WANG

VS

COBOL Reference Manual

VS COBOL Reference Manual

6th Edition — February 1984
Copyright © Wang Laboratories, Inc., 1984
800-1201-06

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

PREFACE

This reference manual explains and details the use of COBOL (COmmon Business Oriented Language) for use in performing data processing functions on the Wang VS. The VS Programmer's Introduction and the VS Program Development Tools discuss how to log on to the system and enter COBOL source code from the workstation; this manual assumes knowledge of that material.

This manual has two parts: a tutorial section (Part I, Chapters 1 through 7), and a reference section (Part II Chapters 8 through 13 and the appendices). The tutorial section discusses file processing and is organized according to the different kinds of I/O devices that the VS uses. It also includes information about disk files, extended disk file processing, workstation files, print files, tape files, and the SORT-MERGE module.

The reference section provides a detailed discussion of all the linguistic units of COBOL, their functions, and the rules governing their use. The appendices present the following information:

Appendix A	is a list if VS COBOL reserved words.
Appendix B	lists and explains the available compiler options.
Appendix C	lists and explains Field Attribute Characters for controlling workstation screen display characteristics.
Appendix D	describes the use if the workstation screen order area for controlling screen and workstation actions.
Appendix E	lists and explains File Status return codes.
Appendix F	explains the hexadecimal characters used to control the writing of a record to a printer file.
Appendix G	explains the storing of the intermediate results of arithmetic operations.
Appendix H	describes the protocol required for passing parameters from one COBOL program to another COBOL program.
Appendix I	is a comparison of VS, ANSI, and FIPS COBOL standards.
Appendix J	explains the use of extension rights.
Appendix K	explains the rules for Segmentation.

The user will find the following helpful for use in conjunction with this manual.

Title	Number
<u>VS COBOL Quick Reference</u>	800-6200
<u>VS COBOL Conversion Guide</u>	800-1204
<u>VS COBOL Coding Form</u>	800-5206
<u>VS DMS/TX Reference</u>	800-1128
<u>VS Programmer's Introduction</u>	800-1101
<u>VS Program Development Tools</u>	800-1307
<u>VS Utilities Reference</u>	800-1303
<u>VS Procedure Langauge Reference</u>	800-1205
<u>VS Procedure Language Pocket Guide</u>	800-6201

SUMMARY OF CHANGES

FOR THE 6TH EDITION OF THE VS COBOL REFERENCE MANUAL

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
Technical Changes	Explanation of the ANSI modules supported in VS COBOL	1-1
	Removal of ADMS references	Chapters 1, 3, 8, 11, Appendices A, I
	Advanced Sharing changed to DMS Sharing	1-5, 2-27 to 2-34 12-52
	BLOCK entry of FD	11-13, 11-14
	COPY Statement	12-33
	CORRESPONDING phrase	11-7 to 11-8, 12-10 12-23, 12-24, 12-67, 12-68, 12-72, 12-90, 12-95, 12-99, 12-101, 12-102, 12-141, 12-142
	DMS/TX	v, vi, 1-5 to 1-6, Chapter 3, 11-13, 11-20, 11-22 to 11-24, 12-50, 12-112, 12-148, 12-150, A-1, A-3, I-22, I-25, J-1
	Extension-Rights	Appendix J
	Lower Case Option	Appendix B
	OCCURS	11-35 to 11-36

SUMMARY OF CHANGES (continued)

FOR THE 6TH EDITION OF THE VS COBOL REFERENCE MANUAL

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
	Qualified data names	10-15, 10-18, 11-5 to 11-7, 11-10, 11-11, 11-20, 11-25, 11-34, 11-58, 11-63, 11-64, 11-66, 12-65, 12-94, 12-101, 12-104, 12-107, 12-110, 12-114, 12-127, 12-133, 12-135, 12-151, 12-154, 12-157, 13-2, I-4
	Relative File Support	1-1, 1-4, 2-1, 2-4, 2-24 to 2-27, 10-18 to 10-20, 11-19, 12-31, 12-38 to 12-39, 12-79 to 12-81, 12-98 to 12-100, 12-110 to 12-111, 12-135 to 12-136, 12-157 to 12-159, E-7 to E-8, I-9
	SEARCH ALL Support	12-113, 12-117 to 12-122, I-22
	Segmentation	Appendix K
	SORT-MERGE Support	1-1, Chapter 7, 10-21, 11-25, 11-26, 12-64 to 12-66, 12-101 to 12-103, 12-126 to 12-130, I-11, I-22, K-2, K-4
	STRING Support	12-138 to 12-140
	UNSTRING Support	12-143 to 12-147
Editorial Changes	Miscellaneous editorial changes	Chapters 1, 2, 3, 4, 7, 8, 9, 10, Appendices A, B, G, I

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

FIGURES

TABLES

ACKNOWLEDGMENT

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

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

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

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

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

PART I

TUTORIAL

CHAPTER 1
COBOL CONCEPTS

1.1 INTRODUCTION TO VS COBOL

COBOL is an acronym for COmmon Business-Oriented Language. This programming language is used in business applications that require repetitious updating of files, applications whose goal is to maintain up-to-date information that can be used as input by other processing tools, such as report generation. Some benefits of COBOL as a programming language are:

- COBOL is subject to industry-wide standards administered by the American National Standards Institute (ANSI). Therefore, COBOL is highly compatible among manufacturers.
- COBOL programs are relatively easy to read, as compared to programs in other languages.
- COBOL provides record formatting, data manipulation, and file handling capabilities that are important in data processing applications.
- Because COBOL is a major programming language, there is a large pool of trained programmers and analysts.

Level 1 of the following ANSI standard modules is available on the Wang VS: The Nucleus, Table Handling, Sequential I-O, Indexed I-O, Segmentation, Library, Debug, and Inter-program Communication. Full Level 2 support is provided for the Relative I/O module. VS COBOL os further enhanced with many other higher level features, including SORT/MERGE and Qualified Names. Refer to Appendix J for a comparison of VS, ANSI, and FIPS COBOL standards.

VS COBOL is enhanced with a number of extensions that support the interactive capabilities of the VS and advanced data management operations. These extensions, together with other VS features provide additional benefits to the COBOL programmer, such as:

- Easy-to-follow menus facilitating interactive communication with the VS.

- The user can enter, validate, and correct data and can enter edit, compile, debug, and run programs directly from the VS workstation. Results appear on the workstation screen immediately. This reduces programming time.

- The user can invoke a complete set of system utilities directly from the workstation to perform common functions such as sorting, copying, and program linking.

- An interactive symbolic debugger allows run-time debugging from the workstation. The user can inspect and modify data by referencing data names rather than addresses in memory.

- VS COBOL enhances standard COBOL with a transaction recovery system providing multiple user sharing and rollback recovery of indexed data files.

- The COBOL programmer can use the VS Procedure language to communicate with the system. This capability reduces the syntactical complexity typical of command or job control languages.

1.2 STRUCTURE OF COBOL PROGRAMS

COBOL has rules for organization and syntax. Every COBOL source program has four divisions, each of which has specific mandatory and optional elements. These elements include sections, paragraphs, entries, statements, clauses, phrases, and sentences. Section 8.5 describes these elements.

The programmer must write the divisions in proper sequence, and each must begin with the proper division header. In sequence, the division headers are

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

1.2.1 Identification Division

The Identification Division defines a unique name that identifies the program. It can also include comments about the program, such as the author's name, the installation, and the date it was written and/or compiled.

For a detailed discussion of the Identification Division, refer to Chapter 9.

1.2.2 Environment Division

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

The Configuration Section describes the characteristics of the particular computer(s) to be used to the compiler. It can also contain a SPECIAL-NAMES paragraph and, in VS COBOL, a paragraph that defines figurative constants.

The Input-Output Section provides information the system needs to control transmission and handling of data between I/O devices and the object program. In this section, the programmer identifies and assigns files for the program to use and assigns them to particular devices. During subsequent program operations, these files are used with the device types specified in this section.

For a detailed discussion of the Environment Division, refer to Chapter 10.

1.2.3 Data Division

The Data Division contains the names and format descriptions of all data to be used in the program. The Data Divison, like the Environment Division, is composed of sections.

The File Section describes the format of each file and each record within each file the program uses. Level numbers delineate the hieraráchy of elements within a file.

The Working-Storage Section describes all data items that do not exist as part of a file, but are used by the object program for specific program processing. Working-Storage records can also be described in terms of a hierarchy of levels.

The Linkage Section is required in a program invoked by another program. A CALL...USING statement in the Procedure Division of the calling program accomplishes the call. This section describes the data that the called program receives from the calling program.

For a detailed discussion of the Data Division, refer to Chapter 11.

1.2.4 Procedure Division

The Procedure Division of a COBOL program controls the processing of data. The Procedure Division consists of two main sections: an optional Declaratives Section and a required section containing nondeclarative procedures. Each of these sections can contain other sections and paragraphs the programmer names. The Procedure Division statements that control data processing include input/output, arithmetic, decision-making, and program control statements. For a detailed discussion of the Procedure Division, refer to Chapter 12.

1.3 DISK FILE PROCESSING

VS COBOL supports three types of disk file organization: consecutive, relative and indexed. A consecutive organization allows a programmer access to that file's records in the same order that they are written. Thus, a request for record number 3 retrieves the third record written to the file. Records in an indexed file are accessed through the value of a field of the record called the "record key". Records in an indexed file can have both primary and alternate record keys. Thus, a request for the record whose primary key is 3 causes the retrieval of the record with that value in its primary key data field, no matter when the record was written to the file or in what order. In addition, programmers can access an alternate indexed file by referencing the file's primary or alternate keys.

Relative files consist of records uniquely identified by an integer value greater than zero which specifies the record's logical ordinal position in the file. This value is the relative record number and controls access to the record. Records can be accessed by sequential, random, or dynamic mode. The sequential mode allows the programmer to access records in the ascending order of the relative record numbers of all the records currently existing in the file. The random mode allows the programmer to access a record by placing its relative record number in a relative key data item, and dynamic access allows the programmer to change mode from sequential to random and back again.

The ORGANIZATION IS (SEQUENTIAL or INDEXED) clause of the FILE-CONTROL entry in the Environment Division specifies the file type within a COBOL program. For indexed files, the RECORD KEY IS clause must also be included; for alternate indexed files, the ALTERNATE RECORD KEY clause must be included as well.

Each of these file types can be accessed by the programmer in one of three ways: sequentially, randomly, or dynamically. Sequential access of a consecutive file means the programmer accesses the records in the order in which they were written. Random access of a consecutive file allows him to access records in any order by reference to a "relative record number" that represents the order in which a record was written to the file.

Sequential access of indexed files retrieves the records in the ascending order of their record key (primary or alternate) values. Using random access for indexed files, a programmer retrieves the desired record by placing the value of its primary or alternate record key in the data item defined in the RECORD KEY IS or ALTERNATE RECORD KEY clause.

For consecutive or indexed files, the dynamic access mode allows the programmer to employ both sequential and random access of the same file within one program.

VS COBOL allows three record formats for files: fixed-length, variable-length, and compressed. If a program specifies variable-length records, the number of characters in the records of the file may vary; if the program specifies fixed-length records, the number of characters must be the same. Compressed records can save space because characters that repeat three or more consecutive times are stored in two bytes, one for the character and one for the number of times it repeats. The RECORD CONTAINS clause of the File Description (FD) paragraph determines the record format for a file.

The following four Procedure Division verbs designate the Input/Output operations that can be performed by a program on disk files: READ, WRITE, REWRITE, and DELETE. A file's organization determines the kinds of operations that a program can perform on that file. Before these operations can take place, an OPEN statement must prepare the file for processing. The OPEN statement specifies the mode in which the file is opened. The operations a program can perform on a file are determined by the file's organization and the mode the program uses to open the file. These modes are: OUTPUT, for creating a file; INPUT, for reading from an existing file; I-O, for any Input/Output operation on an existing file; EXTEND, for writing new records to the end of an existing file; and SHARED, a VS COBOL extension that allows several users to update the same file concurrently.

VS COBOL offers a number of options for enhancing file processing efficiency. For consecutive files, the BUFFER SIZE clause of the FILE-CONTROL entry can increase the size of the buffers set aside for file processing. For indexed files, the RESERVE n AREAS clause of the FILE-CONTROL entry and the SAME AREA FOR statement of the I-O-CONTROL paragraph specify that more than one file is to share a buffer of a certain size; this is called buffer pooling. The VALUE OF DATA AREA and VALUE OF INDEX AREA clauses of the FD paragraph can reduce the number of operations required when records are added to an indexed file.

