATUS clause is specified, a value will be moved by the operating system into the data item specified by data-name-4 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement.

Data-name-4 must be defined in the Data Division as a two-character data item of the category alphanumeric and must not be defined in the File Section.

The I-O CONTROL Paragraph

The I-O CONTROL paragraph specifies the memory area which is to be shared by different files.

FORMAT

```
I-O-CONTROL.
-----------

  [; SAME AREA FOR file-name-1 [, file-name-2] ...] ...
     ----
```

The I-O-CONTROL paragraph is optional.

The SAME AREA clause specifies that two or more files are to use the same memory area during processing. The area being shared includes all storage area assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

More than one SAME clause may be included in a program; however, a file-name must not appear in more than one SAME AREA clause.

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

V

DATA DIVISION

INTRODUCTION

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed falls into three categories:

That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

Constants which are defined by the user.

The Data Division, which is one of the required divisions in a program, is subdivided into three sections:

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry.

The WORKING-STORAGE SECTION describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are assigned in the source program and do not change during the execution of the object program.

The LINKAGE SECTION in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. No space is allocated in the program for data items referenced by data-names in the Linkage Section of that 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. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

FORMAT

```
DATA DIVISION.
---- --------

[FILE SECTION.
 ---- -------

  [file-description-entry]
  [record-description-entry] ... ] ..

[WORKING-STORAGE SECTION.
 --------------- -------

  [77-level-description-entry]
  [record-description-entry  ] ... ]

[LINKAGE SECTION.
 ------- -------

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

FILE SECTION

The File Section header is followed by a file description entry consisting of a level indicator (FD), a file-name and a series of independent clauses. The FD clauses specify the size of the logical and physical records, the presence or absence of label records, the value of label items, and the names of the data records which comprise the file. The entry itself is terminated by a period.

In a COBOL program the file description entry (FD) represents the highest level or organization in the File Section.

FORMAT

```
FILE SECTION.
---- -------

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

The File Description Entry

The File Description furnishes information concerning the physical structure, identification, and record name pertaining to a given file.

FORMAT

```
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 LABEL IS [literal-1]]
      ----- -- -----

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

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

The clauses which follow the name of the file are optional in many cases, and their order of appearance is not significant.

One or more record description entries must follow the file description entry.

A file description entry must end with a period separator.

The BLOCK CONTAINS Clause

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

FORMAT

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

This clause is required except when:

A physical record contains only one complete logical record.

The device assigned to the file has only one physical record size.

The device assigned to the file has a standard record size, although the device may have more than one physical record size. In this case, the absence of this clause denotes the standard physical record size.

The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exist, in which case the RECORDS phrase must not be used:

In mass storage files where logical records may extend across physical records.

The physical record contains padding.

Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are shown, they refer to the minimum and maximum size of the physical record, respectively.

The RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of the data records.

FORMAT

```
   RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
   ------                     --
```

The size of each data record is completely defined with the record description entry, therefore this clause is never required. When present, however, the following notes apply:

> Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record.
>
> If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.
>
> The size is specified in terms of the number of character positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record. The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus any filler characters generated between elementary items because of the SYNCHRONIZED clause.

The LABEL RECORD Clause

The LABEL RECORD clause specifies whether labels are present.

FORMAT

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

This clause is required in every file description entry.

STANDARD specifies that labels exist for the file or the device to which the file is assigned and the labels conform to the operating system specification. STANDARD must be specified for files assigned to a RANDOM device.

OMITTED specifies that no explicit labels exist for the file or the device to which the file is assigned.

The VALUE OF Clause

The VALUE OF clause particularizes the description of an item in the label records associated with a file.

FORMAT

```
    VALUE OF LABEL IS literal-1
    ----- -- -----
```

This clause is treated as commentary.

This clause must not be specified if OMITTED is specified in the LABEL RECORDS clause.

The DATA RECORDS Clause

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

FORMAT

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

Data-name-1 and data-name-2 are the names of data records and must have 01 level-number record descriptions, with the same name, associated with them.

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

Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

WORKING-STORAGE SECTION

The Working-Storage Section is composed of the section header, followed by data description entries for 77 level description entries and/or record description entries.

The data-name of a 01-level data description entry in the Working-Storage Section must be unique since it cannot be qualified. Subordinate data-names need not be unique if they can be made unique by qualification.

FORMAT

```
WORKING-STORAGE SECTION.
--------------- -------

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

LINKAGE SECTION

The structure of the Linkage Section is the same as for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record-name and noncontiguous item name must be unique within the called program since it cannot be qualified.

FORMAT

```
LINKAGE SECTION.
------- -------

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

RECORD DESCRIPTION ENTRY

A record description entry consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name and a series of independent clauses, as required.

FORMAT

```
{data-description-entry} ...
```

Level-Numbers

The first data description of a record must have a level-number of 01 or 1, and must start in area A of a source line.

Each data description entry can be subdivided into multiple data description entries, each having the same level-number; which must be greater than the level-number of the subdivided entry, but less than 50. Level-numbers do not necessarily have to be successive. Thus, a record is a hierarchy of data description entries.

Elementary Items

Any data description entry which is not further subdivided is called an elementary item. A record itself may be an elementary item, consisting of a single level 01 data description entry. A subdivided data description entry with its subdivisions is called a group and is non-elementary. Therefore, a group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered.

Note that certain clauses of the data description entry may occur only in elementary items. They may not occur in 01-level entry as they may affect the subdivisions of that entry. An elementary item must have either a PICTURE clause or INDEX usage; it may not have both.

77 LEVEL DESCRIPTION ENTRY

In the Working-Storage and Linkage Sections, a special level-number of 77 can be used in data description entries which are not subdivisions of other items, and are not themselves subdivided. These data description entries specify noncontiguous data items. Such a data description entry is elementary.

A 77 level description entry must contain a data name and either the PICTURE clause or the USAGE IS INDEX clause, but cannot contain an OCCURS clause. Other clauses are optional and can be used to complete the description of the item if necessary.

FORMAT

data-description-entry

THE DATA DESCRIPTION ENTRY

A data description entry specifies the characteristics of a particular item of data.

FORMAT 1

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

    [;REDEFINES data-name-2]
      ---------

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

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

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

    [;{OCCURS {integer-1 TIMES                                     }
       ------ {integer-1 TO integer-2 TIMES DEPENDING ON data-name-3}
                         --             ---------
       [INDEXED BY index-name-1 [,index-name-2] ...  ]]
        -------

    [;{SYNCHRONIZED} [LEFT ]
       ------------   ----
      {SYNC        } [RIGHT]]
       ----
    [;{JUSTIFIED} RIGHT]
       ---------
      {JUST     }
       ----

    [;BLANK WHEN ZERO]
      -----      ----

    [;VALUE IS literal]
      -----
```

FORMAT 2

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

FORMAT 3

```
88 condition-name; {VALUE IS   } literal-1 [{THROUGH} literal-2]
                    -----                    -------
                   {VALUES ARE}             {THRU    }
                    ------                   ----
      [,literal-3 [{THROUGH} literal-4]] . . . .
                    -------
                   {THRU    }
                    ----
```

The clauses may be written in any order with two exceptions:

the data-name-1 or FILLER clause must immediately follow the level-number;

the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

The words THRU and THROUGH are equivalent.

The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO, must not be specified except for an elementary data item.

Format 3 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Format 3 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any data description entry which contains a level-number except the following:

Another condition-name.

A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

An index data item.

A level 66 item.

Each data description entry must end with a period separator.

The Level-Number

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items, linkage items, condition-names and the RENAMES clause.

FORMAT

```
level-number
```

A level-number is required as the first element in each data description entry.

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

Data description entries in the Working-Storage Section and Linkage Section must have level-numbers with the values 01 through 49, 66, 77 or 88.

The level-number 01 identifies the first entry in each record description.

Level-number 66 is assigned to identify RENAMES entries.

Level-number 77 is assigned to identify noncontiguous working storage data items and noncontiguous linkage data items.

Level-number 88 is assigned to identify condition-names associated with a conditional variable.

Multiple level 01 entries subordinate to any given level indicator FD, represent implicit redefinitions of the same area.

The Data-Name or FILLER Clause

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of the logical record that cannot be referred to explicity.

FORMAT

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

A data-name or the key word FILLER must be the first word following the level-number in each data description entry.

The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly. However, the key word FILLER may be used as a conditional variable because such use does not require explicit reference to the FILLER item, but to its value.

The key word FILLER may not be used in data description entries with a 1, 01, 77, or 88 level-number.

The REDEFINES Clause

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

FORMAT

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

NOTE: Level-number, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

The REDEFINES clause, when specified, must immediately follow data-name-1.

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

This clause must not be used in level 01 entries in the File Section.

The data description entry for data-name-2 cannot contain a REDEFINES clause. Data-name-2 may be subordinate to an entry which contains a REDEFINES clause. The data description entry for data-name-2 cannot contain an OCCURS clause. However, data-name-2 may be subordinate to an item whose data description entry contains an OCCURS clause. In this case, the reference to data-name-2 in the REDEFINES clause may not be subscripted or indexed. Neither the original definition nor the redefinition can include an item whose size is variable as defined in the OCCURS clause.

No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 may occur between the data description entries of data-name-2 and data-name-1.

Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

When the level-number of data-name-1 is other than 01, it must specify the same number of character positions that the data item referenced by data-name-2 contains. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

Multiple redefinitions of the same character positions are permitted. The entries giving the new descriptions of the character positions must follow the entries defining the area being redefined without intervening entries that define new character positions. Multiple redefinitions of the same character positions must all use the data-name of the entry that originally defined the area.

The entries giving the new description of the character positions must not contain any VALUE clauses except in condition-name entries.

Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area.

The PICTURE Clause

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

FORMAT

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

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

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

The maximum number of characters allowed in the character-string is 30.

The PICTURE clause must be specified for every elementary item except an index data item, in which case use of this clause is prohibited.

PIC is an abbreviation for PICTURE.

There are five categories of data that can be described with a PICTURE clause:

alphabetic
numeric
alphanumeric
alphanumeric edited
numeric edited

To define an item as alphabetic:

Its PICTURE character-string can only contain the symbols 'A', and/or 'B'.

Its contents when represented in standard data format must be any combination of the twenty-six (26) letters of the Roman alphabet and the space from the COBOL character set.

To define an item as numeric:

Its PICTURE character-string can only contain the symbols '9', 'P', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive; and

If unsigned, its contents when represented in standard data format must be a combination of the Arabic numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

To define an item as alphanumeric:

Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item; and

Its contents, when represented in standard data format, are allowable characters in the computer's character set.

To define an item as alphanumeric edited:

Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and '/' (stroke);

The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X' or at least one '/' (stroke) and at least one 'X'; or

The character-string must contain at least one '0' (zero) and at least one 'A' or at least one '/' (stroke) and at least one 'A'; and

The contents when represented in standard data format are allowable characters in the computer's character set.

To define an item as numeric edited:

Its PICTURE character-string is restricted to certain combinations of the following symbols: 'B', '/' (stroke), 'P', 'V', 'Z', '0', '9', ',', '.', '*', '-', '+', 'CR', 'DB', and the currency symbol. The allowable combinations are determined from the order of precedence of symbols and the editing rules; and

The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive; and

The character-string must contain at least one '0', 'B', '/' (stroke), 'Z', '*', '+', ',', '.', '-', 'CR', 'DB', or currency symbol.

The contents of the character positions of these symbols that are allowed to represent a digit in standard data format, must be one of the numerals.

The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An integer which is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '/' (stroke), '0', '+', '-', or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

The functions of the symbols used to describe an elementary item are explained as follows:

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

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

Each '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 numeric items. The scaling position character 'P' can appear only to the left or right as a continuous string of 'P's within a PICTURE description; since the scaling position character 'P' implies an assumed decimal point (to the left of 'P's if 'P's are leftmost PICTURE characters and to the right if 'P's are rightmost PICTURE characters), the assumed decimal point symbol 'V' is redundant as either the leftmost or rightmost character within such a PICTURE description. The character 'P' and the insertion character '.' (period) cannot both occur in the same PICTURE character-string. If, in any operation involving conversion of data from one form of internal representation to another, the data item being converted is described with the PICTURE character 'P', each digit position described by a 'P' is considered to contain the value zero, and the size of the data item is considered to include the digit positions so described.

The letter 'S' is used in a character-string to indicate the presence, but neither the representation nor, necessarily, the position of an operational sign; it must be written as the leftmost character in the PICTURE. The 'S' is counted in determining the size (in terms of standard data format characters) of elementary items having DISPLAY or COMPUTATIONAL usage.

The 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear 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.

Each 'X' in the character-string is used to represent a character position which contains any allowable character from the computer's character set.

Each 'Z' in a character-string may only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the contents of that character position is zero. Each 'Z' is counted in the size of the item.

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

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

Each '/' (stroke) in the character-string represents a character position into which the stroke character will be inserted. The '/' (stroke) is counted in the size of the item.

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

When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The character '.' is counted in the size of the item. 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. The insertion character '.' must not be the last character in the PICTURE character-string.

+, -, CR, DB. These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

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

The asterisk when used as the zero suppression symbol and the clause BLANK WHEN ZERO may not appear in the same entry.

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 by either the currency sign or by the single character specified in the CURRENCY SIGN IS clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available:

Simple insertion
Special insertion
Fixed insertion
Floating insertion

There are two types of suppression and replacement editing:

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

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following table specifies which type of editing may be performed upon a given category:

CATEGORY	TYPE OF EDITING
Alphabetic	Simple insertion 'B' only
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B', and '/' (stroke)
Numeric Edited	All, subject to rules below

Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

Simple Insertion Editing

The ',' (comma), 'B' (space), '0', (zero), and '/' (stroke) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Special Insertion Editing

The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

Fixed Insertion Editing

The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two character positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item.

The symbol '+' or '-', when used, must be either the leftmost or rightmost character position to be counted in the size of the item.

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

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

Editing sign control symbols produce the following results depending upon the value of the data item:

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE OR ZERO	DATA ITEM NEGATIVE
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Floating Insertion Editing

The currency symbol and editing sign control symbols, '+' or '-', are the floating insertion characters and as such are mutually exclusive in a given PICTURE character-string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right of this string. These simple insertion characters are part of the floating string.

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data items.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item Nonzero numeric data may replace all the characters at or to the right of this limit.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of non-floating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

Zero Suppression Editing

The suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used the replacement character will be the space and if the asterisk is used, the replacement character will be '*'.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other way is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire data item will be spaces. If the value is zero and the suppression symbol is '*', the data item will be all '*' except for the actual decimal point.

The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

The picture precedence chart shows the order of precedence when using characters as symbols in a character-string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol 'cs'.

At least one of the symbols 'A', 'X', 'Z', '9', or '*', or at least two of the symbols '+', '-', or 'cs' must be present in a PICTURE string.

Nonfloating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-', and 'cs', and other symbol 'P' appear twice in the PICTURE character precedence chart. 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 chart represents its use to the right of the decimal point position.

\1st Sym-bol	2nd\Sym-bol	Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	'	.	{+} {-}	{+} {-}	{CR} {DB}	CS	{Z} {*}	{Z} {*}	{+} {-}	{+} {-}	CS	CS	9	A X	S	V	P	P
NON-FLOATING INSERTION	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	X
	CR DB	X	X	X	X	X				X	X	X			X	X	X			X	X	X
	CS						X															
FLOATING INSERTION	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
	CS	X	X	X	X		X								X							
	CS	X	X	X	X	X	X								X	X				X		X
OTHER	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

PICTURE Character Precedence Chart

The USAGE Clause

The USAGE clause specifies the format of a data item in the computer storage.

FORMAT

```
    [USAGE IS] {COMPUTATIONAL   }
     -----       -------------
               {COMP            }
                 ----
               {COMPUTATIONAL-1}
                 ---------------
               {COMP-1          }
                 ------
               {COMPUTATIONAL-3}
                 ---------------
               {COMP-3          }
                 ------
               {DISPLAY         }
                 -------
               {INDEX           }
                 -----
```

This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operands referenced.

The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

A COMPUTATIONAL (COMPUTATIONAL-1, COMPUTATIONAL-3) item represents a value to be used in computations and must be numeric. If a group is described as COMPUTATIONAL, then the elementary items in the group are COMPUTATIONAL. The group itself is not COMPUTATIONAL (cannot be used in computations.)

The format of a COMPUTATIONAL item is one decimal digit per character position (hexadecimal 00-09). If an 'S' appears in the PICTURE character-string, a trailing byte contains the sign with > 2B being generated for positive and > 2D being generated for negative. COMPUTATIONAL items will be treated as negative if the sign character is > 2D; otherwise they will be considered positive.

The format of a COMPUTATIONAL-1 item (abbreviated COMP-1) is 16 bit two's complement signed binary, independent of the number of nines or appearance of 'S' in the PICTURE character-string. The number of nines is significant when the value is converted to decimal during data manipulation. The value of a COMPUTATIONAL-1 item ranges between -32768 and 32767.

The format of a COMPUTATIONAL-3 item is two decimal digits per character position.

The PICTURE character-string of a COMPUTATIONAL, COMPUTATIONAL-1 or COMPUTATIONAL-3 item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', one or more 'P's. Since a COMPUTATIONAL-1 item must have zero scale it cannot contain any 'P's in its PICTURE character string and if it has a 'V' in its PICTURE character-string the 'V' must be the rightmost character.

The USAGE IS DISPLAY clause indicates that the format of the data is ASCII.

An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table element. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data but the group item name cannot be used in the SET statement or in a relation condition.

An index data item can be referenced explicitly only in a SET statement or a relation condition.

The initial value of an index item is undefined.

The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

The external and internal format of an index data item is the same as a COMPUTATONAL-1 item.

The SIGN Clause

The SIGN clause specifies the position and the mode of representation of the operational sign when it is necessary to describe these properties explicitly.

FORMAT

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

The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 'S'.

The operational sign will be presumed to be the trailing character position of the elementary numeric data item; this character position is not a digit position.

The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.

The numeric data description entries to which the SIGN clause applies must be described as usage is DISPLAY.

At most one SIGN clause may apply to any given numeric data description entry.

The OCCURS Clause

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indices.

FORMAT 1

```
   OCCURS integer-1 TIMES
   ------
        [INDEXED BY index-name-1 [,index-name-2] ...]
         -------
```

FORMAT 2

```
   OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1
   ------           --                 ---------
        [INDEXED BY index-name-1 [,index-name-2] ...]
         -------
```

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement. Further, if the subject of this entry is the name of a group item, then all data-names belonging to the group must be subscripted or indexed whenever they are used as operands, except as the object of a REDEFINES clause.

The OCCURS clause cannot be specified in a data description entry that:

Has an 01, 66, 77, or an 88 level-number.

Describes an item whose size is variable. The size of an item is variable if the data description of any subordinate item contains Format 2 of the OCCURS clause.

Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

The number of occurrences of the subject entry is defined as follows:

In Format 1, the value of integer-1 represents the exact number of occurrences.

In Format 2, the current value of the data item referenced by data-name-1 represents the number of occurrences.

This format specifies that the subject of this entry has a variable number of occurrences. The value of integer-2 represents the maximum number of occurrences and the value of integer-1 represents the minimum number of occurrences. This does not imply that the length of the subject of the entry is variable, but that the number of occurrences is variable.

The value of the data item referenced by data-name-1 must fall within the range integer-1 through integer-2. Reducing the value of the data item referenced by data-name-1 makes the contents of data items, whose occurrence numbers now exceed the value of the data item referenced by data-name-1, unpredictable.

Where both integer-1 and integer-2 are used, the value of integer-1 must be less than the value of integer-2.

The data description of data-name-1 must describe a positive integer. Data-name-1 may be qualified.

A data description entry that contains Format 2 of the OCCURS clause may only be followed, within that record description, by data description entries which are subordinate to it.

When a group item, having subordinate to it an entry that specifies Format 2 of the OCCURS clause, is referenced, only that part of the table area that is specified by the value of data-name-1 will be used in the operation.

An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing. The index-name identified by this clause is not defined elsewhere since its allocation and format are dependent on the hardware, and not being data, cannot be associated with any data hierarchy.

The SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on an even byte boundary.

FORMAT

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

This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost and rightmost natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those natural boundaries, the unused character positions (or portions thereof) must not be used for any other data item. Such unused character positions, however, are included in:

The size of any group item(s) to which the elementary item belongs; and

The character positions redefined when this data item is the object of a REDEFINES clause.

SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left character position of the next available even byte. If the data item contains an odd number of bytes, one trailing byte of FILLER is implied.

SYNCHRONIZED not followed by either RIGHT or LEFT is equivalent to the clause SYNCHRONIZED LEFT.

SYNC is an abbreviation for SYNCHRONIZED.

This clause may only appear with an elementary item.

SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right character position of an integral even byte boundary. If the data item contains an odd number of bytes, a leading byte of FILLER is implied.

Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justifiction, truncation or overflow.

If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, then:

Each occurrence of the data item is SYNCHRONIZED.

Any implicit FILLER generated for other data items within that same table are generated for each occurrence of those data items.

Records of a file and index data items are automatically synchronized left. Records and noncontiguous data-items in working-storage begin on the next available byte unless the first elementary item is synchronized.

The format on external media of records or groups containing elementary items described with the SYNCHRONIZED clause includes any implied FILLER bytes.

When the data item preceding a data item described with the SYNCHRONIZED clause does not terminate on a byte whose address is even, then one implied FILLER byte is generated. Such automatically generated FILLER positions are included in:

The size of any group to which the FILLER item belongs; and

The number of character positions allocated when the group item of which the FILLER item is a part appears as the object of a REDEFINES clause.

The JUSTIFIED Clause

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

FORMAT

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

When a receiving data item is described with the JUSTIFIED clause and the sending data item is larger than the receiving data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and it is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space-fill for the leftmost character positions.

When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply.

The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

The JUSTIFIED clause can be specified only at the elementary item level.

JUST is an abbreviation for JUSTIFIED.

The BLANK WHEN ZERO Clause

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

FORMAT

```
    BLANK WHEN ZERO
    -----      ----
```

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited.

The BLANK WHEN ZERO clause cannot appear in the same entry with a PICTURE clause having an asterisk as the zero suppression symbol.

When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

The VALUE IS Clause

The VALUE IS clause defines the initial value of working storage items, and the values associated with a condition-name.

FORMAT 1

```
   VALUE IS literal
   -----
```

FORMAT 2

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

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

The VALUE clause cannot be stated for any items whose size is variable.

A signed numeric literal must have associated with it a signed numeric PICTURE character-string.

All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

The words THRU and THROUGH are equivalent.

The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

1. If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a working storage item, the literal is aligned in the data item according to the standard alignment rules.

2. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. Editing characters in the PICTURE clause are included in determining the size of the data item but have no effect on initialization of the data item. Therefore, the VALUE of an edited item is presented in an edited form.

Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

A figurative constant may be substituted in both Format 1 and Format 2 wherever a literal is specified.

Condition-Name Rules

In a condition-name entry, the VALUE clause is required. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

Format 2 can be used only in connection with condition-names. Wherever the THROUGH (THRU) phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

Data Description Entries Other Than Condition-Names

Rules governing the use of the VALUE clause differ with the respective sections of the Data Division:

In the File Section, the VALUE clause may be used only in condition-name entries.

In the Working-Storage Section, the VALUE clause must be used in condition-name entries. The VALUE clause may also be used to specify the initial value of any other data item; in which case the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item's description, the initial value is undefined.

In the Linkage Section, the VALUE clause may be used only in condition-name entries.

The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry that is subordinate to any entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within this group.

The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

The RENAMES Clause

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

FORMAT

```
66 data-name-1;

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

NOTE: Level-number 66, data-name-1 and the semicolon are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the RENAMES clause.

All RENAMES entries referring to data items within a given logical record must immediately follow the last data description entry of the associated record description entry.

Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items in the same logical record, and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.

Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the associated level 01 or FD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must be to the right of the end of the area described by data-name-2. Data-name-3, therefore, cannot be subordinate to data-name-2.

Data-name-2 and data-name-3 may be qualified.

None of the items within the range, including data-name-2 and data-name-3, if specified, can be an item whose size is variable as defined in the OCCURS clause.

One or more RENAMES entries can be written for a logical record.

When data-name-3 is specified, data-name-1 is a group item which includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or the last elementary item in data-name-3 (if data-name-3 is a group item).

When data-name-3 is not specified, data-name-2 can be either a group or an elementary item; when data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

The words THRU and THROUGH are equivalent.

Classes of Data

The five categories of data items (see the PICTURE Clause) are grouped into three classes:

alphabetic
numeric
alphanumeric

For alphabetic and numeric, the classes and categories are synonymous.

The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing).

Every elementary item except for an index data item belongs to one of the classes and further to one of the categories. The class of a group item is treated at object time as alphanumeric regardless of the class of elementary items subordinate to that group item.

The following chart depicts the relationship of the class and categories of data items:

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Representation of Numeric Items

The value of a numeric item may be represented in either binary, decimal or packed decimal form depending on the USAGE clause associated with the item. There are two ways of expressing decimal: DISPLAY and COMPUTATIONAL. Binary is COMPUTATIONAL-1. Packed decimal is COMPUTATIONAL-3.

The selection of the proper representation is dependent upon the usage of the numeric item. Items which must be used for input and output should be of DISPLAY usage to eliminate conversions to external forms. For efficiency of arithmetic operations, COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-3 should be used. To reduce conversions and increase efficiency, types should not be mixed in operations except where required by program needs.

Representation of Algebraic Signs

Algebraic signs fall into two categories:

> operational signs which are associated with signed numeric data items, and signed numeric literals to indicate their algebraic properties; and
>
> editing signs which appear to identify the sign of the item.

For DISPLAY, COMPUTATIONAL, and COMPUTATIONAL-3, an unsigned numeric item is assumed to have an operational sign which is positive and will receive the absolute value of signed items. A signed numeric item maintains the operational sign as a separate trailing character.

For COMPUTATIONAL-1 (which is always signed), the operational sign is maintained as part of the item in two's complement signed binary form.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

Standard Alignment Rules

The standard rules of positioning data within an elementary item depend on the category of the receiving item:

If the receiving data item is described as numeric:

a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character and is aligned as in a. above.

If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero-fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space-fill or truncation to the right, as required.

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

QUALIFICATION

Every user-specified name that defines an element in a COBOL source program must be unique, either because no other name has the identical spelling and hyphenation, or because the name exists within a hierarchy of names such that references to the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers and this process that specifies uniqueness is called qualification. Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or a level-number. Therefore, two identical data-names must not appear as entries subordinate to a group item unless they are capable of being made unique through qualification.

In the hierarchy of qualification, names associated with a level indicator are the most significant, then those names associated with level-number 01, then names associated with level-number 02, . . . , 49. The most significant name in the hierarchy must be unique and cannot be qualified.

Qualification is performed by following a data-name, by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

FORMAT 1

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

FORMAT 2

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

The rules for qualification are as follows:

1. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy.

3. If a data name is assigned to more than one data item in a source program, the data-name must be qualified each time it is referred to in the Procedure, Environment, and Data Divisions (except in the REDEFINES clause where qualification is unnecessary and must not be used.)

4. 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 from within the same section.

5. A data-name cannot be subscripted when it is being used as a qualifier.

6. A name can be qualified even though it does not need qualification: if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used. The complete set of qualifiers for a data-name must not be the same as any partial set of qualifiers for another data-name. Qualified data-names may have any number of qualifiers up to a limit of 49.

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a list of table of like elements that have not been assigned individual data-names (see The OCCURS Clause).

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may be qualified but not subscripted.

The subscript may be signed and, if signed, it must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ...n. The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

The subscript, or set of subscripts, that identifies the table element is delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization.

FORMAT

```
{data-name     } (subscript-1 [subscript-2 [,subscript-3]])
{condition-name}
```

INDEXING

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name must be initialized before it is used as a table reference. An index-name can be given an initial value by a SET statement, or a FORMAT 4 PERFORM statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when the index-name is followed by the operator + or -, followed by an unsigned integer numeric literal all delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (when the operator - is used), by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

At the time of execution of a statement which refers to an indexed table element, the value contained in the index referenced by the index-name associated with the table element must neither correspond to a value less than one (1) nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resultant from relative indexing.

FORMAT

```
{data-name}      ({index-name-1 [{+} literal-2]}
{condition-name} {literal-1      {-}           }

                [,{index-name-2 [{+} literal-4]}
                  {literal-3     {-}           }

                [,{index-name-3 [{+} literal-6]}]])
                  {literal-5     {-}           }
```

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:

FORMAT 1

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

            [,subscript-2 [,subscript-3]])]
```

FORMAT 2

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

            [,{index-name-2 [{+} literal-4]}
              {literal-3     {-}           }

            [,{index-name-3 [{+} literal-6]}]])]
              {literal-5     {-}           }
```

Restrictions on qualification, subscripting and indexing are:

A data-name must not itself be subscripted nor indexed when that data-name is being used as an index, subscript or qualifier.

Indexing is not permitted where subscripting is not permitted.

An index may be modified only by the SET and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index-names as data in a form specified by the compiler. 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

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

If qualification is used to make a condition-name unique, the associated conditional variable may 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 is exactly that of 'identifier' except that data-name-1 is replaced by condition-name-1.

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

TABLE HANDLING

Tables of data are common components of business data processing problems. Although items of data that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the item is to be repeated as many times as stated. The item is considered to be a table element and its name and description apply to each repetition or occurrence. Since each occurrence of a table element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. Subscripting and indexing are the two methods that are used to specify the occurrence number of a desired table element.

Table Definition

To define a one-dimensional table, the programmer uses an OCCURS clause as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element.

Example 1:

```
01   TABLE-1.
     02   TABLE-ELEMENT   OCCURS 20 TIMES.
          03   NAME .........
          03   SSAN .........
```

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table. In the description of a three-dimensional table, the OCCURS clause should appear in the data description of 2 nested group items which contain the element. In COBOL, tables of up to 3 dimensions are permitted.

Example 2 shows a table which has one dimension for CONTINENT-NAME, two dimensions for COUNTRY-NAME, and three dimensions for CITY-NAME and CITY-POPULATION. The table includes 100,510 data items--10 for CONTINENT-NAME, 500 for COUNTRY-NAME, 50,000 for CITY-NAME, and 50,000 for CITY-POPULATION. Within the table there are ten occurrences of CONTINENT-NAME. Within each CONTINENT-NAME there are 50 occurrences of COUNTRY-NAME and within each COUNTRY-NAME there are one hundred occurrences of CITY-NAME and CITY-POPULATION.

Example 2:

```
 01   CENSUS-TABLE.
      05 CONTINENT-TABLE  OCCURS 10 TIMES.
         10  CONTINENT-NAME  PIC XXXXXX.
         10  COUNTRY-TABLE  OCCURS 50 TIMES.
             15 COUNTRY-NAME  PIC XXXXXXXX.
             15 CITY-TABLE  OCCURS 100 TIMES.
                20  CITY-NAME  PIC XXXXXXXXXX.
                20  CITY-POPULATION  PIC 999999999999.
```

References to Table Items

Whenever the user refers to a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For access to tables of more than one dimension, an occurrence number must be supplied for each dimension of the table accessed. In Example 2 then, a reference to the 4th CONTINENT-NAME would be complete, whereas a reference to the 4th COUNTRY-NAME would not. To refer to COUNTRY-NAME, which is an element of a two-dimensional table, the user must refer to, for example, the 4th COUNTRY-NAME within the 6th CONTINENT-TABLE.

One method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element. The subscript can be represented either by a literal which is an integer or by a data-name which is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table reference must include as many subscripts as there are dimensions in the table whose element is being referenced. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself. In Example 2, references to CONTINENT-NAME require only one subscript, reference to COUNTRY-NAME requires two, and references to CITY-NAME and CITY-POPULATION require three.

When more than one subscript is required, they are written in order of successively less inclusive dimensions of the data organization. When a data-name is used as a subscript, it may be used to refer to items in many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts, for example: CITY-POPULATION (10, NEWKEY, 42).

Another method of referring to items in a table is indexing. To use this technique, the programmer assigns one or more index-names (defined with the INDEXED-BY phrase of the OCCURS clause) to an item whose data description contains an OCCURS clause. There is no separate entry to describe the index-name since its definition is completely hardware-oriented and it is not considered data per se. At object time the contents of the index-name will correspond to an occurrence number for that specific dimension of the table to which the index-name was assigned. The initial value of an index-name at object time is not determinable and the index-name must be initialized by the SET statement before use.

When a reference is made to a table element, or to an item within a table element, and the name of the item is followed by its related index-name or names in parentheses, then each occurrence number required to complete the reference will be obtained from the respective index-name. The index-name thus acts as a subscript whose value is used in any table reference that specifies indexing.

VI

PROCEDURE DIVISION

THE PROCEDURE DIVISION

The Procedure Division must be included in every COBOL source program. This division may contain declaratives and nondeclarative procedures.

The Procedure Division is identified by and must begin with the following header:

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2] ...] .
---------  --------  -----
```

The USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

Each of the operands in the USING phrase of the Procedure Division header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have a 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their descriptions given in the called program. Of those items defined in the Linkage Section only data-name-1, data-name-2, items subordinate to these data-names, and condition-names and/or index-names associated with such data-names and/or subordinate data items, may be referenced in the Procedure Division.

When the USING phrase is present, the object program operates as if data-name-1 of the Procedure Division header in the called program and data-name-1 in the USING phrase of the CALL statement in the calling program refer to a single set of data that is equally available to both the called and calling programs. Their definitions must contain the same data descriptions; however, they need not be the same name. In like manner, there is an equivalent relationship between data-name-2, ..., in the USING phrase of the called program and data-name-2, ..., in the USING phrase of the CALL statement in the calling program. A data-name must not appear more than once in the USING phrase in the Procedure Division header of the called program; however, a given data-name may appear more than once in the same USING phrase of a CALL statement.

Structure

The body of the Procedure Division must conform to one of 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] ...] ...} ...

[END PROGRAM].
 --- -------
```

FORMAT 2

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2]...].
--------- --------  -----

 {paragraph-name. [sentence] ...} ...

[END PROGRAM].
 --- -------
```

The segment-number must be an integer ranging in value from 0 through 127.

If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

Sections in the declaratives must contain segment-numbers less than 50.

All sections which have the same segment-number constitute a program segment. Sections with the same segment-number must be physically contiguous in the source program.

Segments with segment-numbers 0 through 49 belong to the fixed portion of the object program. Segments with segment-numbers 50 through 127 are independent segments. Independent segments must follow fixed segments.

Declaratives

Declarative sections must be grouped at the beginning of the Procedure Division preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES.

Procedures

A procedure is composed of a paragraph, or group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section. It consists of a paragraph-name (which may be qualified), or a section-name.

A section consists of a section header followed by zero, or more successive paragraphs. A section ends immediately before the next section or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name followed by a period and a space and by zero, or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the Procedure Division or, in the declaratives portion of the Procedure Division, at the key words END DECLARATIVES. A paragraph-name must not be duplicated within a section.

Execution

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules indicate some other order.

PROCEDURE REFERENCES

A procedure is referred to by its paragraph-name or section-name. Paragraph-names may be qualified by the section-name of the section containing the paragraph, whether or not it needs qualification. When referring to a section-name or when using a section-name as a qualifier, the word SECTION must not appear. Qualification is performed by following a paragraph-name with a section-name preceded by IN or OF. IN and OF are logically equivalent. The general format for paragraph qualification is:

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

A paragraph-name need not be qualified when referred to from within the same section or when the paragraph-name is unique.

Explicit and Implicit Transfers of Control

The mechanism that controls program flow transfers control from statement to statement in the sequence in which they were written in the source program unless an explicit transfer of control overrides this sequence or there is no next executable statement to which control can be passed. The transfer of control from statement to statement occurs without the writing of an explicit Procedure Division statement, and therefore, is an implicit transfer of control.

COBOL provides both explicit and implicit means of altering the implicit control transfer mechanism.

In addition to the implicit transfer 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 the following types of implicit control flow alterations which override the statement-to-statement transfers of control:

> If a paragraph is being executed under control of another COBOL statement (for example, PERFORM and USE) and the paragraph is the last paragraph in the range of the controlling statement, then an implied transfer of control occurs from the last statement in the paragraph to the control mechanism of the last executed controlling statement. Further, if a paragraph is being executed under the control of a PERFORM statement which causes iterative execution and that paragraph is the first paragraph in the range of that PERFORM statement, an implicit transfer of control occurs between the control mechanism associated with that PERFORM statement and

the first statement in that paragraph for each iterative execution of the paragraph.

When any COBOL statement is executed which results in the execution of a declarative section, an implicit transfer of control to the declarative section occurs. Note that another implicit transfer of control occurs after execution of the declarative.

An explicit transfer of control consists of an alteration of the implicit control transfer mechanism by the execution of a procedure branching or conditional statement. An explicit transfer of control can be caused only by the execution of a procedure branching or conditional statement. The execution of the procedure branching statement ALTER does not in itself constitute an explicit transfer of control, but affects the explicit transfer of control that occurs when the associated GO TO statement is executed.

In this document, the term 'next executable statement' is used to refer to the next COBOL statement to which control is transferred according to the rules above and the rules associated with each language element in the Procedure Division.

There is no next executable statement following:

The last statement in a declarative section when the paragraph in which it appears is not being executed under the control of some other COBOL statement. In COBOL, the result would be an implicit transfer of control to the first nondeclarative statement.

The last statement in a program when the paragraph in which it appears is not being executed under the control of some other COBOL statement. The result would be as if an implicit STOP RUN statement were executed.

SEGMENTATION

COBOL segmentation is a facility that provides a means by which the user may communicate with the compiler to specify object program overlay requirements. COBOL segmentation deals only with segmentation of procedures.

Segments

When segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program as determined by the assignment of segment numbers. All source paragraphs which contain the same segment-numbers can range from 00 through 127, it is possible to subdivide any object program into a maximum of 128 segments. Segmentation in no way affects the need for qualification of procedure-names to insure uniqueness.

Fixed Portion

The fixed portion is defined as that part of the object program which is always in memory. This portion of the program is composed of segments with segment-numbers 0 through 49.

Independent Segments

An independent segment is defined as part of the object program which can overlay, and can be overlaid by, another independent segment. An independent segment has a segment-number 50 through 127.

An independent segment is in its initial state whenever control is transferred (either implicitly or explicitly) to that segment for the first time during the execution of a program.

On subsequent transfers of control to the segment, an independent segment is also in its initial state when:

> Control is transferred to that segment as a result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.
>
> Control is transferred explicitly to that segment from a segment with a different segment-number.

On subsequent transfer of control to the segment, an independent segment is in its last-used state when control is transferred implicitly to that segment from a segment with a different segment-number.

Segmentation Classification

Sections which are to be segmented are classified using a system of segment-numbers and the following criteria:

Logic Requirements--Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections which are used less frequently are normally classified as belonging to one of the independent segments, depending on logic requirements.

Frequency of Use--Generally, the more frequently a section is referred to, the lower its segment-number; the less frequently it is referred to, the higher its segment-number.

Relationship to Other Sections -- Sections which frequently communicate with one another should be given the same segment-numbers.

Segmentation Control

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

Restrictions on Program Flow

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statements.

The ALTER STATEMENT

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

The PERFORM STATEMENT

A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

> Sections and/or paragraphs wholly contained in one or more fixed segments, or
>
> Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

> Sections and/or paragraphs wholly contained in one or more fixed segments, or
>
> Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

THE USE STATEMENT

The USE statement specifies procedures for input-output error handling that are in addition to the standard procedures provided by the input-output control system. It is a compiler directing statement required in each declarative section.

FORMAT

```
USE AFTER STANDARD {EXCEPTION}
--- -----           ---------
                   {ERROR    }
                    -----

    PROCEDURE ON {file-name-1 [,file-name-2] ...}
    ---------
                 {INPUT                         }
                  -----
                 {OUTPUT                        }
                  ------
                 {I-O                           }
                  ---
                 {EXTEND                        }
                  ------
```

A USE statement, when present, must immediately follow a section header in the declaratives section and must be followed by a period followed by a space. The remainder of the section must consist of zero, one or more procedural paragrahs that define the procedures to be used.

The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedure.

The same file-name can appear in only one USE statement.

The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

The designated procedures can be executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

After execution of a USE procedure, control is returned to the invoking routine.

Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE statement or to the procedures associated with such a USE statement.

Within a USE procedure, there must not be the execution of any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control to the invoking routine.

USE Example:

```
PROCEDURE DIVISION.
DECLARATIVES.
IO-ERROR SECTION.
     USE AFTER STANDARD ERROR PROCEDURE ON I-O.
IO-ERROR.
     DISPLAY "INPUT-OUTPUT ERROR OCCURRED".
     ACCEPT CONTINUE-FLAG POSITION ZERO.
     IF CONTINUE-FLAG = "NO" STOP RUN.
END DECLARATIVES.
```

ARITHMETIC STATEMENTS

The arithmetic statements ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT have several common features:

> The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.
>
> Arithmetic operations are calculated in either binary, decimal, packed decimal, or mixed depending on the USAGE of the operands and receiving item according to the following rules:
>
>> If the receiving data item of a divide operation is DISPLAY or COMPUTATIONAL, the operation is always calculated in decimal with any necessary conversions.
>>
>> Intermediate and final results are calculated in binary if all preceding intermediate results are binary and the next operand has COMPUTATIONAL-1 usage (except as noted in previous paragraph). Otherwise, the remaining intermediate and final results are calculated in decimal with any necessary conversions.
>
> The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is a hypothetical data item resulting from the super-imposition of specified operands in a statement aligned on their decimal points, must not contain more than eighteen decimal digits.

Arithmetic Expressions

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in Combination of Symbols in Arithmetic Expressions Table.

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

Arithmetic Operators

There are four binary arithmetic operators and two unary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space.

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

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by numeric literal +1
-	The effect of multiplication by numeric literal -1.

Formation and Evaluation Rules

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

1st - Unary plus and minus
2nd - Multiplication and division
3rd - Addition and subtraction

Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not secified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

The ways in which operators, variables, and parentheses may be combined in an arithmetic expression are summarized in the following table, where:

The letter 'P' indicates a permissible pair of symbols.

The character '-' indicates an invalid pair.

'Variable' indicates an identifier or literal.

FIRST SYMBOL	SECOND SYMBOL				
	Variable	*/-+	Unary + or -	(	)
Variable	-	P	-	-	P
* / + -	P	-	P	P	-
Unary +or-	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

An arithmetic expression may only begin with the symbol '(', '+', '-', or a variable and may only end with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items.

CONDITIONALS

The conditions are relation, class, condition-name, and switch-status. A condition has a truth value of 'true' or 'false'.

Relation Condition

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier or a literal. A relation condition has the truth value of 'true' if the relation exists between the operands.

Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The general format of a relation condition is as follows:

```
{identifier-1} {IS [NOT] GREATER THAN}{identifier-2     }
                   ---  -------
{literal-1    } {IS [NOT] LESS THAN   }{literal-2        }
                   ---  ----
{index-name-1} {IS [NOT] EQUAL TO    }{index-name-2     }
                   ---  -----
               {IS [NOT] >           }
                   ---
               {IS [NOT] <           }
                   ---
               {IS [NOT] =           }
                   ---
```

The first operand (identifier-1, literal-1 or index-name-1) is called the subject of the condition; the second operand (identifier-2, literal-2 or index-name-2) is called the object of the condition. The relation condition must contain at least one reference to a variable.

The relational operator specifies the type of comparison to be made in a relation condition. A space must precede and follow each reserved word comprising the relational operator. When used, 'NOT' and the next key word or relation character are one relational operator that defines the comparison to be executed for truth value; e.g., 'NOT EQUAL' is a truth test for an 'unequal' comparison; 'NOT GREATER' is a truth test for an 'equal' or 'less' comparison. The meaning of the relational operators is as follows:

Meaning	Relational Operator
Greater than or not greater than	IS [NOT] GREATER THAN
	IS [NOT] >
Less than or not less than	IS [NOT] LESS THAN
	IS [NOT] <
Equal to or not equal to	IS [NOT] EQUAL TO
	IS [NOT] =

NOTE: The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

Comparison of Numeric Operands

For operands whose class is numeric a comparison is made with respect to the algebraic value of the operands. The length of the literals or operands, in terms of number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

Comparison of Nonnumeric Operands

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. If one of the operands is specified as numeric, it must be an integer data item or an integer literal and:

> If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this alphanumeric data item were then compared to 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 as the numeric data item (in terms of standard data format characters), and the contents of this group item were then compared to the nonnumeric operand.

A noninteger numeric operand cannot be compared to a nonnumeric operand.

The size of an operand is the total number of standard data format characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same. There are two cases to consider: operands of equal size and operands of unequal size.

Operands of equal size: If the operands are of equal size, comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

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

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

Comparisons of Index-Names and/or Index Data Items

If two index-names are compared the result is the same as if the corresponding occurrence numbers were compared.

For an index-name and a data item (other than an index data item) or literal, the comparison is made between the occurrence number that corresponds to the value of the index-name and the data item or literal.

When a comparison is made between an index data item and an index-name or another index data item, the actual values are compared without conversion.

The result of the comparison of an index data item with any data item or literal not specified above is undefined.

Class Condition

The class condition determines whether the operand is numeric, that is, consists entirely of the characters '0', '1', '2', '3', ..., '9', with or without the operational sign; or alphabetic, that is, consists entirely of the characters 'A', 'B', 'C', ..., 'Z', space. The general format for the class condition is as follows:

```
     identifier IS [NOT] {NUMERIC   }
                    ---   -------
                         {ALPHABETIC}
                          ----------
```

The usage of the operand being tested must be described as display. When used, 'NOT' and the next key word specify one class condition that defines the class test to be executed for truth value, e.g., 'NOT NUMERIC' is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. Valid operational signs for data items are the standard data format characters, '+' and '-'.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item 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.

Condition-name (Conditional Variable)

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general-format for the condition-name condition is as follows:

```
    condition-name
```

If the condition-name is associated with a range of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

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

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

Switch-Status Condition

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

```
condition-name
```

The result of the test is true if the switch is set to the specified position corresponding to the condition-name.

Complex Conditions

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is that truth value which results from the interaction of all the stated logical operators on the individual truth values of simple conditions, or the intermediate truth values of conditions logically connected or logically negated. The logical operators and their meanings are:

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

The logical operators must be preceded by a space and followed by a space.

Negated Simple Conditions

A simple condition is negated through the use of the logical operator 'NOT'. The negated simple condition effects the opposite truth value for a simple condition. Thus the truth value of a negated simple condition is 'true' if and only if the truth value of the simple condition is 'false'; the truth value of a negated simple condition is 'false' if and only if the truth value of the simple condition is 'true'. The inclusion in parentheses of a negated simple condition does not change the truth value.

The general format for a negated simple condition is:

```
     NOT simple-condition
     ---
```

Combined and Negated Combined Conditions

A combined condition results from connecting conditions with one of the logical operators 'AND' or 'OR'. The general format of a combined condition is:

```
     condition {{AND} condition} ...
                 ---
                {OR }
                 --
```

Where 'condition' may be:

A simple condition, or

A negated simple condition, or

A combined condition, or

A negated combined condition; i.e., the 'NOT' logical operator followed by a combined condition enclosed within parentheses, or

Combinations of the above.

Although parentheses need never be used when either 'AND' or 'OR' (but not both) is used exclusively in a combined condition, parentheses may be used to affect the final truth value when a mixture of 'AND', 'OR' and 'NOT' is used.

Condition Evaluation Rules

Condition Evaluation Rules indicate the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parenthesis is to the left of its corresponding right parenthesis.

Parentheses may be used to specify the order in which individual conditions of complex conditions are to be evaluated when it is necessary to depart from the implied evaluation precedence. Conditions within parentheses are evaluated first, and, within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition. When parentheses are not used, or parenthesized conditions are at the same level of inclusiveness, the following hierarchical order of logical evluation is implied until the final truth value is determined:

Truth values for simple conditions are established.

Truth values for negated simple conditions are established.

Truth values for combined conditions are established:

'AND' logical operators, followed by
'OR' logical operators.

Truth values for negated combined conditions are established.

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

SEQUENTIAL ORGANIZATION INPUT-OUTPUT

The sequential organization input-output statements in the Prodcedure Division are the CLOSE, OPEN, READ, REWRITE, UNLOCK, USE, and WRITE statements.

Function

Sequential organization input-output provides a capability to access records of a file in established sequence. The sequence is established as a result of writing the records to the file.

Organization

Sequential files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

Access Mode

In the sequential access mode, the sequence in which records are accessed is the order in which the records were originally written.

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN and READ statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, or REWRITE statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation.

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation:

'0' - Successful Completion. The input-output statement was successfully executed.

'1' - At End. The sequential READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

'3' - Permanent Error. The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error.

'9' - General Error. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the value of status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character will contain a value as follows:

If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

When status key 1 contains a value of '3' indicating a permanent error condition, status key 2 may contain a value of '4' indicating a boundary violation. This condition indicates that an attempt has been made to write beyond the externally defined boundaries of a sequential file.

When status key 1 contains a value of '9' indicating an operating system error condition, the value of status key 2 may contain a:

> '0' indicating an invalid operation. This condition indicates that an attempt has been made to execute a READ, WRITE, or REWRITE statement that conflicts with the current open mode or a REWRITE statement not preceded by a successful READ statement.
>
> '1' indicating file not opened. This condition indicates that an attempt has been made to execute a delete, start, unlock, read, write, rewrite or close statement on a file which is not currently open.
>
> '2' indicating file not closed. This condition indicates that an attempt has been made to execute an OPEN statement on a file which is currently open.
>
> '3' indicating file not available. This condition indicates that an attempt has been made to execute an OPEN statement for a file closed WITH LOCK.
>
> '4' indicating an invalid open. This condition indicates that an attempt has been made to execute an OPEN statement for a file with no external correspondence or a file having inconsistent parameters.
>
> '5' inidcating invalid device or no next reel. This condition indicates that an attempt has been made to open a file having parameters (e.g., open mode or organization) which conflict with the device assignment (RANDOM, INPUT, PRINT, ...) or that an attempt has been made to execute a CLOSE REEL statement for the last reel/unit of a multi-reel file. In the case of a CLOSE REEL, the file has been closed.
>
> '6' indicating an undefined current record pointer status. This condition indicates that an attempt has been made to execute a READ statement after occurrence of an unsuccessful READ statement without an intervening successful CLOSE and OPEN.
>
> '7' indicating an invalid record length. This condition indicates an attempt has been made to open a file that was defined with a maximum record length different from the externally defined maximum record length, or to execute a WRITE statement that specifies a record with a length smaller than the minimum or larger than the maximum record size, or a REWRITE statement when the new record length is different from that of the record to be rewritten.