Additionally, VS COBOL allows a programmer to exclusively hold resources for update or retrieval. Holding resources is applicable to indexed and alternate indexed files only. Resources can be identified by the file name and by generic key. Generic key is the value of the first N characters of a record's primary key. Resource holding is a feature of both the DMS Sharing environment (discussed in Subsection 1.3.1) and the DMS/TX environment (discussed in Section 1.4).

For a detailed discussion of disk file handling, refer to Chapter 2.

1.3.1 DMS Sharing Environment

The Wang VS offers the DMS Sharing environment in which an application program can hold more than one record at one time. DMS Sharing was referred to as Advanced Sharing in prior versions of this manual. Many of the features of DMS Sharing are incorporated into the DMS/TX environment discussed in Section 1.4 and presented more fully in Chapter 3. A chart comparing the DMS Sharing and DMS/TX functions is also presented in Chapter 3.

Under DMS Sharing, holding requests are made in units called resources. Resources can be either a record, a generic range of keys for an indexed or alternate indexed file, or an entire file. Program initiated requests can be processed by the system either by using a pre-claim strategy, in which all resources are claimed at once, or a claim-as-needed strategy, in which resources are claimed as required by the application.

VS COBOL continues to support all aspects of the DMS Sharing environment. Discussions on specific functionality is presented in Section 2.6.

1.4 DMS/TX

DMS/TX is a transaction recovery system. An extension of DMS Sharing, DMS/TX provides multiple user sharing and rollback recovery of indexed data files processed in Record Access Method (RAM). Files used with DMS/TX are organized into a named set of indexed data files called a database. DMS/TX file updates performed by a VS COBOL program are grouped into units called transactions. A transaction is a related set of record updates that are posted as a group to preserve database consistency.

File sharing under DMS/TX is fully compatible with DMS Sharing. DMS/TX file sharing features include:

- Multiple users are allowed simultaneous access to the same files.
- Programs holding resources for update do so on a claim-as-needed basis.
- Any resource held for update by one task can be read without hold by another task. Any resource held for update cannot be held by any other task.
- More than one task can hold a resource for retrieval.
- Each task can exclusively hold multiple resources for the duration of a transaction.
- The system automatically releases all resources held by a task at the conclusion of a transaction.
- Any deadlock situation is automatically resolved by the system.

DMS/TX safeguards against damage caused by a program or system failure occuring during file updated through Transaction Rollback Recovery. Rollback recovery features include the following:

- Transactions are fully applied or not applied at all, i.e., rolled back.

- If a transaction is rolled back, all updates made to the data files are removed, returning each file to its previous consistent state.

- Consistency is maintained both within a file and between files whose updates must be coordinated.

- Rollback is automatically performed by the system when necessary and can be initiated as a program-invoked function.

For a detailed discussion on using DMS/TX in VS COBOL, refer to Chapter 3.

1.5 WORKSTATION FILE PROCESSING

Because the VS is an interactive system, the user can communicate directly with the system through the workstation, responding to prompts from the system or querying it and receiving an immediate reply. Wang has implemented extensions to COBOL that facilitate, and take advantage of, these interactive capabilities. These extensions allow the user to format the contents of the workstation screen, to move data to and from the system and to the screen, and to determine display characteristics (uppercase or lowercase, alphanumeric or numeric, bright or dim, modifiable or protected, underlined or not underlined, blinking or not blinking, blank or not blank).

The VS COBOL extensions provide two approaches to interactive data handling. The user can combine these approaches within the same program. Both approaches treat the screen syntactically as if it were a file. Thus, a programmer assigns a file name for the screen in a SELECT clause. The programmer also assigns a device type of "DISPLAY" and includes a File Description entry for it in the Data Division.

The Procedure Division statement for the first method of interactive data handling is DISPLAY AND READ. In addition to moving information from internal storage to the workstation screen and vice versa, DISPLAY AND READ automatically performs operations such as setting default display characteristics, initializing fields, and validating data. In order for the programmer to use DISPLAY AND READ, he must describe the screen format with a Working-Storage entry, including a USAGE IS DISPLAY-WS clause.

The second method of interactive data handling is more complex. It uses REWRITE statements to move information to the screen, and READ statements to transfer information from the screen to storage. This method requires that the programmer write code to perform those operations automatically performed by the DISPLAY AND READ statement.

Both methods can use a number of VS COBOL extensions to control screen formatting and display characteristics. Each field displayed on the screen has a byte preceding it that contains its Field Attribute Character (FAC). A FAC is a hexadecimal numeral that represents a set of display characteristics (uppercase, numeric, blinking, and so on). In order to manipulate and test FACs within a program, data names can be assigned to them by the programmer in the FIGURATIVE-CONSTANTS paragraph of the Environment Division.

A 4-byte area in storage, called the "order area", exists for each workstation screen display. The order area controls such workstation features as keyboard locking (the cursor disappears and data cannot be entered by a user from the workstation), alarm sounding (if, for example, a user enters invalid data) and cursor positioning. As with FACs, data names associated with hexadecimal characters in the FIGURATIVE-CONSTANTS paragraph can reference the contents of the order area for modification and testing.

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

For a detailed discussion of workstation files, refer to Chapter 4.

1.6 PRINT FILE PROCESSING

Print files and several extensions of the WRITE statement control the content and format of printed output in VS COBOL. Designating the device type as "PRINTER" in the FILE-CONTROL entry creates print files. The length specified in the record description for this file is the length of the line to be printed.

The programmer specifies the number of lines the printer skips before or after writing a print file record by coding the BEFORE or AFTER ADVANCING clause of the WRITE statement with an integer, a data name having an integer value, or a data name for a hexadecimal character (defined in the FIGURATIVE-CONSTANTS paragraph). The BEFORE or AFTER ADVANCING clause also controls when the printer is to end one page and go to the next. User-defined figurative constants in the WRITE statement can control printer alarm sounding and the use of expanded print characters, functions that some printers support.

For a detailed discussion of print control, refer to Chapter 5.

1.7 TAPE FILE PROCESSING

The VS also supports magnetic tape files. Naming the device type "TAPE" in the SELECT statement identifies a file as a magnetic tape file. Consecutive file organization is the only kind available for tape files.

To make a tape file available to a COBOL program, the programmer must specify its physical location on the tape. To do this, the programmer can either reference labels that mark the beginning of the file on the tape or indicate the relative position of the file on the tape with a file number. The LABEL RECORDS ARE (STANDARD or OMITTED) clause of the FD paragraph indicates whether the file has labels.

The VALUE OF FILENAME, LIBRARY, AND VOLUME clauses of the FD paragraph reference tape labels. If relative position is used to locate the file, the VALUE OF POSITION clause must be coded.

For a detailed discussion of tape files, refer to Chapter 6.

CHAPTER 2
FILE ORGANIZATION AND ACCESS

2.1 INTRODUCTION

This chapter discusses the process of creating and maintaining files in VS COBOL. The discussion will focus on file organization (the physical structure of the file) and file access (the program-determined method of obtaining and storing records) for disk files. The following VS COBOL statements maintain records on files.

READ	Retrieves a record from the file.
WRITE	Stores a record into the file.
REWRITE	Replaces a record that has been previously READ, storing the modified record into the file.
DELETE	Removes a record from the file.
START	Positions the file so that subsequent READs can retrieve the desired group of records.

Each of these basic operations has many variations, depending on the file organization and the precise action desired.

2.1.1 Opening and Closing a File

Before operations can be performed on records in the file, the file must be prepared for processing. The OPEN statement, coded in the Procedure Division, will accomplish this. In addition to recording the fact that a file is open, the VS operating system also must know how records will be processed. This is accomplished by coding a modifier to the OPEN statement. This modifier is called the "open mode". Valid open modes are as follows:

OPEN Statement	Meaning
OPEN OUTPUT file-name	The file does not exist. It will be created; that is, space will be made available for it.
OPEN INPUT file-name	Records will be read from the file by one or more users, but no modifications will occur.
OPEN I-O file-name	Records will be modified on the file. The file is reserved for exclusive use by the program.
OPEN EXTEND file-name	The file already exists and will be prepared for writing records at the end of the file.
OPEN SHARED file-name	Records will be modified on the file, as in OPEN I-O. However, many users can modify different records concurrently. Records are held for modification by the program as requested.

To signal that the program will do no further I-O operations on the file, the program should close the file. Closing the file releases it from program control. To close a file, code CLOSE file-name in the Procedure Division.

Another use of closing a file is to allow the file to be reopened in another mode. For example a file being created must be opened in output mode. However, WRITE is the only operation allowed in output mode. To allow other operations, the file should be closed and reopened in I-O or shared mode. To create a file called FILE1 and then allow all operations on it, code the following statements.

```
OPEN OUTPUT FILE1.
CLOSE       FILE1.
OPEN I-O    FILE1.
```

2.1.2 File Organizations

VS COBOL supports three file organizations: consecutive, indexed and relative files.

Consecutive Files

Consecutive files consist of records that are stored on the file in the order they are written; a consecutive file is specified by coding ORGANIZATION IS SEQUENTIAL in the FILE-CONTROL entry for the file. A WRITE of a record to a consecutive file adds a record to the end of the file. Consecutive files are discussed in Section 2.3.

Consecutive files can be processed in one of three ways, depending on the ACCESS MODE IS clause in the FILE-CONTROL entry for the file: sequentially (ACCESS MODE IS SEQUENTIAL), randomly (ACCESS MODE IS RANDOM), or a combination of sequentially or randomly (ACCESS MODE IS DYNAMIC). Sequential access of a consecutive file is accessing records in the order they were written. The first record is read and processed, then the next record, and so on. Sequential access of a consecutive file is described in Section 2.3.1.

In random access of a consecutive file, a record is accessed by its "relative record number", which is an indicator of the ordinal position of the record within the file. To access the 15th record in the file, request relative record number 15 by moving 15 to the data name referenced in the RELATIVE KEY IS phrase. Random access of a consecutive file is described in Section 2.3.2.

In dynamic access of a consecutive file, records are accessed either in order (sequentially) or by relative record number (randomly). Dynamic access of a consecutive file is described in Section 2.3.3.

Indexed Files

Indexed files consist of records that are stored on the file according to a field in the record. For each record, a field is designated as containing a unique value that identifies the record; for example, an employee number field in an employee record. This field is called the "primary key". Records in an indexed file can be accessed either in primary key order (sequentially) or by a particular primary key value (randomly). Random access of an indexed file means that a record may be read without reading all the records preceding it. This is an efficient method of obtaining a record directly, since it is directly accessible by the primary key value. Storing records in indexed files thus provides added flexibility over sequential files (in which records must be read in order) for record access.

Indexed files can also be processed in one of three ways, depending on the ACCESS MODE IS clause in the FILE-CONTROL entry for the file: sequentially (ACCESS MODE IS SEQUENTIAL), randomly (ACCESS MODE IS RANDOM), or a combination of sequentially or randomly (ACCESS MODE IS DYNAMIC). Sequential access of an indexed file accesses the records in ascending primary key order. The record with the lowest primary key in the ASCII collating sequence is read and processed, then the next record, and so on. Before reading the record with EMPLOYEE-NUMBER = 12345, all records with EMPLOYEE-NUMBER less than 12345 must be read.

In random access of an indexed file, however, a record is accessed by its primary key value. This value is found in the data name referenced in the RECORD KEY IS clause. The RECORD KEY IS data name is a field in the FD entry for the file; its location in the record indicates the primary key for the record. To obtain the record having this primary key value, move the value to the RECORD KEY IS data name and issue the READ. Therefore, to read the record with EMPLOYEE-NUMBER = 12345, the record can be accessed without reading any other records in the file.

In dynamic access of an indexed file, records are accessed either in primary key order (sequentially) or by a primary key value (randomly). Indexed files are discussed in Section 2.4.

Alternate indexed files are extensions of indexed files to allow for access of a record along up to 16 alternate paths, or alternate keys. In addition to accessing the record along the primary key path (the standard indexed file capability) the record can be accessed along the other paths. The COBOL program issues a START along the desired path; subsequent READ instructions obtain records along that particular alternate path. The effect of alternate indexed file processing is that the file is presorted based upon as many as 17 fields. Alternate indexed files are discussed in Section 2.5. The use of relative files requires Release 6.20 or greater of the VS Operating System and any VS machine other than a VS50 or VS80.

Relative Files

Relative files consist of records uniquely identified by an integer value greater than zero which specifies the record's logical ordinal position in the file. A relative file is composed of a serial string of areas. Each area has a relative record number and is capable of holding a logical record. Records are stored and retrieved according to the relative record number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records are written in the first nine record areas.