RELATIVE ORGANIZATION INPUT-OUTPUT

The Relative input-output statements in the Procedure Division are the CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK and WRITE statements.

Function

Relative input-output provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's logical position in the file.

Organization

Relative file organization is permitted only on mass storage devices (RANDOM device).

A relative file consists of records which are identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number, an integer value greater than zero. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is the tenth record area, whether or not records have been written in the first through the ninth record areas.

Access Modes

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

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

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, READ, and START statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation:

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation:

'0' - Successful Completion. The input-output was successfully executed.

'1' - At End. The statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

'2' - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

Duplicate Key
No Record Found
Boundary Violation

'3' - Permanent Error. The input-output statement was unsuccessfully executed as the result of an input-output error, such as data check, parity error, or transmission error.

'9' - General Error. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the value of status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character will contain a value as follows:

If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

When status key 1 contains a value of '2' indicating an INVALID KEY condition, status key 2 is:

'2' indicating a duplicate key value. An attempt has been made to write a record that would create a duplicate key.

'3' indicating no record found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

'4' indicating a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a file.

When status key 1 contains a value of '9' indicating an operating system error condition, the value of status key 2 is:

'0' indicating invalid operation. An attempt has been made to execute a DELETE, READ, REWRITE, START, or WRITE statement which conflicts with the current open mode of the file or a sequential access DELETE or REWRITE statement not preceded by a successful READ statement.

'1' indicating file not opened. This condition indicates that an attempt has been made to execute a delete, start, unlock, read, write, rewrite, or close statement on a file which is not currently open.

'2' indicating file not closed. An attempt has been made to execute an OPEN statement on a file that is currently open.

'3' indicating file not available. An attempt has been made to execute an OPEN statement for a file closed with lock.

'4' indicating invalid open. An attempt has been made to execute an OPEN statement for a file with no external correspondence or a file having inconsistent parameters.

'5' indicating invalid device. This condition indicates that an attempt has been made to open a file having parameters (e.g., open mode or organization) which conflict with the device assignment (RANDOM, INPUT, PRINT, ...).

'6' indicating an undefined current record pointer status. This condition indicates that an attempt has been made to execute a sequential READ statement after the occurrence of an unsuccessful READ or START statement without an intervening successful CLOSE and OPEN.

'7' indicating an invalid record length. This condition indicates that an attempt has been made to open a file that was defined with a maximum record length different from the externally defined maximum record length, or to execute a WRITE statement that specifies a record with a length smaller than the minimum or larger than the maximum record size, or a REWRITE statement when the new record length is different from that of the record to be rewritten.

The INVALID KEY Condition

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

When the INVALID KEY condition is recognized, the System takes these actions in the following order:

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

If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. When the AT END condition occurs, execution of the READ statement is unsuccessful.

INDEXED ORGANIZATION INPUT-OUTPUT

Indexed input-output statements in the Procedure Division are the CLOSE, DELETE, OPEN, READ, REWRITE, START, UNLOCK and WRITE statements.

Function

Indexed input-output provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in a nonsequential organization file is uniquely identified by a key.

Organization

A file whose organization is indexed is a mass storage file in which data records may be accessed by the value of a key. A record description may include one or more key data items, each of which is associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the recorded key for that index.

The data item named in the RECORD KEY clause of the file control entry for a file is the prime record key for that file. For purposes of inserting, updating and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique and must not be changed when updating the record.

Access Modes

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

In the random access mode, the sequence in which records are accessed is controlled by the programmer. For indexed files, the desired record is accessed by placing the value of its record key in a record key data item.

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

Current Record Pointer

The current record pointer is a conceptual entity used in this document to facilitate specification of the next record to be accessed within a given file. The concept of the current record pointer has no meaning for a file opened in the output mode. The setting of the current record pointer is affected only by the OPEN, READ, and START statements.

I-O Status

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input-output operation:

Status Key 1

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation:

'0' - Successful Completion. The input-output was successfully executed.

'1' - At End. The Format 1 READ statement was unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

'2' - Invalid Key. The input-output statement was unsuccessfully executed as a result of one of the following:

Sequence Error
Duplicate Key
No Record Found
Boundary Violation

'3' - Permanent Error. The input-output statement was unsuccessfully executed as the result of an input-output error, such as data check, parity error, or transmission error.

'9' - General Error. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the value of status key 2.

Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character will contain a value as follows:

If no further information is available concerning the input-output operation, then status key 2 contains a value of '0'.

When status key 1 contains a value of 0, indicating a successful completion, status key 2 may contain a value of 2, indicating a duplicate key. This condition indicates:

For a READ statement, the key value for the current key of reference is equal to the value of that same key in the next record within the current key of reference.

For a WRITE or REWRITE statement, the record just written created a duplicate key value for at least one alternate record key for which duplicates are allowed.

When status key 1 contains a value of '2' indicating an INVALID KEY condition, status key 2 is:

'1' indicating a sequence error for a sequentially accessed indexed file. The ascending sequence requirement of successive record key values has been violated or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

'2' indicating a duplicate key value. An attempt has been made to write a record that would create a duplicate key.

'3' indicating no record found. An attempt has been made to access a record, identified by a key, and that record does not exist in the file.

'4' indicating a boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a file.

When status key 1 contains a value of '9' indicating an operating system error condition, the value of status key 2 is:

'0' indicating invalid operation. An attempt has been made to execute a DELETE, READ, REWRITE, START, or WRITE statement which conflicts with the current open mode of the file or a sequential access DELETE or REWRITE statement not preceded by a successful READ statement.

'1' indicating file not opened. This condition indicates an attempt has been made to execute a delete, start, unlock, read, write, rewrite, or close statement on a file that is not currently open.

'2' indicating file not closed. An attempt has been made to execute an OPEN statement on a file that is currently open.

'3' indicating file not available. An attempt has been made to execute an OPEN statement for a file closed with LOCK.

'4' indicating invalid open. An attempt has been made to execute an OPEN statement for a file with no external correspondence or a file having inconsistent parameters.

'5' indicating invalid device. This condition indicates that an attempt has been made to open a file having parameters (e.g., open mode or organization which conflict with the device assignment (RANDOM, INPUT, PRINT, ...).

'6' indicating an undefined current record pointer status. This condition indicates that an attempt has been made to execute a sequential READ statement after the occurrence of an unsuccessful READ or START statement without an intervening successful CLOSE and OPEN.

'7' indicating an invalid record length. This condition indicates that an attempt has been made to open a file that was defined with a maximum record length different from the externally defined maximum record length, or to execute a WRITE statement that specifies a record with a length smaller than the minimum or larger than the maximum record size, or a REWRITE statement when the new record length is different from that of the record to be rewritten.

'8' indicating an invalid indexed file. This condition indicates that the indexed file contains inconsistent data. This is a catastrophic error from which there is no recovery at the present time.

The INVALID KEY Condition

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

When the INVALID KEY condition is recognized, the System takes these actions in the following order:

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

If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

When the INVALID KEY condition occurs, execution of the input-output statement which recognized the condition is unsuccessful and the file is not affected.

The AT END Condition

The AT END condition can occur as a result of the execution of a READ statement. When the AT END condition occurs, execution of the READ statement is unsuccessful.

The ACCEPT ... FROM Statement

The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier-1 according to the rules of the MOVE statement. DATE, DAY, and TIME are conceptual data items and, therefore, are not described in the COBOL program.

FORMAT

```
    ACCEPT identifier-1 FROM {DATE}
    ------              ---- ----
                             {DAY }
                              ---
                             {TIME}
                              ----
```

DATE is composed of the data elements year of century, month of year, and day of month. The sequence of the data element codes is from high order to low order (left to right), year of century, month of year, and day of month. Therefore, July 1, 1979 would be expressed as 790701. DATE, when accessed by a COBOL program behaves as if it had been described in the COBOL program as an unsigned elementary numeric integer data item six digits in length.

DAY is composed of the data elements year of century and day of year. The sequence of the data element codes is from high order to low order (left to right) year of century, day of year. Therefore, July 1, 1979 would be expressed as 79181. DAY, when accessed by a COBOL program as an unsigned elementary numeric integer data item five digits in length.

TIME is composed of the data elements hours, minutes, seconds and hundredths of a second. TIME is based on elapsed time after midnight on a 24-hour clock basis--thus, 2:41 p.m. would be expressed 14410000. TIME, when accessed by a COBOL program behaves as if it had been described in a COBOL program as an unsigned elementary numeric integer data item eight digits in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595999.

ACCEPT ... FROM Examples

```
ACCEPT  YEAR-DAY FROM DAY.
ACCEPT  CLOCK FROM TIME.
```

The ACCEPT Statement (Terminal I-O)

The ACCEPT statement causes low volume data to be accepted from the CRT terminal and transferred to the specified data item. ACCEPT statement phrases allow the specification of position, form and format of the accepted data.

FORMAT

```
    ACCEPT {identifier-1 [,UNIT {identifier-2}]
    ------                 ---- {literal-1   }

             [,LINE {identifier-3}] [,POSITION {identifier-4}]
               ---- {literal-2   }    -------- {literal-3   }

             [,SIZE {identifier-5}] [,PROMPT [literal-5]
               ---- {literal-4   }    ------

             [,ECHO] [,CONVERT] [,TAB] [,ERASE] [,NO BEEP]
               ----    -------    ---    -----    -- ----

             [,OFF]  [,{HIGH}]  [,BLINK]  [,REVERSE]
               ---       ----     -----     -------
                        {LOW }
                         ---

             [,ON EXCEPTION identifier-6 imperative-statement]}....
               -- ---------
```

The ACCEPT statement causes the transfer of data from the CRT device. This data replaces the contents of the data item named by identifier-1. The receiving data item must have usage DISPLAY if ECHO is specified; otherwise, it may have any usage except INDEX.

When an ACCEPT statement contains more than one operand, the values are transferred in the sequence in which the operands are encountered. ACCEPT phrases apply to the previously specified identifier-1 only. A subsequent identifier-1 in the same ACCEPT statement will be treated as if no previous phrases have been specified.

An ACCEPT statement may contain no more than one ON EXCEPTION phrase, and if present it must be associated with the last (or only) identifier-1.

> Note: Features which require support of the host operating system and/or terminal hardware may not be supported on all systems. Any features which are not supported will compile correctly, but will be ignored at runtime. See the User's Guide for specific details.

The UNIT Phrase

The UNIT phrase must be the first phrase if used. The other phrases may be written in any order.

The value of identifier-2 or literal-1 in the UNIT phrase specifies the station identifier of the CRT from which the data is to be accepted. If the UNIT phrase is omitted, the CRT which executed the program will be accessed.

The LINE Phrase

The value of identifier-3 or literal-2 in the LINE phrase specifies the line number from which the data is to be accepted from the screen of the CRT terminal, with 1 being the top line. If the value is greater than the number of lines on the CRT screen, it is adjusted to the maximum line number.

If the value is zero or the LINE phrase is not present in an ACCEPT statement, then data is to be accepted from the next line below the current position of the cursor on the CRT screen unless the value specified in the POSITION phrase is also zero, in which case the data is to be accepted from the line at the current position of the cursor on the CRT screen.

The POSITION Phrase

The value of identifier-4 or literal-3 in the POSITION phrase specifies the number of the character positions to which the cursor is to be positioned within the specified line prior to the accepting of data from the CRT terminal, with 1 being the leftmost character position within a line. If the value is greater than the maximum number of characters within a line on the CRT screen, it is adjusted to the maximum character number.

If the POSITION phrase is not specified, a value of 1 is assumed for the first accepted operand and 0 for each additional operand accepted in the same statement. If a value of 0 is specified, the data is to be accepted starting at the next field on the CRT screen (starting character position plus size of last ACCEPT or DISPLAY).

The SIZE Phrase

The value of identifier-5 or literal-4 in the SIZE phrase specifies the maximum number of characters to be accepted from the CRT terminal, overriding the Data Division definition of the field. If the SIZE phrase is not present or a value of 0 is specified, then the size of identifier-1, (identifier-5, ...) is used. A size greater than 80 is treated as equal to 80.

The size of the accepted field is determined by the SIZE phrase. The number of characters transferred from the CRT is less than or equal to the size of the accepted field. Input is terminated by depression of the return key (which is not considered part of the input). The number of characters actually input is the size of the source in the following:

> If the receiving item is not numeric, the accepted input is stored according to the rules of the MOVE statement for an alphanumeric source and destination. If the receiving item is described JUSTIFIED RIGHT, the clause will apply to the MOVE rules.
>
> If the receiving item is numeric, the accepted input is stored according to the rules of the MOVE statement for a numeric source and destination. If the CONVERT phrase is not specified, the source has the same scale as the receiving item. If the receiving item has a trailing sign and the CONVERT phrase is not specified, the input must contain digits followed by a sign character. If the CONVERT phrase is specified, then the input is converted according to the rules of the CONVERT phrase. The CONVERT phrase is recommended when accepting numeric items.

The PROMPT Phrase

The presence of the key word PROMPT in an ACCEPT statement causes the data to be accepted with prompting. The action of prompting is to display fill characters on the CRT screen in the positions from which data is to be accepted. Literal-5 must be a single character nonnumeric literal which specifies the fill character to be used in prompting. If literal-5 is omitted in the PROMPT phrase, then an underscore will be used as the fill character.

When the PROMPT phrase is not specified, then the data is to be accepted without prompting; the original contents of the field on the CRT will be undisturbed before accepting input.

The ECHO Phrase

The presence of the key word ECHO within an ACCEPT statement causes the contents of identifier-1 to be displayed on the screen of the CRT terminal. Conversion (see CONVERT Phrase), decimal alignment, and justification are performed prior to display. If the specified size is greater than the size of the receiving data-item, the data-item is displayed right justified in the accept field with leading blanks. If the specified size is less than the size of the receiving data-item, the display is truncated on the right. When the ECHO phrase is not specified, the original input data remains in the accept field.

The CONVERT Phrase

If the receiving data-item is numeric, the presence of the key word CONVERT within an ACCEPT statement causes the conversion of an accepted field to a trailing-signed decimal field. The trailing-sign decimal field is then stored in identifier-1. The conversion is accomplished by a left-to-right scan and the rules:

Set the sign according to the rightmost sign given in the input or positive if no sign is present.

Set the scale according to the rightmost period given in the input or to zero if no period is present. If the DECIMAL POINT IS COMMA clause was specified in the source program, a comma replaces the period in determining the scale.

Delete all nonnumeric characters from the accepted field.

When the CONVERT phrase is not specified, or the receiving data-item is not numeric, then the data is to be stored without the above conversion.

The TAB Phrase

The presence of the key word TAB in an ACCEPT statement causes a wait for a tab, return or backspace key in reaching the end of the input field; the return will then terminate input, the backspace character will position the cursor back one character, the tab will reposition the cursor to the beginning of the field and all other input will be ignored. If the key word TAB is omitted, input will automatically be terminated if the end of the input field is encountered.

The ERASE Phrase

The presence of the key word ERASE within an ACCEPT statement causes the screen of the CRT to be erased prior to cursor positioning. When the ERASE phrase is not specified, then the screen is not erased prior to cursor positioning.

The NO BEEP Phrase

The presence of the key words NO BEEP in an ACCEPT statement causes supression of the beep signal upon cursor positioning. If the key words NO BEEP are omitted, a beep signal will occur upon cursor positioning prior to data input.

The OFF Phrase

The presence of the key word OFF within an ACCEPT statement causes data to be input from the terminal keyboard but not displayed to the screen. Blank characters are displayed to the screen in lieu of data characters.

The HIGH/LOW Phrase

The presence of the key word HIGH or LOW causes the PROMPT character and the accepted data (if CONVERT and/or ECHO was specified) to be displayed at the specified intensity.

When HIGH or LOW is not specified, the default display is HIGH.

The BLINK Phrase

The presence of the key word BLINK causes the PROMPT character, and any displayed data, to be BLINKed. When BLINK is not specified, no BLINK is provided.

The REVERSE Phrase

The presence of the key word REVERSE causes the PROMPT character, and any displayed data, to be displayed in a reverse image mode. When REVERSE is not specified, normal display is provided.

The ON EXCEPTION Phrase

The presence of ON EXCEPTION causes the imperative-statement to be executed if an invalid character is entered. The invalid character (in ASCII format) will be placed in identifier-6 prior to execution of the imperative-statement. The invalid character may be determined by declaring identifier-6 as USAGE COMP-1 and testing for its ASCII value.

When ON EXCEPTION and CONVERT are both specified and a conversion error occurs, an error code of "98" is returned in identifier-6.

ACCEPT Examples

```
     ACCEPT  ANSWER-1, ANSWER-2.

     ACCEPT  START-VALUE LINE 1, POSITION K,
             PROMPT, ECHO, CONVERT.

     ACCEPT  NEXT-N POSITION 0,
             PROMPT, ECHO.

     ACCEPT  YEAR, LINE YR-LN, POSITION YR-POS;
             MONTH, LINE MN-LN, POSITION MN-POS.
```

The ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

FORMAT 1

```
    ADD {identifier-1} [,identifier-2] ...
    ---
        {literal-1    } [,literal-2    ]

        TO identifier-m [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]
        ------              -------

        [;ON SIZE ERROR imperative-statement]
             ---- -----
```

FORMAT 3

```
    ADD {CORRESPONDING} identifier-1 TO identifier-2 [ROUNDED]
    --- -------------                --               -------
        {CORR          }
         ----

        [; ON SIZE ERROR imperative-statement]
              ---- -----
```

In Format 1, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-m storing the result immediately into identifier-m.

In Format 2, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m.

In Formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 identifier-m following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

In Format 3, data items in identifier-1 are added to and stored in the corresponding data items in identifier-2.

In Format 3, each identifier must refer to a group item.

Each literal must be a numeric literal.

The ROUNDED Phrase

The ADD statement may optionally include the ROUNDED phrase.

If, after decimal point alignment, the number of places in the fraction of the result of the arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant identifier are represented by the character 'P' in the picture for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

The SIZE ERROR Phrase

If, after appropriate decimal point alignment, the absolute value of the result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the CORRESPONDING phrase is specified, and any of the individual additions produces a size error condition, the imperative-statement is not executed until all of the individual additions are completed.

If the resultant-identifier has COMPUTATIONAL-3 usage, size error is correctly detected only for data items declared with an odd length picture clause. Therefore all COMP-3 data items should be declared with an odd number of character positions.

If the SIZE ERROR phrase is not specified and a size error condition exists, the value of the resultant-identifier is undefined.

If the SIZE ERROR phrase is specified and a size error condition exists, the value of the resultant-identifier is not altered and the imperative statement of the SIZE ERROR phrase is executed.

The CORRESPONDING Phrase

If the CORRESPONDING phrase is used, selected items within identifier-1 are ADDed to, and the result stored in, the corresponding items in identifier-2.

Data items referenced by the CORRESPONDING phrase must adhere to the following rules:

> A data item in identifier-1 and a data item in identifier-2 must not be designated by the key word FILLER and must not have the same data-name and the same qualifiers up to, but not including, identifiers-1 and identifier-2.
>
> Both of the data items must be elementary numeric data items.
>
> The description of identifier-1 and identifier-2 must not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.
>
> A data item that is subordinate to identifier-1 or identifier-2 and contains a REDEFINES, RENAMES, OCCURS or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, identifier-1 and identifier-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

CORR is an abbreviation for CORRESPONDING.

ADD Examples

```
ADD SALARY TO SALARY.
     (doubles the value of SALARY)

ADD JOHNS-PAY, PAULS-PAY, ALBERTS-PAY
     GIVING COMPANY-PAY.

ADD ACCELERATION TO VELOCITY ROUNDED
     ON SIZE ERROR GO TO SOUND-BARRIER.

ADD CORRESPONDING ELEMENT (X)
     TO ELEMENT (Y).

ADD CORR SUB-TOTAL-RECORD TO TOTAL-RECORD ROUNDED
     ON SIZE ERROR GO TO ERR.
```

The ALTER Statement

The ALTER statement modifies a predetermined sequence of operations.

FORMAT

```
    ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
    -----                  --  -------  --

         [,procedure-name-3 TO [PROCEED TO] procedure-name-4]...
                            --  -------  --
```

Each procedure-name-1, procedure-name-3, ..., is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

Each procedure-name-2, procedure-name-4, ..., is the name of a paragraph or section in the Procedure Division.

Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3,..., so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2, procedure-name-4,..., respectively. Modified GO TO statements in independent segments may, under some circumstances, be returned to their initial states.

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number.

The CALL Statement

The CALL statement causes control to be transferred from one object program to another, within the run unit.

FORMAT

```
    CALL {identifier-1} [USING data-name-1 [,data-name-2] ...]
    ---- {literal-1    }  -----
```

The execution of a CALL statement causes control to pass to the program whose name is specified by the value of literal-1 or identifier-1, the 'called' program.

Literal-1 must be a nonnumeric literal.

Identifier-1 must be defined as an alphanumeric data item such that its value can be a program name.

The called program can be another COBOL program or an assembly language program. Refer to the User's Guide for specific details.

Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

The CALL statement may appear anywhere within a segmented program. When a CALL statement appears in a section with a segment-number greater than or equal to 50, the EXIT PROGRAM statement returns control to the calling program.

The USING Phrase

The data-names specified by the USING phrase of the CALL statement indicate those data items available to a calling program that may be referred to in the called program. The order of appearance of the data-names in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Corresponding data-names refer to a single set of data which is available to the called and calling program. The correspondence is positional, not by name. In the case of index-names, no such correspondence is established. Index-names in the called and calling program always refer to separate indices.

The USING phrase is included in the CALL statement only if there is a USING phrase in the Procedure Division header of the called program, and the number of operands in each USING phrase must be identical.

Each of the operands in the USING phrase must have been defined as a data item in the File Section, Working-Storage Section, or Linkage Section, and must have a level-number of 01 or 77. Data-name-1, data-name-2, ..., may be qualified when they reference data items defined in the File Section.

CALL Examples:

```
     CALL "SUBPRG1".

     CALL REORDER
          USING TABLE, INDEX-1, RESULT.
```

The CLOSE Statement (Sequential I-O)

The CLOSE statement terminates the processing of files.

FORMAT

```
    CLOSE file-name-1 [{REEL} [WITH NO REWIND]]
    -----              ----        -- ------
                       {UNIT}
                        ----

                      [WITH {NO REWIND}     ]
                            -- ------
                            {LOCK      }
                             ----

        [,file-name-2 [{REEL} [WITH NO REWIND] ] ] ...
                        ----       -- ------
                       {UNIT}
                        ----

                      [WITH {NO REWIND}     ]
                            -- ------
                            {LOCK      }
                             ----
```

The function of a CLOSE statement (with no options) is to cause the operating system to close the file. For files opened for OUTPUT, the operating system also writes an EOF as it closes the file.

If a STOP RUN statement is executed prior to closing the file, the operating system will close the file without an EOF.

A CLOSE statement may only be executed for a file in an open mode.

Once a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

The execution of a CLOSE statement causes the value of the FILE STATUS data-item, if any, associated with file-name-1 (file-name-2, ...) to be updated.

The REEL and UNIT Phrases

The CLOSE REEL and CLOSE UNIT statements are documentary only and may be included or omitted at the user's discretion.

The NO REWIND Phrase

CLOSE WITH NO REWIND prevents page advancing on files assigned to the printer. It has no effect on other file.

The LOCK Phrase

The function of the CLOSE WITH LOCK statement is to perform the CLOSE function and set a flag to prevent the file from being OPENed again during execution of this program.

CLOSE Examples

```
    CLOSE  TRANSACTION-FILE.

    CLOSE  DATA-BASE WITH LOCK.

    CLOSE  PRINT-FILE WITH NO REWIND.
```

The CLOSE Statement (Relative and Indexed I/O)

The CLOSE statement terminates the processing of files.

FORMAT

```
    CLOSE file-name-1 [WITH LOCK]
    -----                   ----

            [,file-name-2 [WITH LOCK]] ...
                                ----
```

The function of a CLOSE statement (with no options) is to cause the operating system to close the file. For files opened for OUTPUT, the operating system also writes an EOF prior to closing the file.

If a STOP RUN statement is executed prior to closing the file, the operating system will close the file without an EOF.

The files referenced in the CLOSE statement need not all have the same organization or access.

A CLOSE statement may only be executed for a file in an open mode.

If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

The execution of the CLOSE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name-1 (file-name-2, ...) to be updated.

The LOCK Phrase

The function of the CLOSE WITH LOCK statement is to perform the CLOSE function and set a flag to prevent the file from being OPENed during the execution of the program.

CLOSE Examples:

```
        CLOSE TRANSACTION-FILE.

        CLOSE DATA-BASE WITH LOCK.
```

The COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to a data item.

FORMAT

```
   COMPUTE identifier-1 [ROUNDED] = arithmetic-expression
   -------              -------

            [; ON SIZE ERROR imperative-statement]
                  ---- -----
```

Identifier-1 must refer to either an elementary numeric item or an elementary numeric edited item.

An arithmetic expression consisting of a single identifier or literal provides a method of setting the value of identifier-1 equal to the value of the single identifier or literal.

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY and DIVIDE.

Note: Exponentiation is not supported.

The ROUNDED Phrase

The COMPUTE statement may optionally include the ROUNDED phrase. If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the identifier-1, truncation is relative to the size provided for the identifier-1. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in an identifier-1 are represented by the character 'P' in the picture for that identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

The SIZE ERROR Phrase

If, after appropriate decimal point alignment, the absolute value of the result exceeds the largest value that can be contained in identifier-1, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If identifier-1 has COMPUTATIONAL-3 usage, size error is detected only for data items declared with an odd length picture clause. Therefore all COMP-3 data items should be declared with an odd number of character positions.

Division by zero always causes a size error condition.

If the SIZE ERROR phrase is not specified and a size error condition exists, the value of the identifier-1 is undefined.

If the SIZE ERROR phrase is specified and a size error condition exists, the value identifier-1 is not altered and the imperative-statement in the SIZE ERROR phrase is executed.

COMPUTE Examples

```
    COMPUTE SALARY ROUNDED = WAGES * HOURS.

    COMPUTE SECONDS = (((HRS * 60) + MIN) * 60) + SEC.

    COMPUTE AVERAGE = TOTAL / KOUNT
            ON SIZE ERROR MOVE 0 TO AVERAGE.

    COMPUTE PAY (DATE) ROUNDED

            = RATE * 8.
```

The DELETE Statement (Relative and Indexed I-O)

The DELETE statement logically removes a record from a mass storage file.

FORMAT

```
    DELETE file-name RECORD [; INVALID KEY imperative-statement]
    ------                     -------
```

After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

The associated file must be opened in the I-O mode at the time of execution of this statement.

For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The system logically removes from the file the record that was accessed by that READ statement.

For a file in random or dynamic access mode, the system logically removes from the file that record identified by the contents of the key data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated.

The INVALID KEY Phrase

The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

The current record pointer is not affected by the execution of a DELETE statement.

The DISPLAY Statement

The DISPLAY statement causes low volume data to be displayed on the specified CRT terminal. DISPLAY statement phrases allow the specification of position, form and format of the displayed data.

FORMAT

```
   DISPLAY {{identifier-1} [,UNIT {identifier-2}]
   -------                   ----
            {literal-1   }        {literal-2   }

          [,LINE {identifier-3}] [,POSITION {identifier-4}]
            ----                   --------
                 {literal-3   }              {literal-4   }

          [,SIZE {identifier-5}] [,BEEP] [,ERASE]}
            ----                   ----    -----
                 {literal-5   }

          [,{HIGH}] [,BLINK] [,REVERSE]} ...
             ----     -----    -------
            {LOW }
             ---
```

The DISPLAY statement causes the contents of each operand (identifier-1 or literal-1) to be transferred to the CRT device in the order listed. The sending data item must have DISPLAY usage.

When a DISPLAY statement contains more than one operand, the values of the operands are transferred in the sequence in which the operands are encountered.

> Note: Features which require support of the host operating system and/or terminal hardware may not be supported on all systems. Any features which are not supported will compile correctly, but will be ignored at runtime. See the User's Guide for specific details.

The UNIT Phrase

The UNIT phrase, if specified, must be written first. The other phrases may be written in any order.

The value of identifier-2 or literal-2 in the UNIT phrase specifies the station identifier of the CRT upon which the data is to be displayed. If the UNIT phrase is omitted, the CRT which executed the program will be accessed.

The LINE Phrase

The value of identifier-3 or literal-3 in the LINE phrase specifies the line number upon which the data is to be displayed on the screen of the CRT terminal, with one being the top line. If the value is greater than the number of lines on the CRT screen, it is adjusted to the maximum line number. If the value is zero or the LINE phrase is not present in a DISPLAY statement, then data is to be displayed on the next line below the current position of the cursor on the CRT screen unless the value specified in the POSITION phrase is also zero, in which case the data is to be displayed on the line at the current position of the cursor on the CRT screen. If incrementing to the next line generates a line number greater than the maximum number of lines on the CRT screen, the new line is displayed at the bottom.

The POSITION Phrase

The value of identifier-4 or literal-4 in the POSITION phrase specifies the number of the character to which the cursor is to be positioned within the specified line prior to the displaying of data on the screen of the CRT terminal, with 1 being the leftmost character position within a line. If the value is greater than the maximum number of characters within a line on the CRT screen, it is adjusted to the maximum character number.

If the POSITION phrase is not specified, a value of one is assumed for the first displayed operand and zero for each additional operand displayed in he same statement. If a value of zero is specified, the data is to be displayed starting at the next field on the CRT screen (starting character position plus size of the last ACCEPT or DISPLAY).

The SIZE Phrase

The value of identifier-5 or literal-5 in the SIZE phrase specifies the number of characters to be displayed on the screen of the CRT terminal, overriding the Data Division definition of the field. If the SIZE phrase is not present or a value of zero is specified, the size of identifier-1 or literal-1 is used. If literal-1 is a figurative constant, the literal has a size of one. A size greater than 80 is treated as equal to 80.

If the size of the display field is less than the size of the sending data item, only the leftmost characters are displayed. If the specified size is greater than the size of the sending date item, the results are unpredictable. If the sending item is a figurative constant, the constant fills the display field. No conversions are made in the transfer to the display field.

The BEEP Phrase

The presence of the key word BEEP within a DISPLAY statement causes a beep signal to occur on cursor positioning prior to the display of the data. If the BEEP key word is omitted, no signal is given on cursor positioning.

The ERASE Phrase

The presence of the key word ERASE within a DISPLAY statement causes the screen of the CRT terminal to be erased before the content of identifier-1 or literal-1 is displayed on the screen. When the ERASE phrase is not specified, then the screen is not erased prior to the display of the data.

The HIGH/LOW Phrase

The presence of HIGH or LOW causes the data to be displayed at the specified intensity. When HIGH or LOW is not specified, the default display is HIGH.

The BLINK Phrase

The presence of thekey word BLINK causes the displayed data to be BLINKed. the normal mode is no blink.

The REVERSE Phrase

The REVERSE key word causes the data to be displayed in REVERSE video. The normal mode is no reverse.

DISPLAY Examples

```
DISPLAY "FLIGHT ARRIVING AT GATE", LINE FLT-LN,
     POSITION 1, ERASE; GATE-NUMBER, HIGH, BLINK.

DISPLAY "ENTER JOB CODE: ".

DISPLAY CRT-HEADER LINE 1 ERASE.

DISPLAY ZEROES SIZE 5.

DISPLAY QUOTE.
```

The DIVIDE Statement

The DIVIDE statement divides one numeric data item into another and stores the quotient.

FORMAT 1

```
   DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
   ------                ----                -------
          {literal-1   }

       [;ON SIZE ERROR imperative-statement]
            ---- -----
```

FORMAT 2

```
   DIVIDE {identifier-1} INTO {identifier-2}
   ------                ----
          {literal-1   }      {literal-2   }

       GIVING identifier-3 [ROUNDED]
       ------               -------

       [;ON SIZE ERROR imperative-statement]
            ---- -----
```

FORMAT 3

```
   DIVIDE {identifier-1} BY {identifier-2}
   ------                --
          {literal-1   }    {literal-2   }

       GIVING identifier-3 [ROUNDED]
       ------               -------

       [;ON SIZE ERROR imperative-statement]
            ---- -----
```

In Format 1, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient.

In Format 2, the value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2 and the result is stored in identifier-3.

In Format 3, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3.

Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING phrase must refer to either an elementary numeric item or an elementary numeric edited item.

Each literal must be a numeric literal.

The ROUNDED Phrase

The DIVIDE statement may optionally include the ROUNDED phrase.

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant identifier are represented by the character 'P' in the picture for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

The SIZE ERROR Phrase

If, after appropriate decimal point alignment, the absolute value of the result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the resultant-identifier has COMPUTATIONAL-3 usage, size error is detected only for data items declared with an odd length picture clause. Therefore all COMP-3 data items should be declared with an odd number of character positions.

Division by zero always causes a size error condition.

If the SIZE ERROR phrase is not specified and a size error condition exists, the value of the resultant-identifier is undefined.

If the SIZE ERROR phrase is specified and a size error condition exists, the value of the resultant-identifier is not altered and the imperative statement in the SIZE ERROR phrase is executed.

DIVIDE Examples

```
    DIVIDE 10 INTO TOTAL-WORK-LOAD
         GIVING MORRISS-WORK-LOAD

    DIVIDE TOTAL-WORK-LOAD BY 2.5
         GIVING ALFREDS-WORK-LOAD ROUNDED
         ON SIZE ERROR GO TO ALFRED-QUIT.

    DIVIDE 2.5 INTO TOTAL.
```

The EXIT Statement

The EXIT statement provides a common end point for a series of procedures or the logical end of a called program.

FORMAT

```
    EXIT [PROGRAM].
    ----  -------
```

The EXIT statement must appear in a sentence by itself.

The EXIT sentence must be the only sentence in the paragraph.

An EXIT statement without the word PROGRAM serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation or execution of the program.

An execution of an EXIT PROGRAM statement in a CALLED program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement without the word PROGRAM.

The GO TO Statement

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

FORMAT 1

```
    GO TO procedure-name-1.
    --
```

FORMAT 2

```
    GO TO procedure-name-1 [,procedure-name-2] ...,
    --

        procedure-name-n DEPENDING ON identifier-1.
                         ---------
```

If a Format 1 GO TO statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

When a Format 1 GO TO statement is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format-1 GO TO statement.

The DEPENDING ON Phrase

When a Format 2 GO TO statement is executed, control is transferred to procedure-name-1, procedure-name-2, etc., depending on the value of the identifier-1 being 1, 2, ..., n. If the value of the identifier-1 is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

Identifier-1 is the name of a numeric integer elementary item.

The IF Statement

The IF statement causes a specified condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

FORMAT

```
    IF condition; {statement-1   } {;ELSE statement-2   }
    --                               ----
                  {NEXT SENTENCE} {;ELSE NEXT SENTENCE}
                   ---- --------    ---- ---- --------
```

Statement-1 and statement-2 represent either an imperative statement or a conditional statement, and either may be followed by a conditional statement.

When an IF statement is executed, the following transfers of control occur:

> If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

> If the condition is true and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

If the condition is false, statement-1 or its surrogate NEXT SENTENCE is ignored, and statement-2, if specified, is executed. If statement-2 contains a procedure branching or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching or conditional statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

If the condition is false, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

Statement-1 and/or statement-2 may contain an IF statement. In this case the IF statement is said to be nested.

IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not been already paired with an ELSE.

The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.

IF Examples

```
IF CHAR-STR IS ALPHABETIC,
    MOVE CHAR-STR TO ALPHA-STR;
ELSE IF CHAR-STR IS NUMERIC,
        MOVE CHAR-STR TO NUM;
        DISPLAY NUM;
     ELSE NEXT SENTENCE.

IF NUM = OLD-NUM GO TO RE-SET.

IF ALPHA-STR NOT = "TEST"
   ADD 1 TO ERROR-CNT.

IF NUM < LIMIT, ADD 1 TO NUM.

IF NUM IS LESS THAN LIMIT
   ADD 1 TO NUM.

IF PRINT-SWITCH PERFORM PRINT-ROUTINE.
```

The INSPECT Statement

The INSPECT statement provides the ability to tally (Format 1), replace (Format 2), or tally and replace (Format 3) occurrences of single characters or groups of characters in a data item.

FORMAT 1

```
    INSPECT identifier-1
    -------
         TALLYING identifier-2 FOR  {{ALL     } {identifier-3}}
         --------                 ---      ---       {literal-1   }
                                   {{LEADING}                   }
                                     -------
                                   {       CHARACTERS           }
                                           ----------
              [{BEFORE} INITIAL {identifier-4]}]
                ------          {literal-2    }
               {AFTER }
                -----
```

FORMAT 2

```
    INSPECT identifier-1
    -------
          REPLACING  {{ALL     } {identifier-5}} BY {identifier-6}
          ---------     ---      {literal-3    }  -- {literal-4    }
                     {{LEADING}                }
                        -------
                     {{FIRST   }                }
                        -----
                     {       CHARACTERS         }
                             ----------
               [{BEFORE} INITIAL {identifier-7}]
                 ------          {literal-5    }
                {AFTER }
                 -----
```

FORMAT 3

```
   INSPECT identifier-1
   -------

       TALLYING identifier-2 FOR {{ALL     } {identifier-3}}
       --------                  ---           {literal-1    }
                                   {{LEADING}                }
                                      -------
                                   {    CHARACTERS           }
                                        ----------
        [{BEFORE} INITIAL {identifier-4}]
          ------          {literal-2    }
         {AFTER }
          -----

       REPLACING  {{ALL     } {identifier-5}} BY {identifier-6}
       ---------    ---       {literal-3    }    {literal-4    }
                  {{LEADING}                }
                     -------
                  {{FIRST  }                }
                     -----
                  {       CHARACTERS        }
                          ----------

            [{BEFORE} INITIAL {identifier-7}]
              ------          {literal-5    }
             {AFTER }
              -----
```

Identifier-1 must reference either a group item or any category of elementary item, described (either implicitly or explicitly) as usage is DISPLAY.

Identifier-3 ... identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (either implicitly or explicitly) as usage is DISPLAY and a size of one character.

Each literal may be either a figurative constant (which is treated as a one-character data item) or a nonnumeric literal one character in length.

The general rules that apply to the INSPECT statement are:

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 4 through 6.

2. For use in the INSPECT statement, the contents of the data item referenced by identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 will be treated as follows:

   a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.

   b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though it had been redefined as alphanumeric (see general rule 2a) and the INSPECT statement had been written to reference the redefined data item.

   c. If any of the identifier-1, identifier-3, identifier-4, identifier-5, identifier-6, or identifier-7 are described as signed numeric, the data item is inspected as though it had been moved to an unsigned numeric data item of the same length and then the rules in general rule 2b had been applied. (See the MOVE statement.)

3. In general rules 4 through 10 all references to literal-1, literal-2, literal-3, literal-4, and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6, and identifier-7, respectively.

4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Formats 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Formats 2 and 3).

5. The comparison operation to determine the occurrences of literal-1 to be tallied and/or occurrences of literal-3 to be replaced, occurs as follows:

   a. The character specified by literal-1, literal-3 is compared to successive characters, starting with the leftmost character position in the data item referenced by identifier-1. Literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal.

b. If no match occurs in the comparison of literal-1, literal-3, the comparison is repeated starting with the next character position of identifier-1.

c. Whenever a match occurs, tallying and/or replacing takes place as described in general rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the character position that caused the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with literal-1, literal-3.

d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in paragraphs 5a through 5d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in general rule 5 is affected by the BEFORE and AFTER phrases as follows:

   a. If the BEFORE and AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operation as described in general rule 5.

b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not including the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5, within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3, or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Format 1

7. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.

8. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

   b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

   c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each character matched, in the sense of general rule 5e, within the contents of the data item referenced by identifier-1.

Format 2

9. The rules for replacement are as follows:

   a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 5e, in the contents of the data item referenced by identifier-1 is replaced by literal-4.

   b. When ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

   c. When LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

   d. When FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

Format 3

10. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, and the other statement being a Format 2 statement with REPLACING phrases identical to those specified in the Format 3 statement. The general rules given for matching and counting apply to the Format 1 statement and the general rules given for matching and replacing apply to the Format 2 statement.

INSPECT Examples:

```
INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A",

  Where word=LARGE, count=1.
  Where word=ANALYST, count=0.

INSPECT word TALLYING count FOR LEADING "A" BEFORE INITIAL "L".

  Where word=LARGE, count=0.
  Where word=ANALYST, count=1.

INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY
"E" AFTER INITIAL "L".

  Where word=CALLAR, count=2, word=CALLER.
  Where word=SALAMI, count=1, word=SALEMI.
  Where word=LATTER, count=1, word=LETTER.

INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

  Where word=ARXAX, word=GRXAX.
  Where word=HANDAX, word=HGNDGX.

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J"
REPLACING ALL "A" BY "B".

  Where word=ADJECTIVE, count=6, word=BDJECTIVE.
  Where word=JACK, count=3, word=JBCK.
  Where word=JUJMAB, count=5, word=JUJMBB.

INSPECT word REPLACING ALL "W" BY "Q" AFTER
INITIAL "R".

  Where word=RXXBQWY, word=RXXBQQY.
  Where word=YZACDWBR, word=YZACDWBR.
  Where word=RAWRXEB, word=RAQRXEB.

INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

  word before:  12 XZABCD
  word after:   BBBBBABCD
```

The MOVE Statement

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

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 represent the sending area; identifier-2, identifier-3, ..., represent the receiving area(s).

An index data item cannot appear as an operand of a MOVE statement.

The data designated by literal-1 or identifier-1 is moved first to identifier-2, then to identifier-3, ... The rules referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2, ..., is evaluated immediately before the data is moved to the respective data item.

Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands. 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).
```

Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move between these categories:

1. The figurative constant SPACE, a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

2. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

3. A non integer numeric literal or a non integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

4. All other elementary moves are legal and are performed according to the rules given below.

Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

1. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space-filling takes place as defined under Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupies a separate character position (see the SIGN clause), that character will not be moved and the size of the sending item will be considered to be one less than its actual size (in terms of standard data format characters).

2. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules except where zeroes are replaced because of editing requirements.

   When a signed item is the receiving item, the sign of the sending item is placed in the receiving item. (See the SIGN clause). Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

   When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

   When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

3. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the OCCURS clause.

When a sending and receiving item share a part of their storage areas, the result of the execution of such a statement is undefined.

The CORRESPONDING Phrase

When the CORRESPONDING phrase is specified, data items in identifier-1 are moved to corresponding data items in identifier-2 according to the following rules:

A data item in identifier-1 and a data item in identifier-2 are not designated by the key word FILLER and have the same qualifiers up to, but not including, identifier-1 and identifier-2.

At least one of the data items is an elementary data item.

The description of identifier-1 and identifier-2 must not contain level-number 66, 77, or 88 or the USAGE IS INDEX clause.

A data item that is subordinate to identifier-1 or identifier-2 and contains a REDEFINES, RENAMES, OCCURS or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, identifier-1 and identifier-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

Data in the following chart summarizes the legality of the various types of MOVE statements.

CATEGORY OF SENDING DATA ITEM		CATEGORY OF RECEIVING DATA ITEM		
		ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	NUMERIC INTEGER NUMERIC NON-INTEGER NUMERIC EDITED
ALPHABETIC		YES	YES	NO
ALPHANUMERIC		YES	YES	YES
ALPHANUMERIC EDITED		YES	YES	NO
NUMERIC	INTEGER	NO	YES	YES
	NON-INTEGER	NO	NO	YES
NUMERIC EDITED		NO	YES	NO

MOVE Examples

```
MOVE INCOME TO TOTAL-INCOME.

MOVE 1 TO PAGE-COUNT, LINE-NUM

MOVE "MARMACK INDUSTRIES" TO TITLE-HEADER.

MOVE PERSON IN FILE-RECORD TO
     PERSON OF ALABAMA (I-A OF ALABAMA),
     PERSON OF CROSS-CENSUS.

MOVE NUM TO NUM-ED

MOVE TABLE-ELT (N, 1, M) TO NEXT-ENTRY
     PREVIOUS-ENTRY

MOVE -36.7 TO DEFICIT.

MOVE QUOTES TO SECTION-DIVIDER.

MOVE ZERO TO COUN-TER

MOVE ZEROES TO COUN-TER.
```

The MULTIPLY Statement

The MULTIPLY statement causes numeric data items to be multiplied and stores the result.

FORMAT 1

```
   MULTIPLY {identifier-1}
   --------
            {literal-1    }

        BY identifier-2 [ROUNDED]
        --              -------

        [;ON SIZE ERROR imperative-statement]
             ---- -----
```

FORMAT 2

```
   MULTIPLY {identifier-1} BY {identifier-2}
   --------                --
            {literal-1    }   {literal-2    }

        GIVING identifier-3 [ROUNDED]
        ------               -------

        [;ON SIZE ERROR imperative-statement]
             ---- -----
```

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product.

In Format 2, the value of identifier-1 or literal-1 is multiplie d by identifier-2 or literal-2 and the result is stored in identifier-3.

Each identifier must refer to a numeric elementary item, except that in Format 2 the identifier following the word GIVING GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

Each literal must be a numeric literal.

The ROUNDED Phrase

The MULTIPLY statement may optionally include the ROUNDED phrase.

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant-identifier are represented by the character 'P' in the picture for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

The SIZE ERROR Phrase

If, after appropriate decimal point alignment, the absolute value of the result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the resultant-identifier has COMPUTATIONAL-3 usage, size error is detected only for data items declared with an odd length picture clause. Therefore all COMP-3 data items should be declared with an odd number of character positions.

If the SIZE ERROR phrase is not specified and a size error condition exists, the value of the resultant-identifier is undefined.

If the SIZE ERROR phrase is specified and a size error condition exists, the value of the resultant-identifier is not altered and the imperative statement is the SIZE ERROR phrase is executed.

MULTIPLY Examples

```
    MULTIPLY 10 BY INCOME.

    MULTIPLY PRINCIPAL BY INTEREST-RATE
         GIVING INTEREST ROUNDED.