Relative files can be accessed in one of three modes, sequential, random, or dynamic. In the sequential mode, records are accessed in the ascending order of the currently existing relative record numbers. The random access mode allows the programmer to control access to the file's records. A desired record is accessed by the programmer placing that record's relative record number in a relative key data item. Dynamic access allows the programmer to change mode from sequential to random and back again at will. Relative files are discussed in Section 2.6.

2.1.3 Record Types

VS COBOL supports three record types: fixed-length records, variable-length records, and compressed records. Most record types are allowed in all file organizations. Compressed records are not allowed for relative files. The specification of record type for a file is accomplished by variations on the RECORD CONTAINS clause in the FD entry for the file and is established when the file is created. The record type specification cannot be changed after the file is created.

Fixed-Length Records

Fixed-length records, the default, means that all of the records have the same length. Coding of the RECORD CONTAINS clause is unnecessary if fixed-length records are desired. However if you code the RECORD CONTAINS clause, the COBOL compiler will check the record length specified in the RECORD CONTAINS clause against the computed record size in the record description entry: if they disagree, the compiler will produce a warning diagnostic for information purposes only. This checking facility is particularly useful if the record structure is complex and verification of the record size is required.

To specify fixed-length records with record-length checking, for the file FILE1, code in the FILE SECTION as shown.

```
FILE SECTION.
FD  FILE1
    RECORD CONTAINS 100 CHARACTERS.
01  RECORD1                     PICTURE X(100).
```

The Procedure Division statement WRITE RECORD1 will write a 100-byte fixed-length record to the file FILE1.

Variable-Length Records

Variable-length records mean that the record length will vary. The size of the record area being written or rewritten determines the record size. Variable-length records are specified by the RECORD CONTAINS N TO M CHARACTERS clause in the FD entry for the file. M specifies the maximum record size, while N specifies the minimum record size. When a file is created, it contains a maximum record size; any record size equal to or less than the maximum record size can be written. Therefore in the variable-length records specification M cannot be greater than the record size specified when the file was created.

To specify variable-length records for the file FILE1, code in the FILE SECTION as shown.

```
FILE SECTION.
FD  FILE1
    RECORD CONTAINS 50 TO 100 CHARACTERS.
01  FIFTY-BYTE-RECORD           PICTURE X(50).
01  ONEHUNDRED-BYTE-RECORD      PICTURE X(100).
```

To write a 50-byte record to FILE1, code in the Procedure Division as shown.

```
WRITE FIFTY-BYTE-RECORD.
```

To write a 100-byte record to FILE1, code in the Procedure Division as shown.

```
WRITE ONEHUNDRED-BYTE-RECORD.
```

To write a record of any other length (up to 100 characters) add a record description entry for a record of the desired length, and issue the WRITE for that record description entry.

Compressed Records

Compressed records are stored on disk in a manner that, for most files, will economize space. Compressed records are stored so that:

- If a character repeats for 3 or more times (for up to 128 times), the character is stored in 1 byte and another byte is used to store the number of times the character repeats.

- Every nonrepeating sequence of up to 128 characters requires an extra byte to store the number of nonrepeating characters.

Compression is useful when many characters repeat in the record. For example, COBOL source files contain many repeating spaces. For such a file, compression can save much disk space -- 50 per cent and more in many cases.

To specify a compressed file, code in the FILE SECTION as shown.

```
RECORD CONTAINS 100 COMPRESSED CHARACTERS.
```

The COBOL reserved word COMPRESSED defines a file with compressed records. When the file is opened in output mode (OPEN OUTPUT file-name), it will be created as a compressed file. Even though the RECORD CONTAINS clause in the previous example does not indicate variable-length records, the file is also created as a variable-length record file because all files with compressed records are variable-length record files. If the file is opened in any other mode, the COMPRESSED option may be omitted, since the file is already defined to have compressed records and this fact is recorded in the file label.

2.2 THE COBOL FILE PROCESSING ENVIRONMENT

Figure 2-1 is a complete COBOL program that creates an indexed file (with no records) on disk.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  CRE8FILE.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT FILE1
000007         ASSIGN TO "EXTFILE", "DISK",
000008         ORGANIZATION    IS INDEXED
000009         ACCESS MODE     IS DYNAMIC
000010         RECORD KEY      IS THE-PRIMARY-KEY.
000011 DATA DIVISION.
000012 FILE SECTION.
000013 FD  FILE1
000014     LABEL RECORDS ARE STANDARD.
000015 01  FILE1-RECORD.
000016     03  THE-PRIMARY-KEY            PIC X(10).
000017     03  FILLER                     PIC X(70).
000018 PROCEDURE DIVISION.
000019 START-PROGRAM.
000020     OPEN OUTPUT FILE1.
000021     CLOSE FILE1.
000022     STOP RUN.
```

Figure 2-1. File Creation

Files are defined in COBOL by the FILE-CONTROL entries in the Environment Division and the FD entries in the File Section of the Data Division. Each clause in the FILE-CONTROL entry has a default; if a file is to use the default, coding is not necessary. Only those clauses in the FILE-CONTROL and FD necessary for the file need to be coded.

2.2.1 FILE-CONTROL Clauses Required for File Processing

In Figure 2-1, the FILE-CONTROL entry is coded on Lines 6 - 10. Referring to this FILE-CONTROL entry, the clauses for defining a VS COBOL file provide the following information.

- The logical file-name (the file name as known to the COBOL program), coded on Line 6. The logical file name is FILE1. Subsequent statements referring to the file in the program use the logical file name.

- The parameter reference name (PRNAME) of the file, coded on Line 7. This is the external name of the file. The PRNAME is EXTFILE. The parameter reference name is used by the VS as an external tag when the file is opened. The COBOL program may not have specified all the information necessary to open the file; in that case, the VS will display a message at open time requesting additional information. The VS uses the parameter reference name to display a screen (the OPEN GETPARM) requesting the additional information.

- The device on which this file resides, coded on Line 7. The device is DISK. VS COBOL supports files on disk (DISK), tape (TAPE), printer (PRINTER), and the workstation (DISPLAY). Specification of device type is optional, with DISK as the default.

- The file organization, coded on Line 8. The organization is INDEXED. VS COBOL supports consecutive (ORGANIZATION IS SEQUENTIAL), relative, and indexed files. For consecutive files, records are written at the end of the file. For relative files, records are written according to a relative record number. For indexed files, records are written in the order determined by the value in the primary key field. FILE1 is an indexed file.

- The mode by which records can be accessed, coded on Line 9. The access mode is DYNAMIC. VS COBOL supports access of records in the order that they were written (ACCESS MODE IS SEQUENTIAL), random access by a value in a key field (ACCESS MODE IS RANDOM), or a combination of sequential and random access (ACCESS MODE IS DYNAMIC).

- For indexed files, the primary key field, coded on Line 10. The primary key is THE-PRIMARY-KEY. When a record of an indexed file is written, the value in the primary key field determines the placement of the record in the file. THE-PRIMARY-KEY is a field in the record; it is defined in the record description entry as part of the FD.

2.2.2 FD Information Required for File Processing

Referring to Figure 2-1, the FD (File Description) information required for COBOL file processing includes:

1. The logical file name (FILE1), coded on Line 13. This is the same name as the logical file name specified in Line 6 of the FILE-CONTROL entry. For every file defined in a FILE-CONTROL entry, there must exist a corresponding FD entry; for every FD entry, there must be a corresponding FILE-CONTROL entry. This allows the COBOL compiler to relate information specified in the FD entries to information specified for the FILE-CONTROL entry.

2. The labels attached to the corresponding physical file (LABEL RECORDS ARE STANDARD), coded on Line 14. The LABEL RECORDS clause is primarily used for tape files to indicate whether ANSI, IBM, or nonlabelled tapes are being used. The LABEL RECORDS clause is optional for disk files; LABEL RECORDS ARE STANDARD is the default.

3. The record description entry, coded on Lines 15 - 17. The record description entry describes the record and fields associated with the record. The record description entry starts with the record name FILE1-RECORD), coded on Line 15. Fields subordinate to the record description entry are identified with level numbers greater than 01. The record deşcription entry can be implicitly redefined in a subsequent record description entry.

For indexed files, one of the fields in the record must be specified as a record key. The RECORD KEY clause, on Line 10 of the FILE-CONTROL entry, identifies a field to be used as the primary key in the record. The field THE-PRIMARY-KEY has been defined in the FILE-CONTROL entry as the primary key for the file. The field THE-PRIMARY-KEY, specified on Line 16, is located at the first 10 bytes of the record. If a file is indexed, it must have a RECORD KEY clause, and the data name specified as the record key must be defined in the record description entry.

FILE1 is a file containing fixed-length records. The record size for FILE1 is computed by the COBOL compiler by adding the sizes of each of the fields in the record description entry. In this case, the record size is 80 bytes (10 bytes for THE-PRIMARY-KEY added to 70 bytes for FILLER).

2.2.3 Creating the File

In the Procedure Division, FILE1 is created by opening the file in output mode, thereby creating a file label. This is done by successful execution of the OPEN statement on Line 20. Output mode, specified by the OUTPUT modifier of the OPEN statement, implies that the file does not exist. OPEN OUTPUT FILE1 issues an OPEN GETPARM, with a parameter reference name of EXTFILE, requesting the file name, library name, and volume name, as well as the number of records the file is to have. This information is used by the VS operating system to allocate space on the disk. After you have entered the number of records, the file label will be created. The CLOSE statement on Line 21 closes the file, updating the file label. The STOP RUN statement on Line 22 terminates the program.

The OPEN GETPARM screen can be suppressed by either running the program from a VS Procedure (refer to the VS Procedure Language Reference manual for information on writing VS Procedures) or by supplying the file information within the program itself. The program supplies the necessary information to open the file, by coding the NODISPLAY option in the FILE-CONTROL entry for the file, or by coding VALUE OF clauses in the FD for the file.

2.2.4 Using VALUE OF Clauses to Specify File Location

A VS disk file must be uniquely defined at program execution time to the VS operating system. A unique specification of the location of a VS disk file is obtained by specifying three location attributes: file, library, and volume. A disk volume may contain many libraries, which in turn may contain many files. A VS disk file is uniquely defined by specifying a file name, a library name, and a volume name.

If the VS COBOL program does not specify the file location when the file is opened, OPEN will display a message (the OPEN GETPARM) requesting this information. However, VS COBOL provides the facility -- through VALUE OF clauses in the FD for the file and the NODISPLAY option of the FILE-CONTROL entry -- for the program to fill in file location information so that the OPEN GETPARM does not appear. The VALUE OF FILENAME, VALUE OF LIBRARY, and VALUE OF VOLUME clauses allow specification of a data name or a literal for the name of the file, library, and volume respectively. For example, to define the location attributes of the file PAYROLL in the library EMPLIB on the volume SYSTEM for a file with an FD name of PAYFILE, code the FD clauses as shown.

```
    FD  PAYFILE
        VALUE OF FILENAME IS "PAYROLL"
                 LIBRARY  IS "EMPLIB"
                 VOLUME   IS "SYSTEM"
        LABEL RECORDS    ARE STANDARD.
```

When the file is opened, the VS operating system will attempt to locate the file PAYROLL in the library EMPLIB on the volume SYSTEM. If the file has the NODISPLAY option in its FILE-CONTROL entry, only in the case of an error (for example, the disk volume is not mounted) will the OPEN GETPARM (requesting respecification of file parameters) appear.

The VALUE OF clauses will also accept a data name. Specifying a data name in the VALUE OF clauses may be necessary if the actual file, library, and volume names will be determined in the Procedure Division by moving appropriate values in the data-name specified in the VALUE OF clauses. To specify a file PAYROLL in EMPLIB on SYSTEM by this method, code the FD clauses as shown.

```
    FD  EMPFILE
        VALUE OF FILENAME IS FILE-NAME
                 LIBRARY  IS LIBRARY-NAME
                 VOLUME   IS VOLUME-NAME
        LABEL RECORDS    ARE STANDARD.
```

In Working-Storage, code the entries as shown.

```
    WORKING-STORAGE SECTION.
    77  FILE-NAME           PICTURE IS X(8)     VALUE IS "PAYROLL".
    77  LIBRARY-NAME        PICTURE IS X(8)     VALUE IS "EMPLIB".
    77  VOLUME-NAME         PICTURE IS X(6)     VALUE IS "SYSTEM".
```

If any file-related information is changed when the file is opened, the correct information is stored in the VALUE OF data names.

2.2.5 Specifying Initial Space Allocation

When a file is created, the VS operating system requires specification of the number of records to be written to the file. This number is used to allocate initial disk space, or "primary extent" for the file. If the file fills up the primary extent with records, the VS operating system will automatically allocate another disk area, or "secondary extent", for the file. If additional disk areas are needed, up to 12 additional secondary extents are automatically allocated as required. Therefore, the number of records actually on the file can exceed the space for the number of records requested at file creation time.

The VALUE OF SPACE clause is used to specify the number of records for initial space allocation. VALUE OF SPACE requires a data name. Therefore, to request space for 100 records for a file, code in the FD as shown.