    MULTIPLY INFLATION-RATE BY EXPENSES
         ON SIZE ERROR MOVE 0 TO ECONOMY-RATING.
```

The OPEN Statement (Sequential I-O)

The OPEN statement initiates the processing of sequential files.

FORMAT

```
    OPEN {{INPUT {file-name-1 [WITH NO REWIND] }...  }...
    ----   -----                    -- ------

            {OUTPUT {file-name-2 [WITH NO REWIND] }...  } ...
             ------                   -- ------

            {I-O {file-name-3 }...                      } ...
             ---

            {EXTEND {file-name-4 }...                   } ...}...
             ------
```

The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

The successful execution of an OPEN statement makes the associated record area available to the program.

The files referenced in the OPEN statement need not all have the same organization or access.

Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly.

An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In the Permissible Statements Table below, 'X' at an intersection indicates that the specified statement, used in the sequential access mode, may be used with the sequential file organization and open mode given at the top of the column.

Statement	Open Mode			
	Input	Output	Input-Output	Extend
READ	X		X	
WRITE		X		X
REWRITE			X	

Permissible Statements Table

A file may be opened with the INPUT, OUTPUT, EXTEND, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

Execution of the OPEN statement does not obtain or release the first data record.

The file description entry for file-name-1, file-name-3 or file-name-4 must be equivalent to that used when this file was created.

The execution of an OPEN statement causes the value of the specified FILE STATUS data item, if any, associated with file-name-1 ... to be updated.

The INPUT Phrase

For files being opened with the INPUT phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition.

The OUTPUT Phrase

Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

The EXTEND Phrase

When the EXTEND phrase is specified, the OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file has been opened with the OUTPUT phrase.

The EXTEND phrase and NO REWIND phrase can be used only for sequential files. The EXTEND phrase must not be specified for a file whose device-type is INPUT.

When the EXTEND phrase is specified and the LABEL RECORDS clause indicates label records are present, the execution of the OPEN statement includes the following:

> The beginning file labels are processed only in the case of a single reel/unit file.
>
> Processing then proceeds as though the file has been opened with the OUTPUT phrase.

The I-O Phrase

The I-O phrase permits the opening of a mass storage file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the mass storage file is being initially created.

The I-O phrase can be used only for mass storage files (files assigned to the RANDOM device-type).

When the I-O phrase is specified and the LABEL RECORDS clause indicates that label records are present, the execution of the OPEN includes the following:

> The labels are checked.
>
> New labels are written.

The OPEN statement sets the current record pointer to the first record currently existing in the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for that file will result in an AT END condition.

The NO REWIND Phrase

The NO REWIND phrases can only be used with sequential single reel/unit files. Both phrases will be ignored if they do not apply to the storage media on which the file resides.

If the storage medium for the file permits rewinding, the following rule applies:

> When neither the EXTEND nor the NO REWIND phrase is specified, execution of the OPEN statement causes the file to be positioned at its beginning.
>
> When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned; the file must be already positioned at its beginning prior to the execution of the OPEN statement.

The OPEN statement initiates the processing of mass storage files.

FORMAT

```
   OPEN {{INPUT  {file-name-1 }...}...
   ----   -----
         {OUTPUT {file-name-2 }...}...   Destroys any Previous File
          ------
         {I-O    {file-name-3 }...}...}... File must already Exist
          ---
```

The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

The successful execution of the OPEN statement makes the associated record area available to the program.

The files referenced in the OPEN statement need not all have the same organization or access.

Prior to the successful execution of an OPEN statement for a given file, no statement can be executed that references that file, either explicitly or implicitly.

A file may be opened with the INPUT, OUTPUT, and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

Execution of the OPEN statement does not obtain or release the first data record.

If label records are specified for the file, the beginning labels are processed as follows:

When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with the System conventions for input label checking.

When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with the System conventions for output label writing.

The behavior of the OPEN statement when label records are specified but not present, or when label records are not specified but are present, is undefined.

The file description entry for file-name-1 or file-name-3 must be equivalent to that used when this file was created.

The execution of the OPEN statement causes the value of the specified FILE STATUS data item, if any, associated with file-name-1 ... to be updated.

An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. In the Permissible Statements Table below, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, may be used with the open mode given at the top of the column.

File Access Mode	Statement	Open Mode		
		Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

Permissible Statements Table

The INPUT Phrase

For files being opened with the INPUT phrase, the OPEN statement sets the current record pointer to the first record currently existin within the file. If no records exist in the file, the current record pointer is set such that the next executed Format 1 READ statement for the file will result in an AT END condition.

The OUTPUT Phrase

Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

The I-O Phrase

For files being opened with the I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed Format 1 READ statement for the file will result in an AT END condition.

The PERFORM Statement

The PERFORM statement is used to transfer control explicitly to one or more procedures and to return control implicitly whenever execution of the specified procedure is complete.

FORMAT 1

```
   PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
   -------                    -------
                            {THRU    }
```

FORMAT 2

```
   PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
   -------                    -------
                            {THRU    }
                             ----

      {identifier-1} TIMES
                     -----
      {integer     }
```

FORMAT 3

```
   PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
   -------                    -------
                            {THRU    }
                             ----

     UNTIL condition-1
     -----
```

FORMAT 4

```
   PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
   -------                   -------
                             {THRU    }
                              ----

      VARYING {identifier-2} FROM {identifier-3}
      -------                ----
              {index-name-1}      {index-name-2}
                                  {literal-1    }

        BY {identifier-4} UNTIL condition-1
        --                -----
           {literal-2    }

      [AFTER {identifier-5} FROM {identifier-6}
       -----                ----
             {index-name-3}      {index-name-4}
                                 {literal-3    }

        BY {identifier-7} UNTIL condition-2
        --                -----
           {literal-4    }

      [AFTER {identifier-8} FROM {identifier-9}
       -----                ----
             {index-name-5}      {index-name-6}
                                 {literal-5    }

        BY {identifier-10} UNTIL condition-3]]
        --                 -----
           {literal-6     }
```

Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once and then control passes to the next executable statement following the PERFORM statement.

Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer or by the initial value of the data item referenced by identifier-1 for that execution. If, at the time of execution of a PERFORM statement, the value of the data item referenced by identifier-1 is equal to zero or is negative, control passes to the next executable statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the next executable statement following the PERFORM statement.

During execution of the PERFORM statement, references to identifier-1 cannot alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

Format 3 is the PERFORM...UNTIL. The specified procedures are performed until the condition specified by the UNTIL phrase is true. When the condition is true, control is transferred to the next executable statement after the PERFORM statement. If the condition is true when the PERFORM statement is entered, no transfer to procedure-name-1 takes place, and control is passed to the next executable statement following the PERFORM statement.

Format 4 is the PERFORM...VARYING. This variation of the PERFORM statement is used to augment the values referenced by one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. In the following discussion, every reference to identifier as the object of the VARYING, AFTER and FROM (current value) phrases also refers to index-names. When index-name appears in a VARYING and/or AFTER phrase, it is initialized and subsequently augmented (as described below) according to the rules of the SET statement. When index-name appears in the FROM phrase, identifier, when it appears in an associated VARYING or AFTER phrase, is initialized according to the rules of the SET statement; subsequent augmentation is as described below.

In Format 4, when one identifier is varied, identifier-2 is set to the value of literal-1 or the current value of identifier-3 at the point of initial execution of the PERFORM statement; then, if the condition of the UNTIL phrase is false, the sequence of procedures, procedure-name-1 through procedure-name-2, is executed once. The value of identifier-2 is augmented by the specified increment or decrement value (the value of identifier-4 or literal-2) and condition-1 is evaluated again. The cycle continues until this condition is true; at which point, control is transferred to the next executable statement following the PERFORM statement. If condition-1 is true at the beginning of execution of the PERFORM statement, control is transferred to the next executable statement following the PERFORM statement.

Each identifier represents a numeric elementary item described in the Data Division. In Format 2, identifier-1 must be described as a numeric integer.

Each literal represents a numeric literal.

The words THRU and THROUGH are equivalent.

If an index-name is specified in the VARYING or AFTER phrase, then:

> The identifier in the associated FROM and BY phrases must be an integer data item.
>
> The literal in the associated FROM phrase must be a positive integer.
>
> The literal in the associated BY phrase must be a non zero integer.

If an index-name is specified in the FROM phrase, then:

> The identifier in the associated VARYING or AFTER phrase must be an integer data item.
>
> The identifier in the associated BY phrase must be an integer data item.
>
> The literal in the associated BY phrase must be an integer.

Literal in the BY phrase must not be zero.

Condition-1, condition-2, condition-3 may be any conditional expression.

When procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the declarative section of the program then both must be procedure-names in the same declarative section.

The data items referenced by identifier-4, identifier-7, and identifier-10 must not have a zero value.

If an index-name is specified in the VARYING or AFTER phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. This transfer of control occurs only once for each execution of a PERFORM statement. For those cases when a transfer of control to the named procedure does take place, an implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.

If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

If procedure-name-2 is specified and it is a section-name, then the return is after the last statement of the last paragraph in the section.

There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. If there are two or more logical paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

If control passes to these procedures by means other than a PERFORM statement, control will pass through the last statement of the procedure to the next executable statement as if no PERFORM statement mentioned these procedures.

```
                                  ENTRANCE
                                     ¦
                                     V
                        ---------------------------
                        ¦Set identifier-2 equal to¦
                        ¦    current FROM value   ¦
                        ---------------------------
                                     ¦
                                     V
                                /-----------\      True
 ----------------------> ¦ Condition-1 ¦-------------> Exit
 ¦                              \-----------/
 ¦                                   ¦
 ¦                                   V  False
 ¦                      ---------------------------
 ¦                      ¦ Execute procedure-name-1¦
 ¦                      ¦   THRU procedure-name-2 ¦
 ¦                      ---------------------------
 ¦                                   ¦
 ¦                                   V
 ¦                      ---------------------------
 ¦----------------------¦Augment identifier-2 with¦
                        ¦    current BY value     ¦
                        ---------------------------
```

Flowchart for the VARYING Phrase of a PERFORM Statement Having One Condition.

In Format 4, when two identifiers are varied, identifier-2 and identifier-5 are set to the current value of identifier-3 and identifier-6, respectively.

After the identifiers have been set, condition-1 is evaluated; if true, control is transferred to the next executable statement; if false, condition-2 is evaluated. If condition-2 is false, procedure-name-1 through procedure-name-2 is executed once, then identifer-5 is augmented by identifier-7 or literal-4 and condition-2 is evaluated again. This cycle of evaluation and augmentation continues until this condition is true. When condition-2 is true, identifier-5 is set to the value of literal-3 or the current value of identifier-6, identifier-2 is augmented by identifier-4 and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING variable (identifier-2 and index-name-1), the BY variable (identifier-4), the AFTER variable (identifier-5 and index-name-3), or the FROM variable (identifier-3 and index-name-2) will be taken into consideration and will affect the operation of the PERFORM statement.

```
                 ENTRANCE
                    |
                    V
         ------------------------------------
         |Set identifier-2 and identifier-5|
         |      to current FROM values      |
         ------------------------------------
                    |
                    V
               /-------------\       True
------------->|  Condition-1  |-----------------> Exit
|              \-------------/
|                   |
|                   V False
|              /-------------\       True
|  ---------->|  Condition-2  |----------------
|  |           \-------------/                |
|  |                |                         |
|  |                V False                   V
|  |  --------------------------       -------------------------
|  |  |Execute procedure-name-1|       |Set identifier-5 to its|
|  |  |THRU procedure-name-2   |       | current FROM value    |
|  |  --------------------------       -------------------------
|  |                |                         |
|  |                V                         V
|  |  ----------------------------     ---------------------------
|  --|Augment identifier-5 with|       |Augment identifier-2 with|
|   |   current BY value       |       |   current BY value      |
|   ----------------------------       ---------------------------
|                                             |
-----------------------------------------------
```

Flowchart for the VARYING Phrase of a PERFORM Statement Having Two Conditions.

```
                ENTRANCE
                   ¦
                   V
         ----------------------------
         ¦        Set               ¦
         ¦identifier-2,identifier-5,¦
         ¦     identifier-8         ¦
         ¦  to current FROM values  ¦
         ----------------------------
                   ¦
                   V
               /----------\       True
------------->¦ Conditon-1 ¦----------------> Exit
¦              \----------/
¦                  ¦ False
¦                  V
¦             /-----------\       True
¦  ---------->¦ Condition-2 ¦-----------------------------
¦ ¦           \-----------/                              ¦
¦ ¦                ¦ False                               ¦
¦ ¦                V                                     ¦
¦ ¦           /-----------\       True                   ¦
¦ ¦  -------->¦ Condition-3 ¦----------                  ¦
¦ ¦ ¦         \-----------/          ¦                   ¦
¦ ¦ ¦              ¦ False           ¦                   ¦
¦ ¦ ¦              V                 V                   V
¦ ¦ ¦    ------------------    ----------------    ----------------
¦ ¦ ¦    ¦    Execute     ¦    ¦     Set      ¦    ¦     Set      ¦
¦ ¦ ¦    ¦procedure-name-1¦    ¦identifier-8  ¦    ¦identifier-5  ¦
¦ ¦ ¦    ¦THRU procedure- ¦    ¦to its current¦    ¦to its current¦
¦ ¦ ¦    ¦   name-2       ¦    ¦  FROM value  ¦    ¦  FROM value  ¦
¦ ¦ ¦    ------------------    ----------------    ----------------
¦ ¦ ¦             ¦                   ¦                   ¦
¦ ¦ ¦             V                   V                   V
¦ ¦ ¦   -------------------- -------------------- --------------------
¦ ¦ ¦   ¦     Augment      ¦ ¦     Augment      ¦ ¦     Augment      ¦
¦ ¦ --- ¦identifier-8 with ¦ ¦identifier-5 with ¦ ¦ identifier-2 with¦
¦ ¦     ¦ current BY value ¦ ¦ current BY value ¦ ¦ current BY value ¦
¦ ¦     -------------------- -------------------- --------------------
¦ ¦                                  ¦                   ¦
¦  ----------------------------------                    ¦
----------------------------------------------------------
```

Flowchart for the VARYING Phrase of a PERFORM Statement Having Three Conditions.

At the termination of the PERFORM statement identifier-5 contains the current value of identifier-6. Identifier-2 has a value that exceeds the last setting by an increment or decrement value, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-2 contains the current value of identifier-3.

When two identifiers are varied, identifier-5 goes through a complete cycle (FROM, BY, UNTIL) each time identifier-2 is varied.

For three identifiers the mechanism is the same as for two identifiers except that identifier-8 goes through a complete cycle each time that identifier-5 is augmented by identifier-7 or literal-4, which in turn goes through a complete cycle each time identifier-2 is varied.

After the completion of a Format 4 PERFORM statement, identifier-5 and identifier-8 contain the current value of identifier-6 and identifier-9 respectively. Identifier-2 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-2 contains the current value of identifier-3.

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from, the logical sequence referred to by the first PERFORM. Thus an active PERFORM statement, whose execution point begins within the range of another active PERFORM statement, must not allow control to pass to the exit of the other active PERFORM statement; furthermore, two or more such active PERFORM statements may not have a common exit. See the valid illustrations below.

```
x PERFORM a THRU m

a ------------------------------
                               ¦
d PERFORM f THRU j             ¦
                               ¦
h                              ¦
                               ¦
m ------------------------------

f ----------
           ¦
j ----------
```

```
x   PERFORM a THRU m

a  ------------------------------
                                |
d   PERFORM f THRU j            |
                                |
f  -----------                  |
             |                  |
j  -----------                  |
                                |
m  ------------------------------
```

```
x   PERFORM a THRU m

a    -----------------------------
                                 |
f    ---------                   |
             |                   |
m    --------|--------------------
             |
j   ----------

d PERFORM f THRU j
```

A PERFORM statement that appears in a section that is not in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

Sections and/or paragraphs wholly contained in one or more non-independent segments.

Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have within its range, in addition to any declarative sections whose execution is caused within that range, only one of the following:

Sections and/or paragraphs wholly contained in one or more non-independent segments.

Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement.

The READ Statement (Sequential I/O)

The READ statement makes available the next logical record from a file.

FORMAT

```
   READ file-name RECORD  [INTO identifier]
   ----                    ----

          [;AT END imperative-statement]
               ---
```

The associated file must be open in the INPUT or I-O mode at the time this statement is executed.

The record to be made available by the READ statement is determined as follows:

> If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.
>
> If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

When the logical records of a file are described with more than one record description the contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

If, at the time of execution of a READ statement, the position of the current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

The INTO Phrase

If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by thier record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

The AT END Phrase

If, at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

When the AT END condition is recognized the following actions are taken in the specified order.

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

If the AT END phrase is specified in the statement causing the conditio n, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file and that procedure is executed.

When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

The AT END phrase must be specified if no applicable USE procedure is specified for file-name.

The READ Statement (Relative and Indexed I-O)

The READ statement makes available a specified record from a mass storage file.

FORMAT 1

```
   READ file-name [NEXT] RECORD [WITH NO LOCK] [INTO identifier]
   ----           ----              --  ----

      [;AT END imperative-statement]
           ---
```

FORMAT 2

```
   READ file-name RECORD [WITH NO LOCK] [INTO identifier]
   ----                        --  ----

      [;KEY IS data-name]
        ---

      [;INVALID KEY imperative-statement]
        -------
```

Format 1 must be used for all files in sequential access mode.

The NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

Format 2 is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

The associated files must be open in the INPUT or I-O mode at the time this statement is executed.

The KEY phrase may be specified only when the organization of file-name is index. When the KEY clause is present, data-name must be the name of one of the record keys associated with file-name. Data-name may be qualified.

The record to be made available by a Format 1 READ statement is determined as follows:

> The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer. If the record is no longer accessible, which may have been caused by the deletion of the record, the current record pointer is updated to point to the next existing record in the file and that record is then made available.
>
> If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

When the logical records of a file are described with more than one record description, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items which lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

If, at the time of execution of a Format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

The INTO Phrase

If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

The INTO phrase must not be usd when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

For relative files if the RELATIVE KEY phrase is specified, the execution of a Format 1 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

For relative files the execution of a Format 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

For an indexed file if the KEY phrase is specified in a Format 2 READ statement, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

If the KEY phrase is not specified in a Format 2 READ statement, the prime record key is established as the key of reference for this retrieval.

If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established for the file.

For indexed files the execution of a Format 2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. The current record pointer positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

The AT END Phrase

If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

When the AT END condition is recognized, the following actions are taken in the specified order:

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

If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing one of the following:

A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

A successful START statement for that file.

A successful Format2 READ statement for that file.

For a file for which dynamic access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from the file.

The REWRITE Statement (Sequential I/O)

The REWRITE statement logically replaces a record existing in a mass storage file.

FORMAT

```
   REWRITE record-name [FROM identifer]
   -------              ----
```

Record-name and identifier must not refer to the same storage area.

Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

The file associated with record-name must be a mass storage file and must be open in the I-O mode at the time of execution of this statement.

The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement.

The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

The logical record released by successful execution of the REWRITE statement is no longer available in the record area.

The current record pointer is not affected by the execution of a REWRITE statement.

The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

The FROM Phrase

The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

```
MOVE identifier TO record-name
```

followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

The REWRITE Statement (Relative and Indexed I-O)

The REWRITE statement logically replaces a record existing in a mass storage file.

FORMAT

```
   REWRITE record-name [FROM identifier]
   -------              ----

       [; INVALID KEY imperative-statement]
          -------
```

Record-name and identifier must not refer to the same storage area.

Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

For relative files the INVALID KEY phrase must not be specified for a REWRITE statement which references a file in sequential access mode.

The INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified.

For indexed files the INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

The file associated with record-name must be open in the I-O mode at the time of execution of this statement.

For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement without the WITH NO LOCK phrase.

The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

The logical record released by a successful execution of the REWRITE statement is no longer available in the record area.

The current record pointer is not affected by the execution of a REWRITE statement.

The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

The INVALID KEY Phrase

For a relative file accessed in either random or dynamic access mode, the System logically replaces the record specified by the contents of the key data item associated with the file. If the file does not contain the record specified by the key, the INVALID KEY condition exists.

For indexed files the INVALID KEY condition exists when:

> The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record read from the field, or
>
> The value contained in the prime record key item does not equal that of any record stored in the file.

When the INVALID KEY condition exists the updating operation does not take place and the data in the record area is unaffected.

The FROM Phrase

The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

```
      MOVE identifier TO record-name
```

followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

The SET Statement

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

FORMAT 1

```
    SET {identifier-1} [,identifier-2] ...} TO {identifier-3}
    ---                                     -- {index-name-3}
        {index-name-1} [,index-name-2]         {integer-1   }
```

FORMAT 2

```
    SET index-name-4 [,index-name-5] ...  {UP BY  } {identifier-4}
    ---                                    -- --
                                          {DOWN BY} {integer-2   }
                                           ---- --
```

All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

Identifier-1 and identifier-3 must name either index data items, or elementary items described as an integer.

Identifier-4 must be declared as an elementary numeric integer.

Integer-1 and integer-2 may be signed. Integer-1 must be positive.

Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the associated table.

If index-name-4, index-name-5 is specified, the value of the index both before and after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. The value of the index associated with an index-name after the execution of a PERFORM statement may be undefined.

In Format 1, the following action occurs:

Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index data item; no conversion takes place in either case.

If identifier-1 is not an index data item, it may be set only 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.

The process is repeated for index-name-2, identifier-2, etc., if specified. Each time the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

In Format 2, 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; thereafter, the process is repeated for index-name-5, etc. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

Data in the following chart represents the validity of various operand combinations in the SET statement.

Sending Item	Receiving Item		
	Integer Data Item	Index Name	Index Data Item
Integer Literal	No	Valid	No
Integer Data Item	No	Valid	No
Index-Name	Valid	Valid	Valid*
Index Data Item	No	Valid*	Valid*

*No conversion takes place

The START Statement (Relative and Indexed I-O)

The START statement provides a basis for logical positioning within a file, for subsequent sequential retrieval of records.

FORMAT

```
   START file-name [KEY {IS EQUAL TO      } data-name]
   -----            ---     -----
                       {IS =              }
                       {IS GREATER THAN   }
                           -------
                       {IS >              }
                       {IS NOT LESS THAN  }
                           --- ----
                       {IS NOT <          }
                           ---

        [; INVALID KEY imperative-statement]
           -------
```

Note: The required relational characters '>', '<' and '=' are not underlined to avoid confusion with other symbols.

File-name must be the name of a file with sequential or dynamic access.

Data-name may be qualified.

The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

If file-name is the name of a relative file then data-name, if specified, must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

If file-name is the name of an indexed file then data-name, if specified, may reference the data items specified as the record keys associated with file-name or it may reference any data item of category alphanumeric whose leftmost character position corresponds to the leftmost character position of a record key data item.

File-name must be open in the INPUT or I-O mode at the time that the START statement is executed.

If the KEY phrase is not specified the relational operator 'IS EQUAL TO' is implied.

The type of comparison specified by the relational operator in the KEY phrase occurs between a key associated with a record in the file referenced by file-name and a data item.

If file-name references a relative file, the data item used in the comparison is the relative key associated with file-name.

If file-name references an indexed file, the data item used in the comparison is either the prime record key associated with file-name or, if the KEY phrase is specified, the data item referenced in the KEY phrase. If the operands of the comparison are of unequal size, comparison proceeds as though the longer one were truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply except that the presence of the PROGRAM COLLATING SEQUENCE clause will have no effect on the comparison.

The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.

The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

The STOP Statement

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

FORMAT

```
   STOP  {RUN    }
   ----   ---
         {literal}
```

The literal may be numeric or nonnumeric or may be any figurative constant.

If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

If the RUN phrase is used, then a STOP RUN message is logged and the execution is terminated.

If STOP literal is specified, the literal is logged in a STOP "literal-value" message and the execution is suspended.

STOP Examples:

```
     STOP RUN.
     STOP "END OF PROCEDURE".
```

The SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from a numeric data item and store the result.

FORMAT 1

```
    SUBTRACT {identifier-1} [,identifier-2] ...
    --------
             {literal-1    } [,literal-2    ]

       FROM identifier-m [ROUNDED]
       ----               -------

       [;ON SIZE ERROR imperative-statement]
            ---- -----
```

FORMAT 2

```
    SUBTRACT {identifier-1} [,identifier-2] ...
    --------
             {literal-1    } [,literal-2    ]

       FROM {identifier-m}  GIVING identifier-n [ROUNDED]
       ----                 ------               -------
            {literal-m    }

       [;ON SIZE ERROR imperative-statement]
            ---- -----
```

FORMAT 3

```
    SUBTRACT {CORRESPONDING} identifier-1
    --------  -------------
             {CORR          }
              ----
       FROM identifier-2 [ROUNDED]
       ----               -------

       [; ON SIZE ERROR imperative-statement]
             ---- -----
```

In Format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from the current value of identifier-m storing the result immediately into identifier-m.

In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n.

If Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

Each identifier must refer to a numeric elementary item except that:

> In Format 2, the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.
>
> In Format 3, the identifiers must refer to group items.

Each literal must be a numeric literal.

The ROUNDED Phrase

The SUBTRACT statement may optionally include the ROUNDED phrase.

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant-identifier are represented by the character 'P' in the picture for that resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

The SIZE ERROR Phrase

If, after appropriate decimal point alignment, the absolute value of the result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. If the ROUNDED phrase is specified, rounding takes place before checking for size error.

If the resultant-identifier has COMPUTATION-3 usage, size error is detected only for data items declared with an odd length picture clause. Therefore, all COMP-3 data items should be declared with an odd number of character positions. exceeds the largest value that can be contained in the resultant-identifier, a size error condition exists.

If the SIZE ERROR phrase is not specified and a size error condition exists, the value of the resultant-identifier is undefined.

If the SIZE ERROR phrase is specified and a size error condition exists, the value of the resultant-identifier(s) affected by the size error is not altered.

If the CORRESPONDING phrase is specified, and any of the individual subtractions produce a size error condition, the imperative-statement is not executed until all of the individual subtractions are completed.

The CORRESPONDING Phrase

If the CORRESPONDING phrase is used, selected items within identifier-1 are ADDed to, and the result stored in, the corresponding items in identifier-2. Data items referenced by the CORRESPONDING phrase must adhere to the following rules:

> A data item in identifier-1 and a data item in identifier-2 must not be designated by the key word FILLER and must not have the same data-name and the same qualifiers up to, but not including, identifier-1 and identifier-2.
>
> Both of the data items must be elementary numeric data items.
>
> The description of identifier-1 and identifier-2 must not contain level-numbers 66, 77 or 88 or the USAGE IS INDEX clause.
>
> A data item that is subordinate to identifier-1 or identifier-2 and contains a REDEFINES, RENAMES, OCCURS or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, OCCURS, or USAGE IS INDEX clause. However, identifier-1 and identifier-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

CORR is an abbreviation for CORRESPONDING.

SUBTRACT EXAMPLES

```
SUBTRACT TAXES FROM INCOME.

SUBTRACT 1 FROM TALLY GIVING TALLY-1.

SUBTRACT 2.68, INTEREST, PENALTY
     FROM PRINCIPAL ROUNDED
     ON SIZE ERROR GO TO ERROR-HANDLER.
```

The UNLOCK Statement

The UNLOCK statement makes available to other programs the most recently accessed record in a file that was read and locked.

FORMAT

```
    UNLOCK file-name RECORD.
    ------
```

Note: The UNLOCK statement is nonstandard, but provides for compatibility with existing programs written for environments that allow multiple programs to concurrently update a data file. For systems that do not provide this capability, the UNLOCK statement will not affect execution except as described below.

The file associated with the file-name must be open in the I-O mode.

If no record in the file is locked, execution of an UNLOCK statement causes no action to be taken. If a record in the file is locked (unavailable to other programs), the last record to be locked is then made available to any other program upon execution of the UNLOCK statement.

The current record pointer is not affected by the execution of the UNLOCK statement. The FILE STATUS data item associated with the file, if one exists, is updated.

The UNLOCK statement may not be used to unlock records locked by other programs.

Note: Records that are read and locked are automatically unlocked by any subsequent operation on that file from the same program.

The WRITE Statement (Sequential I/O)

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines within a logical page.

FORMAT

```
   WRITE record-name [FROM identifier-1]
   -----             ----

     [{BEFORE} ADVANCING {{identifier-2} [LINE ]}]
       ------
      {AFTER }           {{integer      } [LINES]}
       -----
                         {     PAGE              }
                               ----
```

Record-name and identifier-1 must not reference the same storage area.

The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

Integer or the value of the data item referenced by identifier-2 may be zero.

The associated file must be open in the OUTPUT or EXTEND mode at the time of the execution of this statement.

The logical record released by the execution of the WRITE statement is no longer available in the record area.

Upon completion of a WRITE statement, the information in the area referenced by identifier-1 is available even though the information in the area referenced by record-name may not be available.

The current record pointer is unaffected by the execution of a WRITE statement.

The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

The execution of the WRITE statement releases a logical record to the operating system. The contents of the record area are not changed.

When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists. The following action takes place:

> The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation.
>
> If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicit ly specified for the file, that declarative procedure will then be executed.
>
> If a USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

The FROM Phrase

The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of the statement

```
     MOVE identifier-1 TO record-name
```

according to the rules specified for the MOVE statement, followed by the same WRITE statement without the FROM phrase.

The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

The ADVANCING Phrase

The ADVANCING phrase allows control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided by the compiler to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

> If identifier-2 is specified, the representation of the printed page is advanced the number of lines equal to the current value associated with identifier-2.
>
> If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.
>
> If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced.
>
> If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced.
>
> If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page.

The ADVANCING phrase is valid only if the device-type assigned to the file is PRINT.

THE WRITE STATEMENT (Relative and Indexed I-O)

The WRITE statement releases a logical record for an output or input-output file.

FORMAT

```
    WRITE record-name [FROM identifier]
    -----              ----

          [; INVALID KEY imperative-statement]
             -------
```

Record-name and identifier must not reference the same storage area.

The record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.

The logical record released by the execution of the WRITE statement is no longer available in the record area.

The current record pointer is unaffected by the execution of a WRITE statement.

The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

The execution of the WRITE statement releases a logical record to the operating system.

When a relative file is opened in the output mode, records may be placed into the file by one of the following:

> If the access mode is sequential, the WRITE statement will cause a reco rd to be released to the System. The first record will have a relative record number of one (1) and subsequent records released will have relative record numbers of 2, 3, 4, ... If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the System during execution of the WRITE statement.
>
> If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to the System by execution of the WRITE statement.

When a relative file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record . in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the System.

For an indexed file, the data item specified as the prime record key must set by the program to the desired value prior to the execution of the WRITE statement. Records may be placed into the file by one of the following:

> If the access mode is sequential, records must be released to the Syste m in ascending order of prime record key values.
>
> If the access mode is random or dynamic, records may be released to the System in any program-specified order.

The FROM Phrase

The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of the statement:

```
        MOVE identifier-1 TO record-name
```

according to the rules specified for the MOVE statement, followed by the same WRITE statement without the FROM phrase.

The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

The INVALID KEY Phrase

The INVALID KEY condition exists under the following circumstances:

When the access mode is sequential for an indexed file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or

When an indexed file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, or

When a relative file has random or dynamic access mode and the RELATIVE KEY data item specifies a record which already exists in the file, or

When an attempt is made to write beyond the externally defined boundaries of the file.

When the INVALID KEY condition is recognized the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, associated with file-name of the associated file is set to a value indicating the cause of the condition.

APPENDIX A

ERROR MESSAGES

ERROR MESSAGES (Compile Time)

The text of the source program is checked for syntax and semantic errors as it is scanned. Errors may cause interruption in scanning. In this case, text is ignored until a recovery point is found and a resume message is printed. Recovery points are chosen to minimize the amount of unanalyzed text without producing irrelevant error messages. In any case the constructs at fault are undermarked and error messages listed when the source line is printed. The error message includes either E's or W's indicating error or warning. For example:

```
  004030  02  STOCK  PIC  9(16)PPP COMPUTATIONAL
                                  $
  ***** 1)PICTURE  *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
```

indicates a semantic number size error but

```
  005040  02  PART  PIC X(4BX(5)    SYNC.
                             $        $
  ***** 1)SYNTAX *E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
  ***** 2)SCAN RESUME  *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W
```

indicates a syntax error at the first undermark and a recovery to the second undermark.

The number preceding the error message is the undermark number, counting from left to right. More than one message may refer to the same undermark.

Global errors such as undefined paragraph names and illegal control transfers are listed with the program summary at the end of the source listing.

Compilation always proceeds to the end of the program, regardless of the number of errors found. Object code is produced such that an attempt to execute an erroneous statement will terminate execution with an appropriate error message.

ACCESS CLASH
Nonsequential access given for sequential file.

BLANK WHEN ZERO
BLANK WHEN ZERO clause given for nonnumeric or group item.

CLASS
The referenced identifier is not valid in a class condition.

COPY
COPY statement failed because of permanent error associated with the undermarked file-name.

CORRESPONDING
The CORRESPONDING phrase cannot be used with the referenced identifier.

DATA OVERFLOW
The data area (working-storage and literals) is larger than 65535 bytes in length.

DATA TYPE
Context does not allow data type of the referenced identifier.

DEVICE CLASH
Random characteristics given to nonrandom device.

DEVICE TYPE
OPEN or CLOSE mode inconsistent with device type.

DOUBLE DECLARATION
Multiple declaration of a file or identifier attribute.

DOUBLE DEFINITION
Multiple definition of an identifier.

DUPLICATE
Warning only. Multiple USE procedure declared for same function or file.

FILE DECL ERROR
The referenced file-name is SELECTed and has an invalid or missing file description (FD).

FILE NAME ERROR

The referenced file-name has an invalid external file name declaration.

FILE NAME REQUIRED

File name not given as reference in I/O verb.

FILE RECORD KEY ERROR

The referenced file-name has a RECORD KEY which is incorrectly qualified or is not defined as a data item of the category alphanumeric within a record description entry associated with that file name.

FILE RECORD SIZE ERROR

The referenced file-name has a declared record size which conflicts with the actual data record descriptions or is a relative organization file with variable length records.

FILE RELATIVE KEY ERROR

The referenced file-name has a RELATIVE KEY which is incorrectly qualified, is defined in a record description associated with that file-name, or is not defined as an unsigned integer.

FILE STATUS ERROR

The referenced file-name has a status item which is incorrectly qualified, is not defined in the WORKING-STORAGE SECTION, or is not a two-character alphanumeric item.

FILE TYPE

Access or organization of file conflicts with undermarked statement.

FILLER LEVEL

A nonelementary FILLER item is declared.

GROUP CLASH

USAGE or VALUE clause of group member conflicts with same clause for group.

GROUP VALUE CLASH

Warning only. An item subordinate to a group with the VALUE IS clause is described with the SYNCHRONIZED, JUSTIFIED, or USAGE (other than USAGE IS DISPLAY) clause.

IDENTIFIER

Identifier reference is incorrectly constructed or the identifier has an invalid or double definition.

ILLEGAL ALTER
: An ALTER statement references an unalterable paragraph or violates the rules of segmentation.

ILLEGAL PERFORM
: A PERFORM statement references undefined or incorrectly qualified paragraph or the reference violates the rules of segmentation.

INVALID ID
: The referenced identifier was not successfully defined.

INVALID PARAGRAPH
: Context does not allow section name.

JUSTIFY
: JUSTIFY clause given in conflict with other attributes.

KEY REQUIRED
: Relative key not declared for random access relative file or record key not declared for indexed file.

LABEL
: Presence or absence of label record conflicts with device standards.

LEVEL
: Level-number given is invalid either intrinsically or because of position within a group.

LINKAGE
: An identifier in the USING clause of the PROCEDURE title is not a linkage item or a statement references a linkage item not subordinate to an identifier in the USING clause of the PROCEDURE title.

LITERAL VALUE
: Literal value given is incorrect in context.

MOVE
: Operands of MOVE verb specify an invalid move.

MUST BE INTEGER
: Context requires decimal integer.

MUST BE PROCEDURE
: Context requires procedure name either as reference or definition, or the reference must be a nondeclarative procedure-name.

MUST BE SECTION
: Context requires procedure-name to be section.

NESTING
Illegal nesting of condition that is not an IF condition.

NOT IN REDEFINE
VALUE IS clause given in REDEFINES item.

OCCURS
Occurs clause given at invalid level or after three have been given for the same item.

OCCURS DEPENDING ERROR
The referenced object of a DEPENDING phrase has not been defined correctly.

OCCURS-VALUE CLASH
VALUE IS and OCCURS in effect for the same item.

PICTURE
Invalid picture syntax.

PICTURE-BWZ CLASH
Zero suppression and BLANK WHEN ZERO cannot be in effect for the same item.

PICTURE-USAGE CLASH
USAGE clause or implied usage conflicts with usage implied by picture.

PROCEDURE INDEPENDENCE
PERFORM given for procedures in independent segments not in the current segment.

PROGRAM OVERFLOW
The instruction area is larger than 32767 bytes in length.

RECORD KEY
Record key declared for other than an indexed organization file or a START statement KEY phrase references a data item not aligned on the declared key's leftmost byte.

RECORD REQUIRED
Context requires record name.

REDEFINES
REDEFINES given within an OCCURS or not redefining the last allocated item.

REDEFINES ERROR
The referenced data-name redefines an item which does not have the same number of character positions and is not level 01.

REFERENCE INVALID
Reference given is not valid in context.

RELATION
Operands of relation test are incompatible.

RELATIVE KEY
Relative key declared for other than a relative organization file or a START statement KEY phrase references a data item other than the declared key.

RESERVED WORD CONFLICT
A COBOL reserved word or symbol is given where a user word is required. In the summary this is only a warning about an ANSI COBOL reserved word that is not an implemented COBOL reserved word.

SCAN RESUME
Warning only. Scanning was terminated at previous error message and resumes at undermarked character.

SECTION CLASH
A VALUE IS clause appears in the FILE or LINKAGE section.

SEGMENT
Warning only. Segment number given in an independent segment is not the same as the current segment or the number of a new independent segment.

SEPARATOR
Warning only. Redundant punctuation or a separator is not followed by the required space.

SIGN
SIGN clause given in conflict with usage and picture.

SIZE
Warning only. Size of data referenced not correct for context.

SIZE ERROR
Declared size of record conflicts with present reference.

SUBSCRIPT	Incorrect number of subscripts or indices for a reference.
SYNC	Synchronized clause given for a group item.
SYNTAX	Incorrect character or reserved word given for context.
UNDEFINED	File referenced in FD entry was not defined.
UNDEFINED DECLARATIVE PROCEDURE	A declarative statement references a procedure not defined within the DECLARATIVES.
UNDEFINED PROCEDURE	A GO TO statement references an undefined or incorrectly qualified paragraph.
USE REQUIRED	A DECLARATIVES section must begin with a USE statement.
USING COUNT	Warning only. The item count in the USING list of a CALL statement is different from that of the first reference to the same program name.
VALUE ERROR	Value given in VALUE IS required truncation of nonzero digits.
VALUE	VALUE IS clause given in conflict with other declared attributes.
VARIABLE RECORD	Warning only. The INTO phrase is not allowed with variable size records.

APPENDIX B

RESERVED WORDS

RESERVED WORD LIST

The following is a list of RM/COBOL reserved words where:

* denotes reserved words not reserved in ANSI standard COBOL

+ denotes ANSI COBOL reserved words not reserved by the compiler. Their appearance will generate a warning at the end of the compilation listing.

** denotes system-name.

ACCEPT	ALPHABETIC	AREA
ACCESS	+ALSO	+AREAS
ADD	ALTER	+ASCENDING
ADVANCING	ALTERNATE	ASSIGN
AFTER	AND	AT
ALL	ARE	AUTHOR
*BEEP	*BLINK	BY
BEFORE	BLOCK	
BLANK	+BOTTOM	
CALL	+CODE-SET	COMPUTE
+CANCEL	COLLATING	CONFIGURATION
+CD	+COLUMN	CONTAINS
+CF	COMMA	+CONTROL
+CH	+COMMUNICATION	+CONTROLS
CHARACTER	COMP	*CONVERT
CHARACTERS	*COMP-1	COPY
+CLOCK-UNITS	*COMP-3	CORR
CLOSE	COMPUTATIONAL	CORRESPONDING
+COBOL	*COMPUTATIONAL-1	+COUNT
+CODE	*COMPUTATIONAL-3	CURRENCY
DATA	+DEBUG-SUB-1	+DESCENDING
DATE	+DEBUG-SUB-2	+DESTINATION
+DATE-COMPILED	+DEBUG-SUB-3	+DETAIL
DATE-WRITTEN	+DEBUGGING	+DISABLE
DAY	DECIMAL-POINT	DISPLAY
+DE	DECLARATIVES	DIVIDE
+DEBUG-CONTENTS	DELETE	DIVISION
+DEBUG-ITEM	+DELIMITED	DOWN
+DEBUG-LINE	+DELIMITER	DUPLICATES
+DEBUG-NAME	DEPENDING	DYNAMIC

*ECHO	+END-OF-PAGE	ERROR
+EGI	+ENTER	+ESI
ELSE	ENVIRONMENT	+EVERY
+EMI	+EOP	EXCEPTION
+ENABLE	EQUAL	EXIT
END	*ERASE	EXTEND

FD	FILLER	+FOOTING
FILE	+FINAL	FOR
FILE-CONTROL	FIRST	FROM

+GENERATE	GO	+GROUP
GIVING	GREATER	

+HEADING	HIGH-VALUE
*HIGH	HIGH-VALUES

I-O	INDEXED	INSPECT
I-O-CONTROL	+INDICATE	INSTALLATION
IDENTIFICATION	INITIAL	INTO
IF	+INITIATE	INVALID
IN	INPUT	IS
INDEX	INPUT-OUTPUT	

JUST	JUSTIFIED

KEY

LABEL	+LIMIT	LINES
+LAST	+LIMITS	LINKAGE
LEADING	+LINAGE	LOCK
LEFT	+LINAGE-COUNTER	LOW
+LENGTH	LINE	LOW-VALUE
LESS	+LINE-COUNTER	LOW-VALUES

MEMORY	MODE	+MULTIPLE
+MERGE	MODULES	MULTIPLY
+MESSAGE	MOVE	

NATIVE	NO	NUMERIC
+NEGATIVE	NOT	
NEXT	+NUMBER	

OBJECT-COMPUTER	OMITTED	OR
OCCURS	ON	ORGANIZATION
OF	OPEN	OUTPUT
OFF	+OPTIONAL	+OVERFLOW
PAGE	+PLUS	+PROCEDURES
+PAGE-COUNTER	+POINTER	PROCEED
PERFORM	POSITION	PROGRAM
+PF	+POSITIVE	PROGRAM-ID
+PH	*PRINT	*PROMPT
PIC	+PRINTING	
PICTURE	PROCEDURE	
+QUEUE	QUOTE	QUOTES
RANDOM	+REMAINDER	*REVERSE
+RD	+REMOVAL	+REVERSED
READ	RENAMES	REWIND
+RECEIVE	REPLACING	REWRITE
RECORD	+REPORT	+RF
RECORDS	+REPORTING	+RH
REDEFINES	+REPORTS	RIGHT
REEL	+RERUN	ROUNDED
+REFERENCES	+RESERVE	RUN
RELATIVE	+RESET	
+RELEASE	+RETURN	
SAME	SIZE	+SUB-QUEUE-2
+SD	+SORT	+SUB-QUEUE-3
+SEARCH	+SORT-MERGE	SUBTRACT
SECTION	+SOURCE	+SUM
SECURITY	SOURCE-COMPUTER	+SUPPRESS
+SEGMENT	SPACE	**SWITCH-1
+SEGMENT-LIMIT	SPACES	**SWITCH-2
SELECT	SPECIAL-NAMES	'
+SEND	STANDARD	'
SENTENCE	STANDARD-1	'
SEPARATE	START	**SWITCH-8
SEQUENCE	STATUS	+SYMBOLIC
SEQUENTIAL	STOP	SYNC
SET	+STRING	SYNCHRONIZED
SIGN	+SUB-QUEUE-1	

*TAB	+TEXT	TO
+TABLE	THAN	+TOP
TALLYING	THROUGH	TRAILING
+TAPE	THRU	+TYPE
+TERMINAL	TIME	
+TERMINATE	TIMES	
UNIT	UNTIL	USAGE
*UNLOCK	UP	USE
+UNSTRING	+UPON	USING
VALUE	VALUES	VARYING
WHEN	WORDS	WRITE
WITH	WORKING-STORAGE	
ZERO	ZEROES	ZEROS
+	>	*
-	<	/
=		**

APPENDIX C

GLOSSARY

GLOSSARY

The terms in this appendix are defined in accordance with their meaning as used in this document describing COBOL and may not have the same meaning for other languages.

These definitions are also intended to be either reference material or introductory material to be reviewed prior to reading the detailed language specifications. For this reason, these definitions are, in most instances, brief and do not include detailed syntactical rules.

Access Mode:
The manner in which records are to be operated upon within a file.

Actual Decimal Point:
The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabet-Name:
A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, that assigns a name to a specific character set and/or collating sequence.

Alphabetic Character:
A character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

Alphanumeric Character:
Any character in the computer's character set.

Alternate Record Key:
A key, other than the prime record key, whose contents identify a record within an indexed file.

Arithmetic Expression:
An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator:
A single character that belongs to the following set:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division

Ascending Key:
A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point:
A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition:
A condition caused during the execution of a READ statement for a sequentially accessed file.

Block:
A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Called Program:
A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program:
A program which executes a CALL to another program.

Character:
The basic indivisible unit of the language.

Character Position:
A character position is the amount of physical storage required to store a single standard data format character described as USAGE is DISPLAY (one byte).

Character-String:
A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

Class Condition:
The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause:
A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Character Set:
The complete COBOL character set consists of the 51 characters listed below.

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

COBOL Word. (See Word)

Collating Sequence:
The sequence in which the characters that are acceptable in a computer are ordered for purposes of 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-Entry:
An entry in the Identification Division that may be any combination of characters from the computer character set.

Comment Line:
A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

Compile-Time:
The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler Directing Statement:
A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition:
A condition in which one or more logical operators act upon one or more conditions.

Computer-Name:
A system-name that identifies the computer upon which the program is to be compiled or run (commentary only).

Condition:
A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of a simple condition, optionally parenthesized, consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name:
A user-defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may possess; or the user-defined word assigned to a status of a system software switch.

Condition-Name Condition:
The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression:
A simple condition or a complex condition specified in an IF or PERFORM statement.

Conditional Statement:
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.

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 that describes overall specifications of source and object computers.

Connective:
A reserved word that is used to:

> Associate a data-name, paragraph-name or condition-name with its qualifier.
>
> Link two or more operands written in a series.
>
> Form conditions (logical connectives).

Contiguous Items:
Items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Counter:
A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Currency Sign:
The character '$' of the COBOL character set.

Currency Symbol:
The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

Current Record:
The record which is available in the record area associated with the file.

Current Record Pointer:
A conceptual entity that is used in the selection of the next record.

Data 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:
An entry in the Data Description that is composed of a level-number followed by a data-name, if required, and then followed by a set of data clauses, as required.

Data Item:
A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name:
A user-defined word that names a data item described in a data description entry in the Data Division. When used in the general formats, 'data-name' represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line:
A debugging line is any line with 'D' in the indicator area of the line.

Declaratives:
A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative-Sentence:
A compiler-directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter:
A character or a sequence of contiguous characters that identify 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.

Digit Position:
A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage of the data item describing the digit position.

Division:
A set of zero, one or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data, and Procedure.

Division Header:
A combination of words followed by a period and a space that indicates the beginning of a division. The division headers are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION [USING data-name-1 [data-name-2]. . . ].
```

Dynamic Access:
An access mode in which specific logical records can be obtained from or placed into a mass storage file in a non sequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

Editing Character:
A single character or fixed two-character combination belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	stroke (virgule, slash)

Elementary Item:
A data item that is described as not being further logically subdivided.

End of Procedure Division:
The physical position in a COBOL source program after which no further procedures appear.

Entry:
Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Environment Clause:
A clause that appears as part of an Environment Division entry.

Execution Time. (See Object Time)

Extend Mode:
The state of a file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Figurative Constant:
A compiler generated value referenced through the use of certain reserved words.

File:
A collection of records.

File Clause:
A clause that appears as part of the file description (FD) entries in the Data Division.

FILE-CONTROL:
The name of an Environment Division paragraph in which the data files for a given source program are declared.

File Description Entry:
An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

File-Name:
A user-defined word that names a file described in a file description entry within the File Section of the Data Division.

File Organization:
The permanent logical file structure established at the time that a file is created.

File Section:
The section of the Data Division that contains file description entries together with their associated record descriptions.

Format:
A specific arrangement of a set of data.

Group Item:
A named contiguous set of elementary or group items.

I-O-CONTROL:
The name of an Environment Division paragraph in which sharing of same areas by several data files is specified.

I-O-Mode:
The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

Identifier:
A data-name, followed as required, by the syntactically correct combination of qualifiers, subscripts, and indices necessary to make unique reference to a data item.

Imperative Statement:
A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Index:
A data item, the contents of which represent the identification of a particular element in a table.

Index Data Item:
A data item in which the value associated with an index-name can be stored.

Index-Name:
A user-defined word that names an index associated with a specific table.

Indexed Data-Name:
An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Indexed File:
A file with indexed organization.

Indexed Organization:
The permanent logical file structure in which each record is identified by the value of one fixed length key within that record.

Input File:
A file that is opened in the input mode.

Input Mode:
The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output File:
A file that is opened in the I-O mode.

Input-Output Section:
The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Integer:
A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero, unless explicitly allowed by the rules of that format.

Invalid Key Condition:
A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

Key:
A data item which identifies the location of a record.

Key Word:
A reserved word whose presence is required when the format in which the word appears is used in a source program.

Level Indicator:
Two alphabetic characters that identify a specific type of file or a position in hierarchy.

Level-Number:
A user-defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level-number is expressed as a one- or two-digit number. Level-numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Level-numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level-numbers 77 and 88 identify special properties of a data description entry.

Library-Name:
A user-defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

Linkage Section:
The section in the Data Division of the called program that describes the data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal:
A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator:
One of the reserved words AND, OR, or NOT. In the formation of a condition, both or neither of AND and OR can be used as logical connectives. NOT can be used for logical negation.

Mass Storage:
A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage File:
A collection of records that is assigned to a mass storage medium.

Mnemonic-Name:
A user-defined word that is associated in the Environment Division witha specified system-name.

Native Character Set:
The character set associated with the COBOL Compiler (ASCII).

Native Collating Sequence:
The collating sequence associated with the native character set.

Negated Combined Condition:
The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition:
The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Sentence:
The next sentence to which control will be transferred after execution of the current statement is complete.

Next Executable Statement:
The next statément to which control will be transferred after execution of the current statement is complete.

Next Record:
The record which logically follows the current record of a file.

Noncontiguous Items:
Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item:
A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal:
A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character:
A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item:
A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '-', or other representation of an operational sign.

Numeric Literal:
A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

OBJECT-COMPUTER:
The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

Object of Entry:
A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

Object Program:
A set or group of executable instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

Object Time:
The time at which an object program is executed.

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.

Occurrence Number:
The relative data item number in a table.

Operand:
Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lowercase word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign:
An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Optional Word:
A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File:
A file that is opened in either the output mode or extend mode.