```
        VALUE OF SPACE IS SPACE-PARAMETER
```

In Working-Storage, code as shown.

```
     77  SPACE-PARAMETER         PICTURE IS 9(3)     VALUE IS 100.
```

If the file already exists, SPACE-PARAMETER is set to the actual number of records in the file. This information is useful for processing in which the count of records in the file is important. SPACE-PARAMETER should be initialized to a value, even though its value is replaced by the acutal record count. If the VALUE OF SPACE data item is not initialized, results are unpredictable.

2.2.6 Suppressing OPEN Messages

When a file is opened, the default action is for the OPEN statement to produce a message (the OPEN GETPARM) requesting verification of the accuracy of file parameters. In production environments this is frequently undesirable because it interferes with smooth job execution and permits undesired operator modifications. If the program is to be run as a background job, the OPEN GETPARM must be suppressed because a background job cancels if it encounters an OPEN GETPARM that cannot be satisfied. To suppress OPEN messages (except for error conditions), do one of the following.

1. Write a Procedure to run the program, coding an ENTER statement for each PRNAME. Under normal conditions, this suppresses the OPEN GETPARM. Refer to the VS Procedure Language Reference manual for information on the ENTER statement.

2. Code the NODISPLAY option for the file in the FILE-CONTROL entry, and the relevant VALUE OF clauses in the FD. The VALUE OF FILENAME, VALUE OF LIBRARY, VALUE OF VOLUME, and VALUE OF SPACE clauses, in conjunction with the NODISPLAY option of the FILE-CONTROL entry, will fill in the required information for OPEN. If the information is correct and OPEN can successfully open the file, no message will be displayed.

2.3 CONSECUTIVE FILE PROCESSING IN COBOL

2.3.1 Sequential Access of a Consecutive File in COBOL

Figure 2-2 is a complete COBOL program that creates a consecutive file and processes it sequentially.

The FILE-CONTROL entry for the consecutive file CONSEC is specified on Lines 6 - 9. If all of the information required to open the file successfully is specified, the OPEN GETPARM will not display when the file is opened because the NODISPLAY option is specified on Line 7. Consecutive file organization is defined by specifying ORGANIZATION IS SEQUENTIAL on Line 8. Sequential access -- reading records in the order in which they were written -- is defined by specifying ACCESS MODE IS SEQUENTIAL on Line 9.

The FD entry for CONSEC is specified on Lines 12 - 20. The contents of the data names referenced in the VALUE OF clauses is used to specify information required to open the file successfully. The VALUE OF FILENAME clause on Line 14 specifies that the contents of the data name FILE-NAME is the name of the file on the disk or other external medium. FILE-NAME, defined on Line 22, has a value of "CONSEC". Therefore, the external file name for CONSEC is "CONSEC".

The VALUE OF LIBRARY clause on Line 15 specifies that the contents of the data name DATA-LIBRARY contains the external library name for the file. DATA-LIBRARY, defined on Line 23, has a value of "DATA". Therefore, the external library name for CONSEC is "DATA".

The VALUE OF VOLUME clause on Line 16 specifies that the contents of the data name DATA-VOLUME contains the external volume name for the file. DATA-VOLUME, defined on Line 24, has a value of "SYSTEM". Therefore, the external volume name for CONSEC is "SYSTEM".

The VALUE OF SPACE clause on Line 17 specifies that the contents of the data name SPACE-PARAMETER contains the number of records for initial space allocation for the file. SPACE-PARAMETER, defined on Line 25, has a value of 3. Therefore, CONSEC will be created with an initial space allocation of 3 records.

The RECORD CONTAINS 1 TO 100 COMPRESSED CHARACTERS clause on Line 18 specifies that when CONSEC is subsequently opened in output mode, it will be created as a file with variable-length, compressed records. The phrase "1 TO 100" specifies variable-length records, with maximum record size of 100 bytes; the word "COMPRESSED" specifies compressed records.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  CONSEC.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT CONSEC
000007         ASSIGN TO "CONSEC",   "DISK",     NODISPLAY,
000008         ORGANIZATION    IS SEQUENTIAL
000009         ACCESS MODE     IS SEQUENTIAL.
000010 DATA DIVISION.
000011 FILE SECTION.
000012 FD  CONSEC
000013     LABEL RECORDS ARE STANDARD
000014     VALUE OF FILENAME IS FILE-NAME
000015     VALUE OF LIBRARY  IS DATA-LIBRARY
000016     VALUE OF VOLUME   IS DATA-VOLUME
000017     VALUE OF SPACE    IS SPACE-PARAMETER
000018     RECORD CONTAINS 1 TO  100 COMPRESSED CHARACTERS.
000019 01  ONEHUNDRED-BYTE-RECORD         PIC X(100).
000020 01  FIFTY-BYTE-RECORD              PIC X(50).
000021 WORKING-STORAGE SECTION.
000022 77  FILE-NAME                      PIC X(8)  VALUE "CONSEC".
000023 77  DATA-LIBRARY                   PIC X(8)  VALUE "DATA".
000024 77  DATA-VOLUME                    PIC X(6)  VALUE "SYSTEM".
000025 77  SPACE-PARAMETER    COMP        PIC S9(3) VALUE 3.
000026 PROCEDURE DIVISION.
000027 CREATE-CONSECUTIVE-FILE.
000028     OPEN OUTPUT CONSEC.
000029     MOVE "THIS IS A 100 BYTE RECORD" TO ONEHUNDRED-BYTE-RECORD.
000030     WRITE ONEHUNDRED-BYTE-RECORD.
000031     MOVE "THIS IS A 50 BYTE RECORD" TO FIFTY-BYTE-RECORD.
000032     WRITE FIFTY-BYTE-RECORD.
000033     CLOSE CONSEC.
000034 ADD-TO-CONSECUTIVE-FILE.
000035     OPEN EXTEND CONSEC.
000036     MOVE "THIS IS A 100 BYTE RECORD ADDED IN EXTEND MODE" TO
000037         ONEHUNDRED-BYTE-RECORD.
000038     WRITE ONEHUNDRED-BYTE-RECORD.
000039     CLOSE CONSEC.
000040     OPEN INPUT CONSEC.
000041 CONSECUTIVE-FILE-READS.
000042     READ CONSEC NEXT AT END CLOSE CONSEC  STOP RUN.
000043     DISPLAY ONEHUNDRED-BYTE-RECORD.
000044     GO TO CONSECUTIVE-FILE-READS.
```

Figure 2-2. Sequential Access of a Consecutive File

Two record description entries are specified: an entry for a 100-byte record called ONEHUNDRED-BYTE-RECORD (coded on Line 19), and another entry for a 50-byte record called FIFTY-BYTE-RECORD (coded on Line 20). When a record is written to CONSEC using the WRITE statement, one of these record description entries will be specified. If the WRITE is specified for ONEHUNDRED-BYTE-RECORD, a 100-byte record is written; if the WRITE is specified for FIFTY-BYTE-RECORD, a 50-byte record is written.

In the Procedure Division, the program creates a file with two records (one record 100 bytes long, the other 50 bytes long). Then the file is opened in extend mode and a record is added to the end of the file. Finally, the file is opened in input mode and the three records written to the file are read. A file processed by ACCESS IS SEQUENTIAL can be opened in the following modes.

- Output mode, implying that the file does not exist and is to be created. In the paragraph CREATE-CONSECUTIVE-FILE (Lines 27 - 33), the file CONSEC is opened in output mode, two records are written to the file, and the file is closed. The only valid operation on a file opened in output mode is WRITE. On Line 30, a 100-byte record with the value THIS IS A 100 BYTE RECORD is written; on Line 32, a 50-byte record with the value "THIS IS A 50 BYTE RECORD" is written. The file is closed on Line 33.

- Extend mode, implying that the file does exist and records are to be appended to it. A file cannot be opened in extend mode unless it already exists. In the paragraph ADD-TO-CONSECUTIVE-FILE (Lines 34 - 40) the file CONSEC is opened in extend mode, one record is written to the file, the file is closed and reopened in input mode. The only valid operation on a file opened in extend mode is WRITE. CONSEC is opened in extend mode on Line 35, a 100-byte record with a value of "THIS IS A 100 BYTE RECORD ADDED IN EXTEND MODE" is added to the file on Line 38, the file is closed on Line 39, and the file is reopened in input mode on Line 40.

- Input mode, implying that records are to be read from the file, and that no updates are to be done to it. The paragraph CONSECUTIVE-FILE-READS (Lines 41 - 44) reads the three records written to the file in the order in which they were written. The largest record description entry is displayed if a record was read successfully. This guarantees that the largest record will be displayed. When the end-of-file condition is encountered, the AT END exit of the READ statement (coded on Line 42) is executed; here, CONSEC will be closed and the program will terminate normally.

2.3.2 Random Access of a Consecutive File in COBOL

Figure 2-3 is a complete COBOL program illustrating the facilities of random access of a consecutive file.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  RANDOM.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT RNDFILE
000007         ASSIGN TO "RNDFILE", "DISK",     NODISPLAY,
000008         ORGANIZATION    IS SEQUENTIAL
000009         ACCESS MODE     IS RANDOM
000010         RELATIVE KEY    IS THE-RELATIVE-KEY.
000011 DATA DIVISION.
000012 FILE SECTION.
000013 FD  RNDFILE
000014     LABEL RECORDS    ARE STANDARD.
000015 01  RANDOM-FILE-RECORD-AREA         PIC S9(5)  COMPUTATIONAL.
000016 WORKING-STORAGE SECTION.
000017 01  THE-RELATIVE-KEY                PIC 9(5).
000018 PROCEDURE DIVISION.
000019 GET-THE-FIFTH-RECORD.
000020     OPEN INPUT RNDFILE.
000021     MOVE 5 TO THE-RELATIVE-KEY.
000022     READ RNDFILE INVALID KEY DISPLAY "RECORD NOT FOUND".
000023     CLOSE RNDFILE.
000024 UPDATE-RECORD-NUMBER-3.
000025     OPEN I-O RNDFILE.
000026     MOVE 3 TO THE-RELATIVE-KEY.
000027     READ RNDFILE WITH HOLD INVALID KEY
000028                               DISPLAY "RECORD NOT FOUND".
000029     MOVE 55 TO RANDOM-FILE-RECORD-AREA.
000030     REWRITE RANDOM-FILE-RECORD-AREA.
000031     CLOSE RNDFILE.
000032     STOP RUN.
```

Figure 2-3. Random Access of a Consecutive File

The FILE-CONTROL entry for the file RNDFILE is coded on Lines 6 - 10. RNDFILE is specified as a consecutive file by the ORGANIZATION IS SEQUENTIAL clause coded on Line 8 and is specified as being accessed randomly by the ACCESS MODE IS RANDOM clause coded on Line 9. The relative record number -- the number corresponding to the order of the record on the file -- is to be found in the data name THE-RELATIVE-KEY, as specified by the RELATIVE KEY IS THE-RELATIVE-KEY phrase on Line 10. THE-RELATIVE-KEY is defined on Line 17 of the program as a numeric field in Working-Storage.

The FD for RNDFILE, coded on Lines 13 - 14, specifies a record area containing one packed decimal field of three bytes.

In the Procedure Division, the program attempts to get the fifth record on RNDFILE. The program then attempts to update the third record to contain a value of 55. A file processed by ACCESS IS RANDOM can be opened in the following modes.

- Input mode. Records are read from the file by relative record number. In the paragraph GET-THE-FIFTH-RECORD (Lines 19 - 23), an attempt is made to read the fifth record on the file by moving the value 5 to THE-RELATIVE-KEY (the data name specified in the RELATIVE KEY IS phrase of the FILE-CONTROL entry) on Line 21, and then issuing the READ on Line 22. If the file does not have a fifth record because there are less than five records on the file, the INVALID KEY exit is taken and the message RECORD NOT FOUND is displayed.

- I-O mode. Records are read by relative record number, updated, and rewritten in place. In the paragraph UPDATE-RECORD-NUMBER-3 (Lines 24 - 32), an attempt is made to read the third record on the file by moving the value 3 to THE-RELATIVE-KEY on Line 26 and the READ is issued on Lines 27 - 28. The WITH HOLD option of the READ indicates that the record is held for updating. The value 55 is moved to the record area, RANDOM-FILE-RECORD-AREA, on Line 29. The record is rewritten using the REWRITE on Line 30. Finally the file is closed and the program terminates.

Consecutive files processed using ACCESS MODE IS RANDOM must have fixed-length records. Files with variable-length or compressed records cannot be processed randomly.

2.3.3 Dynamic Access of a Consecutive File in COBOL

Dynamic access of a consecutive file combines sequential and random access in one program. Dynamic access is indicated by ACCESS MODE IS DYNAMIC in the file's FILE-CONTROL entry.

A modifier has been added to the READ statement for dynamic access to indicate whether sequential or random reads of the records is desired. A READ statement without the NEXT modifier implies a random read, while a READ statement with the NEXT modifier implies a sequential read. The program in Figure 2-4 illustrates the diffences in the coding of the READ statement for random and sequential reads.

Dynamic access is specified on Line 9 by coding ACCESS MODE IS DYNAMIC in the FILE-CONTROL entry. Dynamic access supports all of the features of consecutive and random access, with the addition that records can be read either randomly or sequentially.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  DYNAMIC.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT DYNFILE
000007         ASSIGN TO "DYNFILE", "DISK",     NODISPLAY,
000008         ORGANIZATION    IS SEQUENTIAL
000009         ACCESS MODE     IS DYNAMIC
000010         RELATIVE KEY    IS THE-RELATIVE-KEY.
000011 DATA DIVISION.
000012 FILE SECTION.
000013 FD  DYNFILE
000014     LABEL RECORDS    ARE STANDARD.
000015 01  RANDOM-FILE-RECORD-AREA         PIC S9(5)  COMPUTATIONAL.
000016 WORKING-STORAGE SECTION.
000017 01  THE-RELATIVE-KEY                PIC 9(5).
000018 PROCEDURE DIVISION.
000019 GET-FIFTH-RECORD.
000020     OPEN INPUT DYNFILE.
000021     MOVE 5 TO THE-RELATIVE-KEY.
000022     READ DYNFILE INVALID KEY DISPLAY "RECORD NOT FOUND.".
000023 READ-THE-SIXTH-RECORD.
000024     READ DYNFILE NEXT AT END DISPLAY "END OF FILE REACHED.".
000025     DISPLAY RANDOM-FILE-RECORD-AREA.
000026     CLOSE DYNFILE.
000027     STOP RUN.
```