Output Mode:
The state of a file after execution of an OPEN statement, with the OUTPUT or EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

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. The permissible paragraph headers are:

In the Identification Division:

```
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
SECURITY.
```

In the Environment Division:

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O-CONTROL.
```

Paragraph-Name:
A user-defined word that identifies and begins a paragraph in the Procedure Division.

Phrase:
A phrase is an ordered set of one or more consecutive COBOL character-strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record. (See Block)

Prime Record Key:
A key whose contents uniquely identify a record within an indexed file.

Procedure:
A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name:
A user-defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph-name (which may be qualified), or a section-name.

Program-Name:
A user-defined word that identifies a COBOL source program.

Punctuation Character:
A character that belongs to the following set:

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

Qualified Data-Name:
An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier:
A data-name which is used in a reference together with another data name at a lower level in the same hierarchy. A section-name which is used in a reference together with a paragraph-name specified in that section.

Random Access:
An access mode in which the program-specified value of a key data item identifies the logical record that is obtained from, deleted from, or placed into a relative or indexed file.

Record Area:
A storage area allocated for the purpose of processing the record described in a record description entry in the File Section.

Record Description. (See Record Description Entry)

Record Description Entry:
The total set of data description entries associated with a particular record.

Record Key:
The prime record key whose contents uniquely identify a record within an indexed file.

Record-Name:
A user-defined word that names a record described in a record description entry in the Data Division.

Reference Format:
A format that provides a standard method for describing COBOL source programs.

Relation. (See Relational Operator)

Relation Character:
A character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation Condition:
The proposition, for which a truth value can be determined, that the value of a data item has a specific relationship to the value of another data item. (See Relational Operator)

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 in the construction of a relation condition. The permissible operators and their meanings are:

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

Relative File:
A file with relative organization.

Relative Key:
A key whose contents identifies a logical record in a relative file.

Relative Organization:
The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word:
A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words or system-names.

Run Unit:
A set of one or more object programs which function at object time, as a unit to provide problem solutions.

Section:
A set of zero, one, or more paragraphs or entries, called a section body, the first of which is 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 and Procedure Division.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

In the Data Division:

```
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
```

In the Procedure Division, a section header is composed of a section-name, followed by the reserved word SECTION, followed by a segment-number (optional), followed by a period and a space.

Section-Name:
A user-defined word which names a section in the Procedure Division.

Segment-Number:
A user-defined word which classifies sections in the Procedure Division for purposes of segmentation. Segment-numbers may contain only the characters '0', '1',..., '9'. A segment-number may be expressed either as a one- or two-digit number.

Sentence:
A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator:
A punctuation character used to delimit character-strings.

Sequential Access:
An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor-to-successor logical record sequence determined by the order of records in the file.

Sequential File:
A file with sequential organization.

Sequential Organization:
The permanent logical file structure in which a record is identified by a predecessor-successor relationship established when the record is placed into the file.

Simple Condition:
Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch-status condition
(simple-condition)

SOURCE-COMPUTER:
The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Program:
A syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of the Procedure Division. In contexts where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'source program.'

Special Character:
A character that belongs to the following set:

Character	Meaning
+	plus sign
-	minus sign
*	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

Special-Character Word:
A reserved word which is an arithmetic operator or a relation character.

SPECIAL-NAMES:
The name of an Environment Division paragraph in which switch-names are related to user-defined words.

Standard Data Format:
The concept used in describing the characteristics of data in a COBOL Data Division under the characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

Statement:
A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Subject of Entry:
An operand or reserved word that appears immediately following the level indicator or the level-number in a Data Division entry.

Subprogram. (See Called Program)

Subscript:
An integer whose value identifies a particular element in a table.

Subscripted Data-Name:
An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Switch-Status Condition:
The proposition, for which a truth value can be determined that a switch, capable of being set to an 'on' or 'off' status, has been set to a specific status.

System-Name:
A COBOL word which is used to communicate with the operating environment.

Table:
A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element:
A data item that belongs to the set of repeated items comprising a table.

Text-Name:
A file access name that identifies library text.

Truth Value:
The representation of the result of the evaluation of a condition in terms of one of two values:

true
false

Unary Operator:
A plus (+) 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.

User-Defined Word:
A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable:
A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb:
A word that expresses an action to be taken by a COBOL compiler or 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.

Working-Storage Section:
The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

77-Level-Description-Entry:
A data desccription entry that describes a noncontiguous data item with the level-number 77.

APPENDIX D

COMPOSITE LANGUAGE SKELETON

COMPOSITE LANGUAGE SKELETON

This section contains the composite language skeleton of the American National Standard COBOL. It is intended to display complete and syntactically correct formats.

For the general formats of the four divisions the leftmost margin is equivalent to margin A in a COBOL source program. The first indentation after the leftmost margin is equivalent to margin B in a COBOL source program.

For the general formats of the verbs and conditions the leftmost margin indicates the beginning of the format for a new COBOL verb. The first indentation after the leftmost margin indicates continuation of the format of the COBOL verb.

The following is a summary of the formats shown on the following pages:

- Identification Division general format
- Environment Division general format
- The three formats of the file control entry
- Data Division general format
- The three formats for a data description entry
- The format for a field definition entry
- Procedure Division general format
- General format of verbs listed in alphabetical order
- General format for conditions
- Formats for qualification, subscripting, indexing, and an identifier
- General format for a COPY statement

RM/COBOL LANGUAGE SYNTAX

The RM/COBOL language is based upon the ANSI X3.23-1974 COBOL standard. Minor departures from that document are reflected in the syntax description which follows but are not separately noted. Semantic rules are not changed.

The description is in a condensed form of the standard COBOL syntax notation. In some cases separate formats are combined and general terms are employed for user names.

System-names and implementation restrictions are:

computer-name:	User-defined word
program-name:	8-character name
switch-names:	SWITCH-1, . . . , SWITCH-8
device-types:	PRINT
	INPUT
	OUTPUT
	INPUT-OUTPUT
	RANDOM
external-file-name:	One- to thirty-character name

IDENTIFICATION DIVISION GENERAL FORMAT

```
IDENTIFICATION DIVISION.
-------------- --------

PROGRAM-ID.  program-name.
----------

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

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

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

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

ENVIRONMENT DIVISION GENERAL FORMAT

```
ENVIRONMENT DIVISION.
----------- --------

CONFIGURATION SECTION.
------------- -------

SOURCE-COMPUTER.   computer-name.
---------------

OBJECT-COMPUTER.   computer-name
---------------


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

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

[SPECIAL-NAMES.  [, switch-name
 -------------

  {ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]}]
   --       --                      ---        --
  {OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1]}]
   ---        --                     --        --
  [, alphabet-name IS {STANDARD-1}] ...
                       ----------
                      {NATIVE    }
                       ------

  [, CURRENCY SIGN IS literal-1]
     --------      --

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

```
[INPUT-OUTPUT SECTION.
 ------------ -------

FILE-CONTROL.
------------

  {file-control-entry} ...

[I-O-CONTROL.
 -----------

  [; SAME  AREA FOR file-name-1 [, file-name-2] ...]...  .]]
     ----
```

FILE CONTROL ENTRY GENERAL FORMAT

FORMAT 1

```
SELECT  file-name
------

ASSIGN TO device-type {"external-file-name"}
------                {data-name-1         }

  [; ORGANIZATION IS SEQUENTIAL]
     ------------    ----------

  [; ACCESS MODE IS SEQUENTIAL]
     ------         ----------

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

FORMAT 2

```
SELECT file-name
------

ASSIGN TO RANDOM,  {"external-file-name"}
------    ------   {data-name-1         }

   ; ORGANIZATION IS RELATIVE
     ------------    --------

  [; ACCESS MODE IS { SEQUENTIAL  [, RELATIVE KEY IS data-name-2]} ]
     ------           ----------     --------
                    {{RANDOM }     , RELATIVE KEY IS data-name-2 }
                      ------         --------
                    {{DYNAMIC}                                   }
                      -------

  [; FILE STATUS IS data-name-3].
          ------
```

FORMAT 3

```
SELECT file-name
------

ASSIGN TO  RANDOM, {"external-file-name"}
------     ------  {data-name-1         }

   ; ORGANIZATION IS INDEXED
     ------------    -------

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

  ; RECORD KEY IS data-name-2
    ------

  [; ALTERNATE RECORD KEY IS data-name-3 [WITH DUPLICATES]]...
     --------- ------                         ----------

  [; FILE STATUS IS data-name-4].
          ------
```

DATA DIVISION GENERAL FORMAT

```
DATA DIVISION.
---- --------

[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 LABEL IS [nonnumeric-literal-1]]
     ----- -- -----

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

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

[WORKING-STORAGE SECTION.
 --------------- -------

   [77-level-description-entry] ...  ]
   [record-description-entry   ]

[LINKAGE SECTION.
 ------- -------

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

```
DATA DESCRIPTION ENTRY GENERAL FORMAT
-------------------------------------

FORMAT 1

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

  [; REDEFINES data-name-2]
     ---------

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

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

  [; [SIGN IS] TRAILING [SEPARATE CHARACTER] ]
      ----     --------  --------

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

    [INDEXED BY index-name-1 [, index-name-2] ...] ]
     -------
  [; {SYNCHRONIZED} [LEFT ] ]
      ------------   ----
     {SYNC        } [RIGHT]
      ----           -----

  [; {JUSTIFIED} RIGHT]
      ---------
     {JUST     }
      ----

  [; BLANK WHEN ZERO]
     -----      ----

  [; VALUE IS literal] .
     -----

FORMAT 2

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

FORMAT 3

88 condition-name; {VALUE IS   }
                    -----
                   {VALUES ARE}
                    ------

    literal-1 [{THROUGH} literal-2]
                -------
               {THRU    }
                ----

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

PROCEDURE DIVISON GENERAL FORMAT

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] ... ] ... } ...

END PROGRAM.
--- -------
```

FORMAT 2

```
PROCEDURE DIVISION [USING data-name-1 [,data-name-2] ... ] .
--------- --------  -----

{paragraph-name. [sentence] ... } ...

END PROGRAM.
--- -------
```

GENERAL FORMAT FOR VERBS

```
ACCEPT {identifier-1 [, UNIT {identifier-2}]
------                  ---- {literal-1   }

     [, LINE {identifier-3}] [, POSITION {identifier-4}]
        ---- {literal-2   }     -------- {literal-3   }

     [, SIZE {identifier-5}] [, PROMPT [literal-5]]
        ---- {literal-4   }     ------

     [, ECHO] [, CONVERT] [, TAB] [, ERASE] [, NO BEEP]
        ----     -------     ---     -----     -- ----

     [, {OFF}] [, ON EXCEPTION identifier-6 imperative statement]}...
         ---      -- ---------

ACCEPT identifier FROM {DATE}
------            ---- ----
                       {DAY }
                        ---
                       {TIME}
                        ----

ADD {identifier-1} [, identifier-2] ... TO identifier-m [ROUNDED]
--- {literal-1   } [, literal-2   ]     --               -------

     [; ON SIZE ERROR imperative-statement]
           ---- -----

ADD {identifier-1},  {identifier-2} [, identifier-3] ...
--- {literal-1   }   {literal-2   } [, literal-3   ]

     GIVING identifier-m [ROUNDED]
     ------               -------

     [; ON SIZE ERROR imperative-statement]
           ---- -----

ADD {CORRESPONDING} identifier-1 TO identifier-2
--- -------------                --
    {CORR         }
     ----

      [ROUNDED] [; ON SIZE ERROR imperative-statement]
       -------        ---- -----
```

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
-----                  --  ------- --

     [, procedure-name-3 TO [PROCEED TO] procedure-name-4] ...
                         --  ------- --

CALL {identifier-1} [USING data-name-1 [, data-name-2] ... ]
---- {literal-1   }  -----

CLOSE file-name-1 [{REEL} [WITH NO REWIND] ]
-----               ----        -- ------
                   {UNIT}
                    ----
                   WITH {NO REWIND}
                         -- ------
                        {LOCK     }
                         ----

   [, file-name-2 [{REEL} [WITH NO REWIND] ] ] ...
                    ----        -- ------
                   {UNIT}
                    ----
                 WITH {NO REWIND}
                       -- ------
                      {LOCK     }
                       ----

COMPUTE identifier-1 [ROUNDED] = arithmetic-expression
-------               -------

       [; ON SIZE ERROR imperative-statement]
             ---- -----

DELETE file-name RECORD [; INVALID KEY imperative-statement]
------                     -------

DISPLAY {{identifier-1} [, UNIT {identifier-2} ]
-------  {literal-1   }    ---- {literal-2   }

      [, LINE {identifier-3}][, POSITION {identifier-4}]
         ---- {literal-3   }    -------- {literal-4   }

      [, SIZE {identifier-5}][, BEEP][, ERASE]
         ---- {literal-5   }    ----    -----

      [, {HIGH}][, BLINK][, REVERSE]} ...
          ----     -----    -------

         {LOW }
          ---
```

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
------ {literal-1    } ----              -------

       [; ON SIZE ERROR imperative-statement]
          ---- -----

DIVIDE {identifier-1} INTO {identifier-2} GIVING identifier-3
------ {literal-1    } ---- {literal-2    } ------

       [ROUNDED] [; ON SIZE ERROR imperative-statement]
        -------          ---- -----

DIVIDE {identifier-1} BY {identifier-2} GIVING identifier-3 [ROUNDED]
------ {literal-1    } -- {literal-2    } ------              -------

       [; ON SIZE ERROR imperative-statement]
          ---- -----

EXIT [PROGRAM].
----  -------

GO TO procedure-name-1
--

GO TO procedure-name-1 [, procedure-name-2] ... , procedure-name-n
--

     DEPENDING ON identifier
     ---------

IF condition;  {statement-1   } {; ELSE statement-2  }
--                                 ----
               {NEXT SENTENCE} {; ELSE NEXT SENTENCE}
                ---- --------      ---- ---- --------
```

```
INSPECT identifier-1
-------

     [TALLYING identifier-2 FOR {{ALL     } {identifier-3}}
      --------              ---    ---      {literal-1    }}
                                {{LEADING}
                                   -------
                                {   CHARACTERS           }
                                    ----------

          [{BEFORE} INITIAL {identifier-4}]]
            ------          {literal-2    }
           {AFTER }
            -----

     [REPLACING      {{ALL     } {identifier-5}} BY {identifier-6}
      ---------         ---      {literal-3    }  -- {literal-4    }
                     {{LEADING}                }
                        -------
                     {{FIRST   }               }
                        -----
                     {      CHARACTERS         }
                            ----------

           [{BEFORE}  INITIAL {identifier-7}]]
             ------           {literal-5    }
            {AFTER }
             -----
```

NOTE: The TALLYING option, the REPLACING option, or both options must be selected.

```
MOVE {identifier-1} TO identifier-2 [, identifier-3]...
---- {literal     } --

MOVE {CORRESPONDING} identifier-1 TO identifier-2
---- -------------                --
     {CORR         }
     ----

MULTIPLY {identifier-1} BY identifier-2 [ROUNDED]
-------- {literal-1   } --               -------

      [; ON SIZE ERROR imperative-statement]
            ---- -----

MULTIPLY {identifier-1} BY {identifier-2} GIVING identifier-3
-------- {literal-1   } -- {literal-2   } ------

      [ROUNDED] [; ON SIZE ERROR imperative-statement]
       -------          ---- -----

OPEN {{INPUT file-name-1 [WITH NO REWIND]}
----   -----                   -- ------
          [, file-name-2 [WITH NO REWIND]...
                               -- ------

     {OUTPUT file-name-3 [WITH NO REWIND]}
      ------                   -- ------
          [, file-name-4 [WITH NO REWIND]]...
                               -- ------

     {I-O file-name-5}[, file-name-6]...
      ---

     {EXTEND file-name-7}[, file-name-8]...}...
      ------
```

```
PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
-------                    -------
                          {THRU   }
                           ----

PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
-------                    -------
                          {THRU   }
                           ----
     {identifier-1} TIMES
     {literal-1   } -----

PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
-------                    -------
                          {THRU   }
                           ----
       UNTIL condition-1
       -----

PERFORM procedure-name-1 [{THROUGH} procedure-name-2]
-------                    -------
                          {THRU   }
                           ----

     VARYING {identifier-2} FROM {identifier-3}
     ------- {index-name-1} ---- {index-name-2}
                                 {literal-1   }

       BY {identifier-4} UNTIL condition-1
       -- {literal-3   } -----

     [AFTER {identifier-5} FROM {identifier-6}
      ----- {index-name-3} ---- {index-name-4}
                                {literal-3   }

      BY {identifier-7} UNTIL condition-2
      -- {literal-4   } -----

    [AFTER {identifier-8} FROM {identifier-9}
     ----- {index-name-5} ---- {index-name-6}
                               {literal-5   }

      BY {identifier-10} UNTIL condition-3 ]  ]
      -- {literal-6    } -----
```

```
READ file-name RECORD [INTO identifier]
----                   ----
      [; AT END imperative-statement]
            ---

READ file-name [NEXT] RECORD [WITH NO LOCK] [INTO identifier]
----            ----               -- ----   ----
      [; AT END imperative-statement]
            ---

READ file-name RECORD [WITH NO LOCK] [INTO identifier]
----                        -- ----   ----
      [; KEY IS data-name]
         ---
      [; INVALID KEY imperative-statement]
         -------

REWRITE record-name [FROM identifier]
-------              ----
      [; INVALID KEY imperative-statement]
         -------

SET {identifier-1 [,identifier-2] ...} TO {identifier-3}
--- {index-name-1 [,index-name-2] ...} -- {index-name-3}
                                          {integer-1   }

SET index-name-4 [,index-name-5] ... {UP BY  } {identifier-4}
---                                   -- --    {integer-2   }
                                     {DOWN BY}
                                      ---- --
```

```
START file-name [KEY {IS EQUAL TO      } data-name]
-----            ---     -----
                     {IS =             }
                     {IS GREATER THAN  }
                         -------
                     {IS >             }
                     {IS NOT LESS THAN}
                         --- ----
                     {IS NOT <         }
                         ---

  [; INVALID KEY imperative-statement]
     -------

STOP {RUN       }
---- ---
     {literal   }

SUBTRACT {identifier-1} [, identifier-2] ... FROM identifier-m
-------- {literal-1   } [, literal-2   ]     ----

      [ROUNDED] [; ON SIZE ERROR imperative-statement]
       -------        ---- -----

SUBTRACT {identifier-1} [,identifier-2] ... FROM {identifier-m}
-------- {literal-1   } [,literal-2   ]     ---- {literal-m   }

      GIVING identifier-n [ROUNDED]
      ------               -------

      [; ON SIZE ERROR imperative-statement]
            ---- -----

SUBTRACT {CORRESPONDING} identifier-1 FROM identifier-2 [ROUNDED]
-------- -------------               ----               -------
         {CORR         }
          ----
      [; ON SIZE ERROR imperative-statement]
            ---- -----

UNLOCK file-name-1 RECORD
------             ------
```

```
USE AFTER STANDARD {EXCEPTION}
--- -----          ---------
                   {ERROR    }
                    -----
      PROCEDURE ON {file-name-1 [, file-name-2] ...} .
      ---------
                   {INPUT                          }
                    -----
                   {OUTPUT                         }
                    ------
                   {I-O                            }
                    ---
                   {EXTEND                         }
                    ------

WRITE record-name [FROM identifier-1]
-----              ----

     {BEFORE} ADVANCING {{identifier-2} {LINE }}
      ------            {{integer     } {LINES}}
     {AFTER }           {              PAGE    }
      -----                            ----

WRITE record-name [FROM identifier]
-----              ----
       [; INVALID KEY imperative-statement]
          -------
```

GENERAL FORMAT FOR CONDITIONS

RELATION CONDITION:

```
{identifier-1   } {IS [NOT] GREATER THAN} {identifier-2      }
{literal-1      }      ---  -------       {literal-2         }
{index-name-1   } {IS [NOT] LESS THAN   } {index-name-2      }
                       ---  ----
                  {IS [NOT] EQUAL TO    }
                       ---  -----
                  {IS [NOT] >           }
                       ---
                  {IS [NOT] <           }
                       ---
                  {IS [NOT] =           }
                       ---
```

CLASS CONDITION:

```
  identifier IS [NOT] {NUMERIC    }
                 ---   -------
                      {ALPHABETIC}
                       ----------
```

CONDITION-NAME CONDITION:

```
  condition-name
```

SWITCH-STATUS CONDITION:

```
  condition-name
```

NEGATED SIMPLE CONDITION:

```
  NOT simple-condition
  ---
```

COMBINED CONDITION:

```
  condition {{AND} condition} ...
              ---
             {OR }
              --
```

MISCELLANEOUS FORMATS

QUALIFICATION:

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

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

SUBSCRIPTING:

```
  {data-name      } (subscript-1 [, subscript-2 [, subscript-3] ] )
  {condition-name}
```

INDEXING:

```
  {data-name      } ({index-name-1 [{+} literal-2]}
  {condition-name}  {literal-1     {-}           }

      [, {index-name-2[{+} literal-4]}
         {literal-3    {-}           }

      [, {index-name-3 [{+} literal-6] } ] ] )
         {literal-5     {-}            }
```

IDENTIFIER:

FORMAT 1

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

         [(subscript-1 [, subscript-2 [, subscript-3] ] ) ]
```

FORMAT 2

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

   [, {index-name-2 [{+} literal-4]}
      {literal-3      {-}           }

   [, {index-name-3 [{+} literal-6]} ]])]
      {literal-5      {-}           }
```

GENERAL FORMAT FOR COPY STATEMENT

```
COPY text-name
----
```

COBOL LEVEL OF IMPLEMENTATION

Function Module	Implementation
Nucleus	Level 2.
Table Handling	Level 1+.
Sequential I/O	Level 2.
Relative I/O	Level 2.
Indexed I/O	Level 2.
Sort-Merge	Null.
Report Writer	Null.
Segmentation	Level 1.
Library	Level 1.
Debug	N/S. Conditional compile and execution time interactive debugger.
Inter-program Communication	Level 1.
Communication	Modified ACCEPT and DISPLAY for terminal communication.

ANSI COBOL X3.23 1974

MODULE	FEDERAL INFORMATION PROCESSING STANDARD (FIPS)				RM COBOL
	HIGH	HIGH INTERMEDIATE	LOW INTERMEDIATE	LOW	
NUCLEUS	2	2	1	1	2
TABLE HANDLING	2	2	1	1	1+
SEQUENTIAL I/O	2	2	1	1	2
RELATIVE I/O	2	2	1	-	2
INDEXED I/O	2	-	-	-	2
SORT-MERGE	2	1	-	-	-
REPORT WRITER	-	-	-	-	-
SEGMENTATION	2	1	1	-	1
LIBRARY	2	1	1	-	1
DEBUG	2	2	1	-	N/S
INTER-PROGRAM COMMUNICATION	2	2	1	-	1+
COMMUNICATION	2	2	-	-	N/S

N/S = Nonstandard

EXTENSIONS BEYOND STATED LEVELS

Level 2 Nucleus (2 NUC):

- Data description includes a USAGE type of COMPUTATIONAL-1 or COMP-1 for describing single word twos complement signed binary data (nonstandard).
- Data description includes a USAGE type of COMPUTATIONAL-3 or COMP-3 for describing packed decimal data (nonstandard).
- The ACCEPT statement allows multiple operands (nonstandard).
- The ACCEPT statement includes syntax for specifying CRT control information (nonstandard).
- The DISPLAY statement includes syntax for specifying CRT control information (nonstandard).

Level 1 Table Handling (1 TBL):

- Variable group size (OCCURS DEPENDING).

Level 2 Sequential I-O (2 SEQ):

- The file control SELECT clause allows specification of the external file name as a literal or data item (nonstandard).
- The READ statement includes the WITH NO LOCK option (nonstandard).
- The UNLOCK statement is included (nonstandard).

Level 2 Relative I-O (2 REL):

- The file control SELECT clause allows specification of the external file name as a literal or data item (nonstandard).
- The READ statement includes the WITH NO LOCK option (nonstandard).
- The UNLOCK statement is included. (nonstandard).

Level 2 Indexed I-O (2 INX):

- The file control SELECT clause allows specification of the external file name as a literal or data item (nonstandard).
- The READ statement includes the WITH NO LOCK option (nonstandard).
- The UNLOCK statement is included (nonstandard).

Level 1 Debug (1 DEB):

- An interactive execution time debug facility is provided (nonstandard).

Level 1 Inter-Program Communication (1 IPC):

- The CALL statement allows literals in USING phrase (nonstandard).
- The CALL statement allows identifiers in the USING phrase to be described with level number 01 through 49 and level number 77 (nonstandard).
- The CALL statement supports specification of a variable program name as identifier-1 (level 2 IPC).

Level 1 Communication (1 COM):

- ACCEPT and DISPLAY allow specification of complete screen format in the Procedure Division (nonstandard).

EXCEPTIONS TO STATED LEVELS

Level 2 Nucleus (2 NUC):

- DATE-COMPILED is not supported in the Identification Divison.
- In data description the SIGN clause cannot specify LEADING for the operational sign; omission of the SEPARATE phrase has no effect; all operational signs are separate trailing characters.
- Alphabet-name IS literal or implementor-name may not be specified in SPECIAL-NAMES paragraph.
- Multiple results are not supported in arithmetic statements.
- REMAINDER is not supported in DIVIDE statement.
- A procedure-name is required in GO TO statements.
- INSPECT data items are restricted to single character.
- Compound TALLYING and REPLACING clauses in the INSPECT statement are not supported.
- When used in the Procedure Division, the numeric literal in the ALL form of a figurative constant may not contain more than one character.
- Arithmetic expressions may be used only in COMPUTE statements.
- Exponentiation to a noninteger power is not supported.
- Sign conditions are not supported.
- Abbreviated combined relation conditions are not supported.
- The STRING and UNSTRING statements are not supported.

Level 2 Sequential I-O (2 SEQ):

- OPTIONAL and RESERVE may not be specified in the SELECT clause.
- RERUN, SAME AREA or MULTIPLE FILE clauses are not supported in I-O-CONTROL.

- CODE-SET and LINAGE clauses may not be specified in a file description entry.

- The menmonic-name and EOP options of the WRITE statement are not supported.

- The REVERSED option of the OPEN statement is not supported.

- The FOR REMOVAL option of the CLOSE statement is not supported.

Level 2 Relative I-O (2 REL):

The RESERVE clause of the SELECT entry is not supported.

- RERUN, SAME AEA or MULTIPLE FILE clauses are not supported in I-O-CONTROL.

- The VALUE OF clause in an FD entry must not specify a data name.

Level 2 Indexed I-O (2 INX):

- The RESERVE clause of the SELECT entry is not supported.

- RERUN, SAME AREA or MULTIPLE FILE clauses are not supported in I-O-CONTROL.

Level 1 Segmentation (1 SEQ):

- All independent segments must physically follow the fixed permanent segments in the source program.

Level 1 Library (1 LIB):

- A copy sentence must be the last entry in area B of a source record.

Level 1 Inter-Program Communication (1 IPC):

- A CALLed program is automatically cancelled upon execution of the EXIT PROGRAM statement.

APPENDIX E

SAMPLE PROGRAMS

```
TRS-80 Model II COBOL (RM/COBOL 1.2)              4/ 8/80  08.33.41  PAGE     1
SOURCE FILE: EXAMPLE1                            OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B............................................................ID.....