Figure 2-4. Dynamic Access of a Consecutive File

In the paragraph GET-FIFTH-RECORD (Lines 19 - 22), an attempt is made to read the fifth record of the file by moving the value 5 to the RELATIVE KEY IS data name (THE-RELATIVE-KEY) on Line 21 and issuing the READ on Line 22. This sequence of operations is equivalent to a READ using ACCESS MODE IS RANDOM. Refer to the program illustrated in Figure 2-3, which contains a similar attempt to read the fifth record of a consecutive file.

In the paragraph READ-THE-SIXTH-RECORD (Lines 23 - 27), an attempt is made to read the sixth record of the file. This attempt is made by executing the READ NEXT statement on Line 24. If the attempt to get the fifth record in the paragraph GET-FIFTH-RECORD is successful, an indicator recording that fact is established. This indicator is known as the "current record pointer". A READ NEXT issued after establishment of the current record pointer will attempt to read the next record in the file -- in this case, the sixth record. If either: (a) the attempt to get the fifth record had failed, thereby failing to establish a current record pointer; or (b) exactly 5 records exist on the file, the execution of the READ NEXT statement on Line 24 would invoke the AT END exit.

2.4 INDEXED FILE PROCESSING IN COBOL

Indexed files contain records that can be accessed by referring to the contents of a field in the file. This field is known as the "primary key". Since the contents of the primary key field uniquely identify a particular record, the primary key value must not duplicate a primary key value for any other record; an attempt to write a record having a duplicate primary key value will produce an error.

To specify an indexed file in COBOL, code the ORGANIZATION IS INDEXED clause in the FILE-CONTROL entry. The primary key field is identified by the RECORD KEY IS data name in the FILE-CONTROL entry. The RECORD KEY is data name must be a field in the record as defined in the File Section. All COBOL access modes (SEQUENTIAL, RANDOM, and DYNAMIC) are permitted for indexed files. COBOL supports all indexed file I-O operations (READ, WRITE, REWRITE, DELETE, and START) on all record formats.

Figure 2-5 is a complete COBOL program that illustrates processing of an indexed file. The program creates an indexed file containing address records for the following two hypothetical employees.

NUMBER	NAME	ADDRESS	CITY	STATE
1	William Shakespeare	555 Madison Ave.	New York	NY
2	Christopher Marlowe	555 Madison Ave.	New York	NY

The FILE-CONTROL entry for EMPLOYEE-ADDRESS-FILE, the file containing these two records, is coded on Lines 6 - 10. EMPLOYEE-ADDRESS-FILE is an indexed file because its FILE-CONTROL entry contains the ORGANIZATION IS INDEXED clause, coded on Line 8. The primary key, EMPLOYEE-NUMBER, is specified in the RECORD KEY is clause on Line 10.

EMPLOYEE-NUMBER is defined in the record description entry for EMPLOYEE-ADDRESS-FILE on Line 16 as the first field in the record. The Procedure Division paragraph CREATE-RECORDS-FOR-2-EMPLOYEES, coded on Lines 22 - 36, creates EMPLOYEE-ADDRESS-FILE and writes an address record for EMPLOYEE-NUMBER 1 (William Shakespeare), and for EMPLOYEE-NUMBER 2 (Christopher Marlowe).

In the paragraph UPDATE-EMPLOYEE-2, coded on Lines 37 - 42, assume that EMPLOYEE-NUMBER 2 (Christopher Marlowe), has moved from 555 Madison Avenue to 508 West 85th Street and that his address record is to be updated to reflect the new address. NAME-AND-ADDRESS-FILE is opened in I-O (or update) mode on Line 38. To access the record for EMPLOYEE-NUMBER 2, the contents of the RECORD KEY IS data name (EMPLOYEE-NUMBER) must be the correct employee number, which is 2. This initialization of EMPLOYEE-NUMBER to Christopher Marlowe's employee number is accomplished by successful execution of the MOVE statement on Line 39. The record of EMPLOYEE-NUMBER 2, with the old address information, is read by successful execution of the READ statement on Line 40. Since the access mode is dynamic, this READ statement is a random read by the record key. The WITH HOLD option signifies that the record will be either modified or deleted. The record is modified by moving the new address (585 West 85th Street) to the EMPLOYEE-ADDRESS field on Line 41, and the record is updated using the REWRITE statement on Line 42.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  INDEXED.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT EMPLOYEE-ADDRESS-FILE
000007         ASSIGN TO "NAMEADDR", "DISK",    NODISPLAY,
000008         ORGANIZATION    IS INDEXED
000009         ACCESS MODE     IS DYNAMIC
000010         RECORD KEY      IS EMPLOYEE-NUMBER.
000011 DATA DIVISION.
000012 FILE SECTION.
000013 FD  EMPLOYEE-ADDRESS-FILE
000014     LABEL RECORDS ARE STANDARD.
000015 01  EMPLOYEE-ADDRESS-RECORD.
000016     03  EMPLOYEE-NUMBER            PIC S9(5)      COMPUTATIONAL.
000017     03  EMPLOYEE-NAME              PIC X(20).
000018     03  EMPLOYEE-ADDRESS           PIC X(20).
000019     03  EMPLOYEE-CITY              PIC X(20).
000020     03  EMPLOYEE-STATE             PIC X(2).
000021 PROCEDURE DIVISION.
000022 CREATE-RECORDS-FOR-2-EMPLOYEES.
000023     OPEN OUTPUT EMPLOYEE-ADDRESS-FILE.
000024     MOVE 1                      TO EMPLOYEE-NUMBER.
000025     MOVE "WILLIAM SHAKESPEARE"  TO EMPLOYEE-NAME.
000026     MOVE "555 MADISON AVENUE"   TO EMPLOYEE-ADDRESS.
000027     MOVE "NEW YORK"             TO EMPLOYEE-CITY.
000028     MOVE "NY"                   TO EMPLOYEE-STATE.
000029     WRITE EMPLOYEE-ADDRESS-RECORD.
000030     ADD  1                      TO EMPLOYEE-NUMBER.
000031     MOVE "CHRISTOPHER MARLOWE"  TO EMPLOYEE-NAME.
000032     MOVE "555 MADISON AVENUE"   TO EMPLOYEE-ADDRESS.
000033     MOVE "NEW YORK"             TO EMPLOYEE-CITY.
000034     MOVE "NY"                   TO EMPLOYEE-STATE.
000035     WRITE EMPLOYEE-ADDRESS-RECORD.
000036     CLOSE EMPLOYEE-ADDRESS-FILE.
000037 UPDATE-EMPLOYEE-2.
000038     OPEN  I-O EMPLOYEE-ADDRESS-FILE.
000039     MOVE 2 TO EMPLOYEE-NUMBER.
000040     READ EMPLOYEE-ADDRESS-FILE WITH HOLD.
000041     MOVE "508 WEST 85 STREET" TO EMPLOYEE-ADDRESS.
000042     REWRITE EMPLOYEE-ADDRESS-RECORD.
000043 GET-THE-FIRST-RECORD.
000044     MOVE 0 TO EMPLOYEE-NUMBER.
000045     START EMPLOYEE-ADDRESS-FILE KEY > EMPLOYEE-NUMBER
000046             INVALID KEY DISPLAY "NO RECORDS IN FILE".
000047     READ EMPLOYEE-ADDRESS-FILE NEXT.
000048 DELETE-EMPLOYEE-1.
000049     MOVE 1 TO EMPLOYEE-NUMBER.
000050     READ EMPLOYEE-ADDRESS-FILE WITH HOLD.
000051     DELETE EMPLOYEE-ADDRESS-FILE.
000052     STOP RUN.
```

Figure 2-5. Indexed File Processing

The START statement enables logical positioning of a file by a particular primary key. After the successful execution of a START statement, a group of records can be read sequentially either by issuing READ NEXTs for files having ACCESS MODE IS DYNAMIC or READs for files having ACCESS MODE IS SEQUENTIAL. The START statement is useful in reading a group of records related by their primary key values; for example, if the city of residence is the first field of a primary key, a START, using the name of a particular city, positions the file so that subsequent sequential READs obtain employees residing in that particular city.

In Figure 2-5, the paragraph GET-THE-FIRST-RECORD, coded on Lines 43 - 47, gets the first record of EMPLOYEE-ADDRESS-FILE. On Line 43, zeroes are moved to EMPLOYEE-NUMBER, the RECORD KEY IS data name. Assuming no employees can have negative employee numbers, the START statement on Lines 45 - 46 will position the file to the first record, except for the case in which no records have been written to the file. If no records have been written to the file, the INVALID KEY exit would be taken, and the program would display the fact that no records had been written to the file. The START statement really means "position the file to the first record having EMPLOYEE-NUMBER greater than 0". The READ NEXT, on Line 47, will read the first record with EMPLOYEE-NUMBER greater than 0, which is the record of EMPLOYEE-NUMBER 1, or William Shakespeare's address record.

Assume that William Shakespeare has left the company and that his address record is to be deleted. The paragraph DELETE-EMPLOYEE-1, coded on Lines 48 - 52, deletes his record and ends the program. The record with EMPLOYEE-NUMBER 1 (that of William Shakespeare) is read randomly by moving 1 to the primary key data name EMPLOYEE-NUMBER on Line 49 and issuing the READ on Line 50. The WITH HOLD option of the READ indicates that the record will subsequently be either modified or deleted. In this case, the record is deleted after successful execution of the DELETE statement on Line 51. Finally, the program ends after successful execution of the STOP RUN statement on Line 52.

2.5 <u>ALTERNATE INDEXED FILE PROCESSING IN COBOL</u>

As an extension of indexed file support, VS COBOL supports processing of files by up to 16 alternate access paths, or alternate indices. This powerful facility is the equivalent of having a file presorted on up to 16 different fields.

Alternate indexed file processing is used in situations where access of a record by one of several key fields is desired. For example, a company maintains a file of employees. Each employee has a unique employee number. However, for reporting or updating purposes, it might be necessary to read the employee file by city of residence. If the CITY field is specified as an alternate index (or alternate access path), the file (by use of the START statement) can be positioned on the alternate path CITY. After a successful START, sequential reads will obtain records as if they had been sorted by city.

Further, if processing all employees who lived in the city of Lowell, the file could be positioned (using START), with the value LOWELL in the CITY field. Sequential reads would obtain employees living in Lowell, in employee-number order. The primary key values must be unique for every record (every employee must have a unique employee identification number). However, alternate key values may or may not be unique (more than one employee may live in Lowell): the WITH DUPLICATES phrase on the alternate key specification in the FILE-CONTROL entry for the file specifies the option. Thus, alternate indexed file processing can provide a quick method of referencing a particular record or group of records.

Figure 2-6 is a complete COBOL program illustrating the features of alternate indexed file support in COBOL.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  SHOWALTX.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT EMPLOYEE-ADDRESS-FILE
000007         ASSIGN TO "NAMEADDR", "DISK",     NODISPLAY,
000008         ORGANIZATION    IS INDEXED
000009         ACCESS MODE     IS DYNAMIC
000010         RECORD KEY      IS EMPLOYEE-NUMBER
000011         ALTERNATE RECORD KEY
000012         01 IS EMPLOYEE-NAME
000013         01 IS EMPLOYEE-NAME-1
000014         02 IS EMPLOYEE-CITY   WITH DUPLICATES
000015         02 IS EMPLOYEE-CITY-1 WITH DUPLICATES
000016         03 IS EMPLOYEE-STATE  WITH DUPLICATES.
000017 DATA DIVISION.
000018 FILE SECTION.
000019 FD  EMPLOYEE-ADDRESS-FILE
000020     LABEL RECORDS ARE STANDARD.
000021 01  EMPLOYEE-ADDRESS-RECORD.
000022     03  EMPLOYEE-NUMBER            PIC S9(5)      COMPUTATIONAL.
000023     03  EMPLOYEE-NAME              PIC X(20).
000024     03  EMPLOYEE-ADDRESS           PIC X(20).
000025     03  EMPLOYEE-CITY              PIC X(20).
000026     03  EMPLOYEE-STATE             PIC X(2).
000027 01  ALTERNATE-ADDRESS-RECORD.
000028     03  EMPLOYEE-NUMBER-1          PIC S9(5)      COMPUTATIONAL.
000029     03  EMPLOYEE-NAME-1            PIC X(20).
000030     03  EMPLOYEE-ADDRESS-1         PIC X(20).
000031     03  EMPLOYEE-CITY-1            PIC X(20).
000032     03  EMPLOYEE-STATE-1           PIC X(2).
```