    1       000100 IDENTIFICATION DIVISION.
    2       000110   PROGRAM-ID.                   EXAMPLE1.
    3       000120   AUTHOR.                       D.H.WEISS.
    4       000130   INSTALLATION.                 RADIO SHACK.
    5       000140   DATE-WRITTEN.                 JAN 80.
    6       000150
    7       000160
    8       000170 ENVIRONMENT DIVISION.
    9       000180  CONFIGURATION SECTION.
   10       000190   SOURCE-COMPUTER.             MODELII.
   11       000200   OBJECT-COMPUTER.             MODELII-64K.
   12       000210  INPUT-OUTPUT SECTION.
   13       000220   FILE-CONTROL.
   14       000230*                               FOR A RELATIVE FILE:
   15       000240*                                 ORGANIZATION MUST BE RELATIVE
   16       000250*                                 ACCESS MODE MAY BE SEQUENTIAL OR RANDOM
   17       000260*                                 THE RELATIVE KEY DATA-NAME MUST BE
   18       000270*                                   DEFINED IN WORKING-STORAGE AS
   19       000280*                                   AN UNSIGNED INTEGER
   20       000290     SELECT NAME-ADDRESS,
   21       000300         ASSIGN TO RANDOM, "COBNAMES/REL",
   22       000310         ORGANIZATION IS RELATIVE
   23       000320         ACCESS MODE IS RANDOM
   24       000330         RELATIVE KEY IS WST-REL-KEY.
   25       000340
   26       000350
   27       000360 DATA DIVISION.
   28       000370  FILE SECTION.
   29       000380*                              EACH RECORD OF THE FILE WILL CONTAIN:
   30       000390*                                FULL NAME
   31       000400*                                TWO LINES OF ADDRESS
   32       000410*                                AND A SEPARATE FIELD FOR ZIPCODE
   33       000420*                              FILLER IS INCLUDED TO ALLOW ROOM FOR
   34       000430*                                EXPANSION WITHOUT THE NEED TO RESIZE
   35       000440*                                THE RECORD
   36       000450 FD  NAME-ADDRESS
   37       000460         BLOCK CONTAINS 1 RECORDS
   38       000470         RECORD CONTAINS 120 CHARACTERS
   39       000480         LABEL RECORDS ARE STANDARD
   40       000490         DATA RECORD IS NAME-RECD.
   41       000500 01  NAME-RECD.
   42       000510     03  NAM-NAME                PIC X(30).
   43       000520     03  NAM-ADD1                PIC X(30).
   44       000530     03  NAM-ADD2                PIC X(30).
   45       000540     03  NAM-ZIPC                PIC 9(5).
   46       000550     03  FILLER                  PIC X(25).
   47       000560
   48       000570 WORKING-STORAGE SECTION.
   49       000580*                              COUNT OF NUMBER OF RECORDS PROCESSED
   50       000590   77  WST-COUNT                 PIC S999   COMP-3  VALUE 0.
   51       000600*                              CONSTANT FOR USE AS DISPLAY SIZE
   52       000610   77  WST-THIRTY                PIC S99    COMP-3  VALUE 30.
   53       000620*                              SCREEN LINE COUNT TO FACILITATE CLEARING
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                    4/ 8/80  08.33.41  PAGE     2
SOURCE FILE: EXAMPLE1                                  OPTION LIST: (L O T X

LINE   DEBUG PG/LN  A...B............................................................ID.....

  54         000630*                                   LINES AFTER EACH RECORD IS WRITTEN
  55         000640    77  WST-LINE                      PIC S99   COMP-3.
  56         000650*                                   MUST BE DEFINED FOR RELATIVE FILE TO
  57         000660*                                   ACCESS A RECORD RANDOMLY
  58         000670    77  WST-REL-KEY                   PIC 9               VALUE 1.
  59         000680*                                   FIELD WITH USAGE OF DISPLAY TO HOLD
  60         000690*                                   WST-COUNT FOR DISPLAYING ON SCREEN
  61         000700    77  DSP-COUNT                     PIC 999.
  62         000710
  63         000720
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                 4/ 8/80  08.33.41  PAGE     3
SOURCE FILE: EXAMPLE1                               OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B............................................................ID.....

  64         000730/
  65         000740 PROCEDURE DIVISION.
  66   >0000 000750   A005-INITIALIZE.
  67   >0000 000760     DISPLAY "BEGIN EXAMPLE1", LINE 1, POSITION 1, ERASE.
  68         000770
  69   >000C 000780   A010-MAINLINE.
  70         000790*                                STRUCTURED PROGRAM DESIGN USING
  71         000800*                                  ONE MAINLINE PARAGRAPH TO CONTROL
  72         000810*                                  ALL THE PROCESSING
  73         000820*
  74         000830*                                OPENS & CLOSES FILE
  75         000840*                                DISPLAYS ENDING MESSAGE
  76         000850*                                PERFORMS EXTENDED PROCESSING THRU
  77         000860*                                  OTHER MODULES
  78         000870*
  79   >000C 000880     OPEN OUTPUT NAME-ADDRESS.
  80   >0012 000890     PERFORM B005-PROCESS-INPUT THRU B005-EXIT  3 TIMES.
  81   >0020 000900     CLOSE NAME-ADDRESS.
  82   >0026 000910     DISPLAY "END EXAMPLE1 - WROTE  ", LINE 23.
  83   >002C 000920     MOVE WST-COUNT TO DSP-COUNT.
  84   >0030 000930     DISPLAY DSP-COUNT, LINE 23, POSITION 22.
  85   >0038 000940     DISPLAY " RECORDS", LINE 23, POSITION 25.
  86   >0040 000950     STOP RUN.
  87         000960
  88         000970
  89   >0042 000980   B005-PROCESS-INPUT.
  90         000990*                                SUB-CONTROL MODULE
  91         001000*
  92         001010*                                PERFORMS CLEAR MODULE USING FORMAT 4
  93         001020*                                PERFORMS EXTENDED PROCESSING THRU
  94         001030*                                  OTHER MODULES
  95         001040*
  96   >0042 001050     PERFORM U010-CLEAR-LINES THRU U010-EXIT
  97         001060         VARYING WST-LINE FROM 10 BY 2 UNTIL WST-LINE > 16.
  98   >0056 001070     PERFORM B010-DISPLAY-CAPTIONS THRU B010-EXIT.
  99   >0058 001080     PERFORM B030-ENTER-NAMES THRU B030-EXIT.
 100   >005C 001090   B005-EXIT.    EXIT.
 101         001100
 102   >005E 001110   B010-DISPLAY-CAPTIONS.
 103         001120*                                SELF-CONTAINED DISPLAY MODULE
 104         001130*
 105         001140*                                DISPLAYS FIELD CAPTIONS
 106         001150*
 107   >005E 001160     DISPLAY "ENTER NAME (30)", LINE 10, POSITION 10.
 108   >0066 001170     DISPLAY "ENTER ADD1 (30)", LINE 12, POSITION 10.
 109   >006E 001180     DISPLAY "ENTER ADD2 (30)", LINE 14, POSITION 10.
 110   >0076 001190     DISPLAY "ENTER ZIPC (5)", LINE 16, POSITION 10.
 111   >0080 001200   B010-EXIT.    EXIT.
 112         001210
 113   >0082 001220   B030-ENTER-NAMES.
 114         001230*                                INPUT MODULE
 115         001240*
 116         001250*                                CLEARS BUFFER FOR NEW RECORD
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                 4/ 8/80  08.33.41  PAGE     4
SOURCE FILE: EXAMPLE1                                OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B.............................................................ID.....

 117        001260*                                  ACCEPTS THE FOUR FIELDS OF THE RECORD
 118        001270*                                    ECHOING THE DATA BACK TO THE SCREEN
 119        001280*                                  PERFORMS OTHER MODULE TO WRITE NEW
 120        001290*                                    RECORD
 121        001300*
 122  >0082 001310      MOVE SPACES TO NAME-RECD.
 123  >0086 001320      ACCEPT NAM-NAME, LINE 10, POSITION 42,
 124        001330          SIZE WST-THIRTY, ECHO.
 125  >0092 001340      ACCEPT NAM-ADD1, LINE 12, POSITION 42,
 126        001350          SIZE WST-THIRTY, ECHO.
 127  >009E 001360      ACCEPT NAM-ADD2, LINE 14, POSITION 42,
 128        001370          SIZE WST-THIRTY, ECHO.
 129  >00AA 001380      ACCEPT NAM-ZIPC, LINE 16, POSITION 42,
 130        001390          SIZE 5, ECHO.
 131  >00B6 001400      PERFORM W010-WRITE-NAME THRU W010-EXIT.
 132  >00BA 001410    B030-EXIT.    EXIT.
 133        001420
 134        001430
 135  >00BC 001440    U010-CLEAR-LINES.
 136        001450*                                  SELF-CONTAINED CLEAR MODULE
 137        001460*
 138        001470*                                  CLEARS DATA FROM PREVIOUSLY
 139        001480*                                    ACCEPTED FIELDS
 140        001490*
 141  >00BC 001500      DISPLAY SPACES, LINE WST-LINE, POSITION 42, SIZE 38.
 142  >00C8 001510    U010-EXIT.
 143        001520
 144        001530
 145  >00CA 001540    W010-WRITE-NAME.
 146        001550*                                  SELF-CONTAINED WRITE MODULE
 147        001560*
 148        001570*                                  WRITES NEW RECORD TO FILE
 149        001580*                                  KEEPS COUNT
 150        001590*
 151  >00CA 001600      WRITE NAME-RECD  INVALID KEY  GO TO Z999-ABORT.
 152  >00D8 001610      ADD 1 TO WST-COUNT.
 153  >00DE 001620      ADD 1 TO WST-REL-KEY.
 154  >00E6 001630    W010-EXIT.    EXIT.
 155        001640
 156        001650
 157  >00E8 001660    Z999-ABORT.
 158        001670*                                  ABNORMAL TERMINATION OF PROGRAM
 159        001680*
 160  >00E8 001690      DISPLAY "INVALID KEY - ABORT".
 161  >00EC 001700      STOP RUN.
 162        001710
 163        001720 END PROGRAM.
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                  4/ 8/80  08.33.41  PAGE     5
SOURCE FILE: EXAMPLE1                                OPTION LIST: (L O T X

 ADDRESS  SIZE DEBUG ORDER TYPE                      NAM


          120               FILE                     NAME-ADDRESS
   >0002  120  GRP     0    GROUP                     NAME-RECD
   >0002   30  ANS     0    ALPHANUMERIC                NAM-NAME
   >0020   30  ANS     0    ALPHANUMERIC                NAM-ADD1
   >003E   30  ANS     0    ALPHANUMERIC                NAM-ADD2
   >005C    5  NSU     0    NUMERIC UNSIGNED            NAM-ZIPC

   >007E    2  NPS     0    PACKED SIGNED            WST-COUNT

   >0080    2  NPS     0    PACKED SIGNED            WST-THIRTY

   >0082    2  NPS     0    PACKED SIGNED            WST-LINE

   >0084    1  NSU     0    NUMERIC UNSIGNED         WST-REL-KEY

   >0086    3  NSU     0    NUMERIC UNSIGNED         DSP-COUNT

READ ONLY BYTE SIZE =        >0282

READ/WRITE BYTE SIZE =       >00E4

OVERLAY SEGMENT BYTE SIZE =  >0000

TOTAL BYTE SIZE =            >0366

    0 ERRORS

    0 WARNINGS
```

```
SOURCE FILE: EXAMPLE1                                   OPTION LIST: (L O T X

CROSS REFERENCE                  /DECL/ *DEST

A005-INITIALIZE                  /0066/
A010-MAINLINE                    /0069/
B005-EXIT                         0080  /0100/
B005-PROCESS-INPUT                0080  /0089/
B010-DISPLAY-CAPTIONS             0098  /0102/
B010-EXIT                         0098  /0111/
B030-ENTER-NAMES                  0099  /0113/
B030-EXIT                         0099  /0132/
DSP-COUNT                        /0061/ *0083*  0084
NAME-ADDRESS                     /0020/ /0036/  0079   0081
NAME-RECD                        /0041/ *0122* *0151*
NAM-ADD1                         /0043/ *0125*
NAM-ADD2                         /0044/ *0127*
NAM-NAME                         /0042/ *0123*
NAM-ZIPC                         /0045/ *0129*
U010-CLEAR-LINES                  0096  /0135/
U010-EXIT                         0096  /0142/
WST-COUNT                        /0050/  0083  *0152*
WST-LINE                         /0055/ *0097*  0097   0141
WST-REL-KEY                      *0024* /0058/ *0153*
WST-THIRTY                       /0052/  0124   0126   0128
W010-EXIT                         0131  /0154/
W010-WRITE-NAME                   0131  /0145/
Z999-ABORT                        0151  /0157/
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                  4/ 8/80  08.35.27  PAGE      1
SOURCE FILE: EXAMPLE2                                OPTION LIST: (L O T X

LINE   DEBUG PG/LN  A...B............................................................ID.....

   1         000100 IDENTIFICATION DIVISION.
   2         000110   PROGRAM-ID.                       EXAMPLE2.
   3         000120   AUTHOR.                           D.H.WEISS.
   4         000130   INSTALLATION.                     RADIO SHACK.
   5         000140   DATE-WRITTEN.                     JAN 80.
   6         000150
   7         000160
   8         000170 ENVIRONMENT DIVISION.
   9         000180  CONFIGURATION SECTION.
  10         000190   SOURCE-COMPUTER.                 MODELII.
  11         000200   OBJECT-COMPUTER.                 MODELII-64K.
  12         000210  INPUT-OUTPUT SECTION.
  13         000220   FILE-CONTROL.
  14         000230*                                   SAME AS EXAMPLE 1
  15         000240     SELECT NAME-ADDRESS,
  16         000250         ASSIGN TO RANDOM, "COBNAMES/REL",
  17         000260         ORGANIZATION IS RELATIVE
  18         000270         ACCESS MODE IS RANDOM
  19         000280         RELATIVE KEY IS WST-REL-KEY.
  20         000290
  21         000300
  22         000310 DATA DIVISION.
  23         000320  FILE SECTION.
  24         000330*                                   SAME AS EXAMPLE 1
  25         000340 FD  NAME-ADDRESS
  26         000350         BLOCK CONTAINS 1 RECORDS
  27         000360         RECORD CONTAINS 120 CHARACTERS
  28         000370         LABEL RECORDS ARE STANDARD
  29         000380         DATA RECORD IS NAME-RECD.
  30         000390 01  NAME-RECD.
  31         000400     03  NAM-NAME                     PIC X(30).
  32         000410     03  NAM-ADD1                     PIC X(30).
  33         000420     03  NAM-ADD2                     PIC X(30).
  34         000430     03  NAM-ZIPC                     PIC 9(5).
  35         000440     03  FILLER                       PIC X(25).
  36         000450
  37         000460 WORKING-STORAGE SECTION.
  38         000470   77  WST-COUNT                      PIC S999  COMP-3  VALUE 0.
  39         000480   77  WST-THIRTY                     PIC S99   COMP-3  VALUE 30.
  40         000490   77  WST-LINE                       PIC S99   COMP-3.
  41         000500   77  WST-REL-KEY                    PIC 9             VALUE 1.
  42         000510   77  DSP-COUNT                      PIC 999.
  43         000520*                                   NEW FIELD "ANSWER" PROVIDES A PAUSE
  44         000530*                                     TO FACILITATE OPERATOR CONTROL
  45         000540   77  ANSWER                         PIC X.
  46         000550
  47         000560
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)          4/ 8/80  08.35.27  PAGE     2
SOURCE FILE: EXAMPLE2                         OPTION LIST: (L O T X

LINE  DEBUG PG/LN  A...B.......................................................ID.....

  48        000570/
  49        000580 PROCEDURE DIVISION.
  50  >0000 000590   A005-INITIALIZE.
  51  >0000 000600     DISPLAY "BEGIN EXAMPLE2", LINE 1, POSITION 1, ERASE.
  52        000610
  53  >000C 000620   A010-MAINLINE.
  54        000630*                                   MAINLINE PARAGRAPH
  55        000640*
  56  >000C 000650     OPEN INPUT NAME-ADDRESS.
  57  >0012 000660     PERFORM B005-PROCESS-INPUT THRU B005-EXIT  3 TIMES.
  58  >0020 000670     CLOSE NAME-ADDRESS.
  59  >0026 000680     DISPLAY "END EXAMPLE2 - READ  ", LINE 23.
  60  >002C 000690     MOVE WST-COUNT TO DSP-COUNT.
  61  >0030 000700     DISPLAY DSP-COUNT, LINE 23, POSITION 22.
  62  >0038 000710     DISPLAY " RECORDS", LINE 23, POSITION 25.
  63  >0040 000720     STOP RUN.
  64        000730
  65        000740
  66  >0042 000750   B005-PROCESS-INPUT.
  67        000760*                                   SUB-CONTROL MODULE
  68        000770*
  69  >0042 000780     PERFORM U010-CLEAR-LINES THRU U010-EXIT
  70        000790         VARYING WST-LINE FROM 10 BY 2 UNTIL WST-LINE > 16.
  71  >0056 000800     PERFORM B010-DISPLAY-CAPTIONS THRU B010-EXIT.
  72  >0058 000810     PERFORM B030-ENTER-NAMES THRU B030-EXIT.
  73  >005A 000820     PERFORM B050-SUSPEND-PROCESSING THRU B050-EXIT.
  74  >005E 000830   B005-EXIT.    EXIT.
  75        000840
  76  >0060 000850   B010-DISPLAY-CAPTIONS.
  77        000860*                                   SAME AS EXAMPLE 1
  78        000870*
  79  >0060 000880     DISPLAY "ENTER NAME (30)", LINE 10, POSITION 10.
  80  >0068 000890     DISPLAY "ENTER ADD1 (30)", LINE 12, POSITION 10.
  81  >0070 000900     DISPLAY "ENTER ADD2 (30)", LINE 14, POSITION 10.
  82  >0078 000910     DISPLAY "ENTER ZIPC (5)", LINE 16, POSITION 10.
  83  >0082 000920   B010-EXIT.    EXIT.
  84        000930
  85  >0084 000940   B030-ENTER-NAMES.
  86        000950*                                   DISPLAY MODULE
  87        000960*
  88        000970*                                   PERFORMS OTHER MODULE TO READ
  89        000980*                                     NEXT RECORD
  90        000990*                                   DISPLAYS EACH FIELD,
  91        001000*                                     OPTION FOR REVERSE VIDEO USED
  92        001010*                                     ON ZIPCODE FIELD
  93        001020*
  94  >0084 001030     PERFORM R010-READ-NAME THRU R010-EXIT.
  95  >0086 001040     DISPLAY NAM-NAME, LINE 10, POSITION 42,
  96        001050         SIZE WST-THIRTY.
  97  >0090 001060     DISPLAY NAM-ADD1, LINE 12, POSITION 42,
  98        001070         SIZE WST-THIRTY.
  99  >009A 001080     DISPLAY NAM-ADD2, LINE 14, POSITION 42,
 100        001090         SIZE WST-THIRTY.
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                    4/ 8/80  08.35.27  PAGE     3
SOURCE FILE: EXAMPLE2                                  OPTION LIST: (L O T X

  NE   DEBUG PG/LN  A...B...........................................................ID.....

 101   >00A4 001100      DISPLAY NAM-ZIPC, LINE 16, POSITION 42,
 102         001110          SIZE 5, REVERSE.
 103   >00B3 001120    B030-EXIT.    EXIT.
 104         001130
 105         001140
 106   >00B5 001150    B050-SUSPEND-PROCESSING.
 107         001160*                                      SELF-CONTAINED SUSPEND MODULE
 108         001170*
 109         001180*                                      PROVIDES OPERATOR CONTROL OF
 110         001190*                                        DISPLAY SO HE CAN VIEW EACH
 111         001200*                                        RECORD AS LONG AS HE WISHES
 112         001210*
 113   >00B5 001220      DISPLAY "NEXT RECORD?", LINE 20, POSITION 20
 114         001230      ACCEPT ANSWER, LINE 0, POSITION 0.
 115   >00C7 001240    B050-EXIT.    EXIT.
 116         001250
 117         001260
 118   >00C9 001270    U010-CLEAR-LINES.
 119         001280*                                      SAME AS EXAMPLE 1
 120         001290*
 121   >00C9 001300      DISPLAY SPACES, LINE WST-LINE, POSITION 42, SIZE 38.
 122   >00D5 001310    U010-EXIT.
 123         001320
 124         001330
 125   >00D7 001340    R010-READ-NAME.
 126         001350*                                      SELF-CONTAINED READ MODULE
 127         001360*
 128   >00D7 001370      READ NAME-ADDRESS  INVALID KEY  GO TO Z999-ABORT.
 129   >00E3 001380      ADD 1 TO WST-COUNT.
  30   >00E9 001390      ADD 1 TO WST-REL-KEY.
 131   >00F1 001400    R010-EXIT.    EXIT.
 132         001410
 133         001420
 134   >00F3 001430    Z999-ABORT.
 135         001440*                                      SAME AS EXAMPLE 1
 136         001450*
 137   >00F3 001460      DISPLAY "INVALID KEY - ABORT".
 138   >00F7 001470      STOP RUN.
 139         001480
 140         001490 END PROGRAM.
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)                     4/ 8/80  08.35.27   PAGE      4
SOURCE FILE: EXAMPLE2                                   OPTION LIST: (L O T X

 ADDRESS  SIZE DEBUG ORDER TYPE                         NAM


           120                FILE                      NAME-ADDRESS
  >0002    120  GRP     0     GROUP                      NAME-RECD
  >0002     30  ANS     0     ALPHANUMERIC                NAM-NAME
  >0020     30  ANS     0     ALPHANUMERIC                NAM-ADD1
  >003E     30  ANS     0     ALPHANUMERIC                NAM-ADD2
  >005C      5  NSU     0     NUMERIC UNSIGNED            NAM-ZIPC

  >007E      2  NPS     0     PACKED SIGNED             WST-COUNT

  >0080      2  NPS     0     PACKED SIGNED             WST-THIRTY

  >0082      2  NPS     0     PACKED SIGNED             WST-LINE

  >0084      1  NSU     0     NUMERIC UNSIGNED          WST-REL-KEY

  >0086      3  NSU     0     NUMERIC UNSIGNED          DSP-COUNT

  >008A      1  ANS     0     ALPHANUMERIC              ANSWER

READ ONLY BYTE SIZE =          >02B0

READ/WRITE BYTE SIZE =         >00EA

OVERLAY SEGMENT BYTE SIZE =    >0000

TOTAL BYTE SIZE =              >039A

    0 ERRORS

    0 WARNINGS
```

```
TRS-80 Model II COBOL (RM/COBOL 1.2)              4/ 8/80  08.35.27  PAGE     5
SOURCE FILE: EXAMPLE2                            OPTION LIST: (L O T X

CROSS REFERENCE                 /DECL/ *DEST

ANSWER                          /0045/ *0114*
A005-INITIALIZE                 /0050/
A010-MAINLINE                   /0053/
B005-EXIT                        0057  /0074/
B005-PROCESS-INPUT               0057  /0066/
B010-DISPLAY-CAPTIONS            0071  /0076/
B010-EXIT                        0071  /0083/
B030-ENTER-NAMES                 0072  /0085/
B030-EXIT                        0072  /0103/
B050-EXIT                        0073  /0115/
B050-SUSPEND-PROCESSING          0073  /0106/
DSP-COUNT                       /0042/ *0060*  0061
NAME-ADDRESS                    /0015/ /0025/  0056   0058   0128
NAME-RECD                       /0030/
NAM-ADD1                        /0032/  0097
NAM-ADD2                        /0033/  0099
NAM-NAME                        /0031/  0095
NAM-ZIPC                        /0034/  0101
R010-EXIT                        0094  /0131/
R010-READ-NAME                   0094  /0125/
U010-CLEAR-LINES                 0069  /0118/
U010-EXIT                        0069  /0122/
WST-COUNT                       /0038/  0060  *0129*
WST-LINE                        /0040/ *0070*  0070   0121
WST-REL-KEY                     *0019* /0041/ *0130*
WST-THIRTY                      /0039/  0096   0098   0100
Z999-ABORT                       0128  /0134/
```

RADIO SHACK A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U. K.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

PRINTED IN U.S.A.