Figure 2-6. Alternate Indexed File Processing

```
000033 PROCEDURE DIVISION.
000034 CREATE-RECORDS-FOR-2-EMPLOYEES.
000035     OPEN OUTPUT EMPLOYEE-ADDRESS-FILE.
000036     MOVE 1                        TO EMPLOYEE-NUMBER.
000037     MOVE "WILLIAM SHAKESPEARE"    TO EMPLOYEE-NAME.
000038     MOVE "1 ADMAN LANE"           TO EMPLOYEE-ADDRESS.
000039     MOVE "PALM SPRINGS"           TO EMPLOYEE-CITY.
000040     MOVE "CA"                     TO EMPLOYEE-STATE.
000041     WRITE EMPLOYEE-ADDRESS-RECORD.
000042     ADD  1                        TO EMPLOYEE-NUMBER-1.
000043     MOVE "CHRISTOPHER MARLOWE"    TO EMPLOYEE-NAME-1.
000044     MOVE "555 MADISON AVENUE"     TO EMPLOYEE-ADDRESS-1.
000045     MOVE "NEW YORK"               TO EMPLOYEE-CITY-1.
000046     MOVE "NY"                     TO EMPLOYEE-STATE-1.
000047     WRITE ALTERNATE-ADDRESS-RECORD.
000048     CLOSE EMPLOYEE-ADDRESS-FILE.
000049 PUT-MARLOWE-ON-STATE-PATH.
000050     OPEN  I-O EMPLOYEE-ADDRESS-FILE.
000051     MOVE 2 TO EMPLOYEE-NUMBER.
000052     READ EMPLOYEE-ADDRESS-FILE WITH HOLD.
000053     REWRITE EMPLOYEE-ADDRESS-RECORD.
000054 READ-LOWEST-CITY-RECORD.
000055     MOVE LOW-VALUES TO EMPLOYEE-CITY.
000056     START EMPLOYEE-ADDRESS-FILE KEY EMPLOYEE-CITY > EMPLOYEE-CITY
000057             INVALID KEY DISPLAY "NO RECORDS ON CITY PATH".
000058     READ EMPLOYEE-ADDRESS-FILE NEXT.
000059 DELETE-SHAKESPEARE.
000060     MOVE 1 TO EMPLOYEE-NUMBER.
000061     READ EMPLOYEE-ADDRESS-FILE WITH HOLD.
000062     DELETE EMPLOYEE-ADDRESS-FILE.
000063     STOP RUN.
```

Figure 2-6. Alternate Indexed File Procesing (continued)

The FILE-CONTROL entry for EMPLOYEE-ADDRESS-FILE, coded on Lines 6 - 16, specifies an alternate indexed file, because the ORGANIZATION IS INDEXED clause is coded on Line 8. The primary key, specified in the RECORD KEY IS clause on Line 10, is EMPLOYEE-NUMBER. As is the case for indexed files, the primary key value must be unique for each record. The clauses on Lines 11 - 16 specify the alternate paths. Each data name used as an alternate path is identified in the ALTERNATE RECORD KEY IS clause, which specifies:

- The ordinal number (from 1 to 16) of the path. This number identifies the order of the path.

- The data name associated with the path. The data name must be defined in the record description entry for the file.

- Whether duplicates are allowed. Optionally, duplicate values for alternate keys are allowed. If the WITH DUPLICATES phrase is coded, duplicate alternate key values are allowed; if the WITH DUPLICATES phrase is not coded, duplicate alternate key values are prohibited.

For example, on Line 12, ordinal path 1 (EMPLOYEE-NAME) does not allow duplicate values because the WITH DUPLICATES phrase is omitted; on Line 14, ordinal path 2 (EMPLOYEE-CITY) does allow duplicate values because the WITH DUPLICATES phrase is coded.

The record description entry for EMPLOYEE-ADDRESS-FILE is coded on Lines 19 - 32. Two records, EMPLOYEE-ADDRESS-RECORD (coded on Lines 22 - 26) and ALTERNATE-ADDRESS-RECORD (coded on Lines 27 - 32), are specified. <u>Three</u> alternate paths (EMPLOYEE-NAME, EMPLOYEE-CITY, and EMPLOYEE-STATE) are associated with the EMPLOYEE-ADDRESS-RECORD record. These paths were identified in the FILE-CONTROL entry as paths 1, 2, and 3. <u>Two</u> alternate paths (EMPLOYEE-NAME-1 and EMPLOYEE-CITY-1) are associated with the ALTERNATE-ADDRESS-RECORD record. When a record is written to the file, the record is also written along all the path(s) associated with that record.

The Procedure Division paragraph CREATE-RECORDS-FOR-2-EMPLOYEES, coded on Lines 34 - 48, creates EMPLOYEE-ADDRESS-FILE with the records for William Shakespeare and Christopher Marlowe. The Shakespeare record is written on Line 41 using EMPLOYEE-ADDRESS-RECORD -- a record containing the three alternate paths EMPLOYEE-NAME, EMPLOYEE-CITY, and EMPLOYEE-STATE. The Marlowe record is written on Line 47 using ALTERNATE-ADDRESS-RECORD -- a record containing the two alternate paths EMPLOYEE-NAME-1 and EMPLOYEE-CITY-1. The Marlowe record is <u>not</u> written along the state path.

The Marlowe record is written along the state path through successful execution of the paragraph PUT-MARLOWE-ON-STATE-PATH, coded on Lines 49 - 53. The record of EMPLOYEE-NUMBER 2 (the Marlowe record) is randomly READ with the HOLD option on Line 52. The record is <u>rewritten</u> using EMPLOYEE-ADDRESS-RECORD; i.e., the record containing all three paths. The Marlowe record was originally written using ALTERNATE-ADDRESS-RECORD, which contained only two paths; however, by rewriting the record using a record area containing all three paths, the record is accessible along the third path (EMPLOYEE-STATE) as well.

The use of the START statement to position the file along a particular path of an alternate indexed file is illustrated in the paragraph READ-LOWEST-CITY-RECORD, coded on Lines 54 - 58. The START statement functions like the START statement for indexed files (refer to Figure 2-5) except that here, a particular alternate path is identified. The sequence of statements on Lines 55 - 57 first initializes the EMPLOYEE-CITY path to LOW-VALUES on Line 55, and then issues the START statement using the key of EMPLOYEE-CITY. Since EMPLOYEE-CITY contains LOW-VALUES, the START statement will position the file at the first record containing a key higher than LOW-VALUES; which is to say, the record on the file containing the lowest value for EMPLOYEE-CITY. The READ NEXT statement on Line 58 will actually read this record.

The paragraph DELETE-SHAKESPEARE, coded on Lines 59 - 63, deletes the employee address record for William Shakespeare and terminates the program. The method is identical to the deletion of a record from an indexed file (refer to Figure 2-5, paragraph DELETE-EMPLOYEE-1). The record of EMPLOYEE-NUMBER 1 is READ with the HOLD option on Line 61, and the record is deleted on Line 62. Records of an alternate indexed file are deleted only by primary key; to remove a record from a path, the record must be READ with HOLD and then rewritten using a record description that does not specify the path.

When an alternate indexed file is opened in output mode, the alternate index paths for the records that are written are not created until the file is closed. When an alternate indexed file is opened in I-O mode, the alternate index paths for the records that are written are created immediately. This dynamic creation of the index paths may produce a noticeable delay in response time; the benefit for the cost incurred is that the record is immediately accessible along many access paths.

2.6 RELATIVE FILE PROCESSING IN COBOL

Relative files consist of records uniquely identified by an integer value greater than zero which specifies the record's logical ordinal position in the file. A relative file is composed of a serial string of areas. Each area has a relative record number and is capable of holding a logical record. Records are stored and retrieved according to the relative record number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records are written in the first nine record areas.

Figure 2-7 is a complete COBOL program illustrating relative file processing. The program sequentially reads the input file, TRANS-FILE, (line 5900), moves the record key information into the relative file key field (line 6100), and then retrieves the record in question from the relative file (line 7300) and writes it to the output print file (line 8300).

This program example illustrates random access of a relative file. The FILE-CONTROL entry for the relative file, lines 1600 through 2000, details the organization, access mode, and relative key. In this example, the records will be accessed and printed according to the order of records in the input file, TRANS-FILE.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     REL001.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.    WANG-VS.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINT-FILE
        ASSIGN TO "PRINT1" "PRINTER".

    SELECT TRANS-FILE
        ASSIGN TO "T-FILE" "DISK"
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.

    SELECT REL-FILE
        ASSIGN TO "R-FILE" "DISK"
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS RANDOM
        RELATIVE KEY IS WS-RELATIVE-KEY.

DATA DIVISION.
FILE SECTION.

FD  PRINT-FILE
    LABEL RECORDS ARE STANDARD.
01  PRINT-REC                   PIC X(132).

FD  TRANS-FILE
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 80 CHARACTERS.
01  TRANS-RECORD.
    05  TRANS-KEY-FIELD     PIC X(10).
    05  TRANS-DATA          PIC X(70).

FD  REL-FILE
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 80 CHARACTERS.
01  REL-RECORD.
    05  REL-KEY-FIELD       PIC X(10).
    05  REL-DATA            PIC X(70).

WORKING-STORAGE SECTION.

01  WS-RELATIVE-KEY         PIC X(10).
01  WS-PRINT-LINE.
    05  WS-PRINT-KEY-FIELD  PIC X(10).
    05  WS-PRINT-DATA       PIC X(70).


```

Figure 2-7. Relative File Processing

```
005100 PROCEDURE DIVISION.
005200
005300 INITIAL-RTN.
005400     OPEN INPUT TRANS-FILE.
005500     OPEN I-O REL-FILE.
005600     OPEN OUTPUT PRINT-FILE.
005700
005800 PROCESS-RTN.
005900     READ TRANS-FILE NEXT
006000             AT END GO TO END-OF-JOB.
006100     MOVE TRANS-KEY-FIELD TO WS-RELATIVE-KEY.
006200     PERFORM PROCESS-A-RELATIVE THRU PROCESS-EXIT.
006300     GO TO PROCESS-RTN.
006400
006500 END-OF-JOB.
006600     CLOSE TRANS-FILE
006700           REL-FILE
006800           PRINT-FILE.
006900     STOP RUN.
007000
007100
007200 PROCESS-A-RELATIVE.
007300     READ REL-FILE
007400         INVALID KEY DISPLAY "RECORD NOT FOUND"
007500         GO TO PROCESS-EXIT.
007600     MOVE REL-KEY-FIELD TO WS-PRINT-KEY-FIELD.
007700     MOVE REL-DATA TO WS-PRINT-DATA.
007800     PERFORM PRINT-ONE THRU PRINT-EXIT.
007900 PROCESS-EXIT.
008000     EXIT.
008100
008200 PRINT-ONE.
008300     WRITE PRINT-REC FROM WS-PRINT-LINE AFTER ADVANCING 1 LINE.
008400 PRINT-EXIT.
008500     EXIT.
008600
```

Figure 2-7. Relative File Processing (continued)

Relative files can also be accessed by the sequential and dynamic access modes. When accessing a relative file sequentially, programmers can use both the READ and START statements. The READ statement makes available the next logical record of the file, while the START statement provides a basis for logical positioning within the file for subsequent, sequential retrieval. After the successful execution of a START statement, records can be read sequentially by issuing READ statements.

Dynamic access mode allows a programmer to employ both random and sequential access in a single program. For instance, random access is accomplished by issuing a READ statement, while sequential acccess can be accomplished by issuing a READ NEXT statement following the successful execution of a START statement.

2.7 DMS SHARING ENVIRONMENT

The DMS Sharing environment allows multiple programs to access and update the same file concurrently. Consecutive, indexed, and alternate indexed files can be opened in shared mode. The functions provided for shared consecutive files ("log files") differ from the functions provided for shared indexed or alternate indexed files. These differing functions will be discussed in this section, along with program examples.

DMS Sharing is implemented in COBOL by means of the OPEN SHARED, HOLD, HOLD LIST, and FREE ALL statements. The programmer can code the WITH KEYS and INITIAL phrases with HOLD and HOLD LIST to request a generic range of records. Resource conflicts can be handled by coding the TIMEOUT and HOLDER-ID phrases with the HOLD, READ, and WRITE statements.

2.7.1 Shared Consecutive File (Log File) Support

A shared consecutive file, or log file, provides the facility of logging information regarding file-related updates required for an application program. A log file can be used to provide a user-defined audit trail of additions, updates, and deletions to a file or files. For example, many users can update a data file concurrently by opening the file in shared mode. By writing a record to the log file recording the change to the data file at the time of the change, a history of changes to the file is preserved. This may be useful for reporting purposes, or for restoring a file to a previous state.

A log file is a consecutive file created in shared mode. If a log file is not opened in shared mode, it is processed as a consecutive file. Therefore, by opening a consecutive file in shared mode, a program can process it as a log file; subsequently, another program, by opening the same consecutive file with an open mode other than shared, can process it as a consecutive file.

In some applications, it is necessary to write a record immediately to the log file; in other applications, it is not necessary to do so. If the application includes procedures that require the log file to reflect every change to the data files at the moment they have been made, the log file record must be written to disk immediately. If no such requirement exists, records to be written to the log file are temporarily stored in a buffer and written to the file only when the buffer is full (refer to Section 2.8 for a discussion of buffering). The facility for writing a record immediately to the log file is called the "write-through" option.

The VS distinguishes between a log file with the write-through option and a log file without the write-through option by examining the first character of the file name. If the first character of the file name is an at-sign (@) and it is opened as a shared consecutive file, records will be written immediately to the disk -- in other words, it will have the write-through option. Otherwise, if the first character of the file name is not an at-sign (@), and it is opened as a shared consecutive file, records will be buffered and the block written to the disk only when the block is full -- in other words, records will be buffered, without the write-through option.

The only valid operation on log files is WRITE. Since the purpose of a log file is to record information based on updates to other files (indexed or alternated indexed shared files), it is assumed that while the updates are being done the file will be opened in shared mode, to allow many users to log their activity to the file. However, if a report of such activity is to be produced, or if the data file is to be restored based on information recorded in the log file, the file should be opened using an open mode other than shared and processed as a consecutive file.

The log file need not exist before being opened in shared mode. The operating system will create a log file with a default record count of 1000.

Figure 2-8 is a complete COBOL program that writes a record to a log file using the write-through option.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  LOGFILE.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT LOGFILE
000007         ASSIGN TO "LOGFILE", "DISK",     NODISPLAY,
000008         ORGANIZATION    IS SEQUENTIAL
000009         ACCESS MODE     IS DYNAMIC.
000010 DATA DIVISION.
000011 FILE SECTION.
000012 FD  LOGFILE
000013     VALUE OF FILENAME IS "@LOGFILE"
000014     LABEL RECORDS ARE STANDARD.
000015 01  LOG-FILE-RECORD-NUMBER         PIC S9(5)  COMPUTATIONAL.
000016 PROCEDURE DIVISION.
000017 OPEN-THE-LOGFILE.
000018     OPEN SHARED LOGFILE.
000019 WRITE-RECORD-IMMEDIATELY.
000020     MOVE 1 TO LOG-FILE-RECORD-NUMBER.
000021     WRITE LOG-FILE-RECORD-NUMBER.
000022     STOP RUN.
```

Figure 2-8. Processing a Log File with the Write-Through Option

The FILE-CONTROL entry for LOGFILE, coded on Lines 6 - 9, specifies a consecutive file, because the ORGANIZATION IS SEQUENTIAL clause is coded on Line 8, to be processed in dynamic access mode, because ACCESS MODE IS DYNAMIC is coded on Line 9. At this point, the log file cannot be distinguished from any other consecutive file to be processed in dynamic access mode.

The record description entry for LOGFILE is coded on Lines 12 - 15. As specified, this could be a record description entry for a consecutive file.

Only when the file is opened in shared mode, by the OPEN statement on Line 18, does the file become a log file. If LOGFILE had been opened in any mode other than shared (for example, in I-O) mode, it would not have become a log file.

The only valid operation on a log file is WRITE. A record containing the value 1 is written to LOGFILE on Line 21. In addition, since the first character of the file name is an "@" (the VALUE OF FILENAME clause on Line 13 specified the name "@LOGFILE") the record is written immediately to the disk. If the first character of the file name had not been an "@", the record would have been written only if the buffer was full.

2.7.2 Shared Indexed File Support

The DMS Sharing environment offers two levels of functionality for indexed (or alternate indexed) files opened in shared mode. The first level, Elemental DMS Sharing, allows a program to hold one record at a time. The second level, DMS Sharing, allows a program to hold more than one record at a time. Holding a record reserves that record for subsequent modification. Opening a file in shared mode provides the programmer with the same functions as opening a file in I-O mode. These functions are READ, WRITE, REWRITE, DELETE, and START. To open a file in shared mode, the programmer codes the OPEN statement as follows:

```
OPEN SHARED file-name.
```

To reserve a record for subsequent modification, the programmer codes the READ statement with the HOLD phrase as follows:

```
READ WITH HOLD
```

Elemental Sharing

Elemental DMS sharing allows a program to hold only one record at a time. No program can hold a record currently being held by another program. Therefore, another program issuing a READ with the WITH HOLD phrase must wait until the program holding the record releases it. Releasing the record is done implicitly by successful execution of the following statements:

- A REWRITE of the record
- A DELETE of the record
- A READ WITH HOLD of another record (even if the record is in another file)
- A CLOSE on the file.

Because other programs must wait for a held record to be released, it is recommended that a record be held only where necessary. A READ with HOLD must be issued previous to issuing a REWRITE or a DELETE, regardless of whether the file is opened in I-O or shared mode. A REWRITE or a DELETE issued without a previous READ with the WITH HOLD phrase will produce an error.

DMS Sharing

DMS Sharing allows a program to hold more than one record and/or file at a time. The data being held is referred to as resources. Resources are held as a group. This allows a program to perform related updates or retrieval, with options for handling conflicting resource requests. Requests for resources may be made on either a preclaim or claim-as-needed basis.

Resources can be a record, a range of records in an indexed file (identified by a generic primary key), or a file. Resources are identified by the primary key as follows:

- A record resource is identified by the value of the record's primary key.
- A generic key resource is identified by the value of the first N characters of the records' primary key. For example, if the primary key is five characters long, the user can specify a generic key with the first three characters of the primary key equal to "100". In this case, all records which have the first three characters of the primary key equal to "100" are included in the resource.
- A file resource is identified by the name of the indexed or alternate indexed file.

Holds for Update and for Retrieval

A resource can be held either for update or for retrieval depending upon the level of concurrent access that is desired.

When a resource is held for update, records within the resource can be modified (by WRITE, REWRITE, or DELETE) only by the program issuing the hold. Other programs can read the data in the resource; however, no other programs can either hold or update the resource. This restriction guarantees the integrity of the data in the resource.

When a resource is held for retrieval, no program, including the one holding the resource, can modify the resource, but any program can read the resource. This ensures that no records within the resource are modified until the resource is released. If this precaution is not taken, it is possible that a program will read a record which does not reflect the most recent modifications to it. More than one program can hold the same resource for retrieval.

A program holding a resource in one hold class, retrieval or update, must release the resource before it can hold that resource or any items within it in the other hold class.

Preclaim Strategy

Programs claiming resources using the preclaim strategy hold all resources at once. The object program issues a HOLD statement, which requests that all resources specified in the HOLD statement and in any HOLD LIST statements issued after a previous HOLD statement (or after the start of the program, if no HOLD statements have yet been issued) be held as a group. The system must be able to hold all the resources at once; if any resource cannot be held, the entire HOLD request is denied. When a HOLD request is denied, the list of desired resources must be built again before issuing another HOLD statement. In the preclaim strategy, in order to request additional resources, previously held resources must first be released by means of the FREE ALL statement.

Handling Resource Request Conflicts

A request to hold resources for update will not be honored while another program is holding the resources for update. The requesting program must wait for the holding program to release the rights or resources. The programmer can specify, through the TIMEOUT phrase, how many seconds (0 to 255) to wait for a hold request to be granted. If the request cannot be granted within the specified period, the program can examine the HOLDER-ID data name to determine the ID of the user holding the requested items. Without the TIMEOUT phrase, the length of the wait is unbounded.

HOLD Statement

The HOLD statement is the COBOL method for requesting holding of specified resources (records, a generic key, and/or files). The resource can be held either for retrieval purposes (using the FOR RETRIEVAL phrase) or for update purposes (using the FOR UPDATE phrase). The number of seconds (0 to 255) which the program will wait can be specified in the TIMEOUT phrase. If the HOLD request cannot be satisfied within the specified number of seconds, the data name associated with the HOLDER-ID phrase contains the ID of the user who is running the program preventing the HOLD request from being granted.

Figure 2-9 is a complete COBOL program illustrating the VS COBOL statements that support the holding of multiple resources. Lines 51 to 56 illustrate the HOLD statement with the TIMEOUT phrase. In this example, the program requests that records of PERSONNEL-FILE be held for retrieval. If this request cannot be honored (for example, another program is holding all or part of PERSONNEL-FILE for update), the program waits 5 seconds (specified in the TIMEOUT phrase). If the request is not honored within 5 seconds, the 3-character field name WHO-HAS-IT contains the ID of the user whose program is preventing the request from being honored, and a message indicating that WHO-HAS-IT is holding PERSONNEL-FILE is displayed.

Holding a Generic Key of Records

A COBOL program can hold a generic range of records of a shared indexed file. The WITH KEYS option of the HOLD statement defines the range as those records having a particular primary key. Coding the INITIAL phrase defines the range as those records having a specified value in the first N characters of the primary key.

Lines 48 to 51 of Figure 2-9 illustrate how to hold a range of records of an indexed file. EMPLOYEE-FILE is an indexed file with a primary key, EMPLOYEE-NUMBER, of 5 characters. Data name DEPARTMENT references the first three characters of EMPLOYEE-NUMBER. To hold the records of employees in department 100, move "100" to DEPARTMENT on line 48 and specify WITH KEYS INITIAL 3 CHARACTERS OF EMPLOYEE-RANGE-NUMBER on line 50. When the HOLD is issued (line 51), all records of EMPLOYEE-FILE whose primary key start with 100 are held.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  HOLDEXMP.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT EMPLOYEE-FILE
000007       ASSIGN TO "EMPLOYEE",      "DISK",     NODISPLAY,
000008       ORGANIZATION    IS INDEXED
000009       ACCESS MODE
000010       IS DYNAMIC
000011       RECORD KEY     IS EMPLOYEE-NUMBER.
000012     SELECT PERSONNEL-FILE
000013        ASSIGN TO "PERSONS",     "DISK",     NODISPLAY,
000014        ORGANIZATION    IS INDEXED
000015        ACCESS MODE     IS DYNAMIC
000016        RECORD KEY     IS PERSONNEL-RECORD-NUMBER.
000017 DATA DIVISION.
000018 FILE SECTION.
000019 FD  EMPLOYEE-FILE
000020     LABEL RECORDS ARE STANDARD.
```

Figure 2-9. Holding Multiple Resources in COBOL

```
000021 01  EMPLOYEE-RECORD.
000022     03  EMPLOYEE-NUMBER.
000023         05 DEPARTMENT PIC XXX.
000024         05 FILLER PIC XX.
000025     03  EMPLOYEE-NAME      PIC X(20).
000026 FD  PERSONNEL-FILE
000027     LABEL RECORDS ARE STANDARD.
000028 01  PERSONNEL-RECORD.
000029     03  PERSONNEL-RECORD-NUMBER    PIC 9(5).
000030     03  PERSONNEL-DATA   PIC X(20).
000031 WORKING-STORAGE SECTION.
000032 77  WHO-HAS-IT       PIC X(3).
000033 PROCEDURE DIVISION.
000034 START-PROGRAM.
000035     PERFORM HOLD-RESOURCES THRU END-HOLD.
000036     STOP RUN.
000037 HOLD-RESOURCES.
000038     OPEN SHARED EMPLOYEE-FILE.
000039     OPEN SHARED PERSONNEL-FILE.
000048     MOVE "100" TO DEPARTMENT.
000049     HOLD LIST RECORDS OF EMPLOYEE-FILE
000050     WITH KEYS INITIAL 3 CHARACTERS OF EMPLOYEE-NUMBER.
000051     HOLD RECORDS OF PERSONNEL-FILE FOR RETRIEVAL
000052            TIMEOUT OF 5 SECONDS
000053            HOLDER-ID IN WHO-HAS-IT
000054        DISPLAY WHO-HAS-IT
000055       " is holding PERSONNEL-FILE and records with first 3 charac
000056-      "ters of '100' in  EMPLOYEE-FILE."
000057       GO TO CLOSE-FILES.
000058     DISPLAY "PERSONNEL-FILE and records with first 3 characters
000059-      "of '100' in EMPLOYEE-FILE are held by this program.".
000061     FREE ALL.
000062 CLOSE-FILES.
000063     CLOSE EMPLOYEE-FILE,  PERSONNEL-FILE.
000064 END-HOLD.
000065     EXIT.
```

Figure 2-9. Holding Multiple Resources in COBOL (continued)

HOLD LIST Statement

A program in the DMS Sharing environment can build a list of resources using the HOLD LIST statement. Execution of the HOLD LIST statement does not hold any resources; the resource request is merely added to a list. When a HOLD statement (without the LIST option) is encountered, the program attempts to hold all the resources on the list. If it is impossible for any resource on the list to be held, none of the resources are held, and the list must be rebuilt before being requested again.

The ability to construct a list of resources to hold can be useful in many applications. For example, when customer orders are being processed, it is usually desirable to update both the order file and the inventory file at the same time. The following code can hold two such files for simultaneous updating.

```
    HOLD LIST RECORDS OF INVENTORY-FILE FOR UPDATE.
    HOLD RECORDS OF ORDER-FILE FOR UPDATE.
```

When the HOLD for records in ORDER-FILE is executed, an attempt is made to hold both INVENTORY-FILE (the resource requested in HOLD LIST) and ORDER-FILE (the resource requested by the HOLD).

The HOLD LIST statement is illustrated on lines 49 and 50 of Figure 2-9.

FREE Statement

A program should release resources when the need for them has been satisfied. This is done by coding the FREE statement. If the program does not code the FREE statement, other programs are prevented from obtaining needed resources. The program will release all resources at once if FREE ALL is coded.

2.8 FILE PERFORMANCE OPTIONS IN COBOL

Relative to processing logical records in memory, transferring physical blocks from the disk to memory is a time consuming process: the more disk I/O operations, the slower the file performance. The VS provides strategies for tuning file performance. These strategies are the large buffer strategy for consecutive files, the buffer pooling strategy for indexed files, and the specification of index and data packing density. These strategies are not guaranteed to increase file performance. For example, if the size of the buffers is enlarged (an action that should theoretically enhance performance), other unintended consequences that reduce performance can result (such as an increase in the paging rate). The file performance options provided by COBOL are tools that should be used with care.

2.8.1 Large Buffer Strategy for Consecutive Files

A buffer is a memory area that temporarily holds blocks transferred from the disk. The larger the buffer, the more data that can be transferred per disk I/O operation. The minimum buffer size for a disk file is 2K, or one physical disk block. For processing consecutive files, VS COBOL provides the option of increasing the buffer size to a maximum of 18K. To increase the buffer size, code the BUFFER SIZE IS clause in the FILE-CONTROL entry for the file.

2.8.2 Buffer Pooling Strategy for Indexed Files

Multiple Files in a Buffer Pool

For indexed files opened in I-O mode, file performance may be enhanced by use of the "buffer pooling" strategy. Buffer pooling is automatically used for indexed files opened in shared mode. In buffer pooling, one buffer area is used by many indexed files. When a disk block is read, the VS operating system will determine what block in the buffer pool has been least recently used, and overlay that block. This means that buffer areas that have been recently referenced will remain in memory, thus saving disk I/O operations. A buffer pool of up to 120K (60 areas of 2K each) can be allocated; any number of indexed files in the same program can participate in the pool.

To specify buffer pooling, perform the following operations.

1. Specify the size of the buffer pool. This is accomplished through the RESERVE NN AREAS clause of the FILE-CONTROL entry for one (and only one) of the indexed files participating in the buffer pool.

2. Specify the files sharing the buffer pool. This is accomplished through the SAME AREA FOR file1, file2, ... clause in the I-O-CONTROL section of the Environment Division.

The buffer pool is allocated when one of the files specified in the SAME AREA clause is opened in I-O mode. The buffer pool is deallocated when all the files in the buffer pool have been closed. After a file in the buffer pool has been opened in I-O mode, buffer pooling statistics can be shown by pressing HELP, PF3 and PF2. The buffer pooling statistics will show buffer hit counts and buffer miss counts, which are counts of whether a requested record to be read was found in the buffer (hit count), or whether it was not found in the buffer (miss count). The buffer pooling statistics are a tool for monitoring file access performance. Recompiling the program after enlarging the buffer pool (coding a larger number in the RESERVE NN AREAS clause) or altering the number of files in the buffer poolmay enhance file processing performance.

Figure 2-10 is a complete COBOL program showing buffer pooling implementation for two indexed files. The indexed files EMPLOYEE-FILE and PERSONNEL-FILE participate in a buffer pool. This participation is specified by the SAME AREA clause coded on Line 18. The size of the buffer pool is 20 areas (10 areas of 2K bytes each). This size is specified by the RESERVE 10 AREAS clause of the FILE-CONTROL entry for EMPLOYEE-FILE, as coded on Line 11.

The buffer pool is actually allocated when one of the files is opened in I-O mode. After EMPLOYEE-FILE is successfully opened in I-O mode by execution of the OPEN statement on Line 33, the buffer pool is in use. The buffer pool is deallocated after successful execution of the CLOSE statement on Line 37.

One File In A Buffer Pool

Buffer pooling can be specified for one file. A buffer pool for one file should improve file performance because the program takes advantage of the buffer replacement optimization provided by the VS operating system. To specify buffer pooling for one file, repeat the file name in the SAME AREA clause. To specify a buffer pool containing only the file FILE1 code in I-O-CONTROL as shown.

```
SAME AREA FOR FILE1 FILE1.
```

Multiple Buffer Pools

Multiple buffer pools can be specified by the following procedure:

1. Repeat the SAME AREA clause for each buffer pool to be defined. Specify the files to share the buffer pool after each SAME AREA specification. For example, if FILE1 and FILE2 share one buffer pool, and FILE3 and FILE4 share another buffer pool, code in I-O-CONTROL as follows:

   ```
   I-O-CONTROL.
       SAME AREA FOR FILE1 FILE2
       SAME AREA FOR FILE3 FILE4.
   ```

2. Specify the size of each buffer pool. One (and only one) file referenced in each SAME AREA clause must have a RESERVE NN AREAS clause coded, which declares the size of each buffer pool.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  BUFPOOL.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMPLOYEE-FILE
        ASSIGN TO "EMPLOYEE",       "DISK",     NODISPLAY,
        ORGANIZATION    IS INDEXED
        ACCESS MODE     IS DYNAMIC
        RECORD KEY      IS EMPLOYEE-NUMBER
        RESERVE         10 AREAS.
    SELECT PERSONNEL-FILE
        ASSIGN TO "PERSONS",        "DISK",     NODISPLAY,
        ORGANIZATION    IS INDEXED
        ACCESS MODE     IS DYNAMIC
        RECORD KEY      IS PERSONNEL-RECORD-NUMBER.
I-O-CONTROL.
    SAME AREA FOR EMPLOYEE-FILE PERSONNEL-FILE.
DATA DIVISION.
```

Figure 2-10. Buffer Pooling For Two Indexed Files

```
000020 FILE SECTION.
000021 FD  EMPLOYEE-FILE
000022     LABEL RECORDS ARE STANDARD.
000023 01  EMPLOYEE-RECORD.
000024     03  EMPLOYEE-NUMBER            PIC 9(5).
000025     03  EMPLOYEE-NAME              PIC X(20).
000026 FD  PERSONNEL-FILE
000027     LABEL RECORDS ARE STANDARD.
000028 01  PERSONNEL-RECORD.
000029     03  PERSONNEL-RECORD-NUMBER    PIC 9(5).
000030     03  DEPARTMENT                 PIC X(20).
000031 PROCEDURE DIVISION.
000032 START-PROGRAM.
000033     OPEN I-O PERSONNEL-FILE.
000034     OPEN I-O EMPLOYEE-FILE.
000035     DISPLAY "BOTH FILES SHARE A 20K BUFFER POOL.".
000036     CLOSE PERSONNEL-FILE.
000037     CLOSE EMPLOYEE-FILE.
000038 NO-BUFFER-POOL.
000039     STOP RUN.
```

Figure 2-10. Buffer Pooling For Two Indexed Files (continued)

Figure 2-11 is a complete COBOL program specifying two buffer pools, each with one file. The first buffer pool, specified in the SAME AREA clause on Line 19, reserves a buffer pool for EMPLOYEE-FILE; the second buffer pool, specified in the SAME AREA clause on Line 20, reserves a buffer pool for PERSONNEL-FILE. As in the program in Figure 2-11, the buffer pool is allocated when a file in it is opened, so that the first buffer pool is allocated by the OPEN statement on Line 35 and the second buffer pool is allocated by the OPEN statement on Line 36. The buffer pool is deallocated when all files in it have been closed; therefore, the first buffer pool is deallocated by the CLOSE statement on Line 38, while the second buffer pool is deallocated by the CLOSE statement on Line 39.

To show statistics for the two buffer pools, press HELP, PF 3, and PF 2 when the DISPLAY statement on Line 37 appears.

2.8.3 Setting the Index and Data Packing Densities

All VS disk files are stored in 2K units, called blocks. Records of indexed files are stored in a data block in primary key order. If more records than can fit in a block are added, a new block is designated as a data block and the record is added there. All data blocks contain a pointer to the next data block. An index block contains pointers to the data block containing the record; for a read by primary key, the index block is scanned to obtain the block number for the data block containing the record, and the data block, in turn, is scanned for the actual record.

If a record or an index does not fit in a block, because it has to be added in the middle of a block, it is necessary to move the part of the block that does not fit to another block, and to add the record in the original block. This process is called "block splitting".

To reduce the need for block splitting and increase record access performance on indexed files, VS COBOL provides a method for allocating space for future record additions. The added records can fit in the preallocated space and a new block need not be created. A percentage of the index block and/or the data block can be filled with records; the remainder of the block is not filled with records but available for future additions. In the File Section, coding the following statements for the file -- assuming EIGHTY and FIFTY are defined in Working-Storage as numeric items with values of 80 and 50 -- when the file is opened in output mode and records are written to it, only 80 per cent of the index block and 50 per cent of the data block is filled with records; the rest is available for future record additions.

```
    VALUE OF INDEX AREA IS EIGHTY
    VALUE OF DATA AREA IS FIFTY
```

Setting the index or data packing density in itself does not guarantee improved file performance. The optimum index and data packing density depends upon the amount and degree of randomness of record updates and/or additions and, as such, is application-dependent.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  BUFPOOL2.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT EMPLOYEE-FILE
000007         ASSIGN TO "EMPLOYEE",      "DISK",     NODISPLAY,
000008         ORGANIZATION    IS INDEXED
000009         ACCESS MODE     IS DYNAMIC
000010         RECORD KEY      IS EMPLOYEE-NUMBER
000011         RESERVE         10 AREAS.
000012     SELECT PERSONNEL-FILE
000013         ASSIGN TO "PERSONS",       "DISK",     NODISPLAY,
000014         ORGANIZATION    IS INDEXED
000015         ACCESS MODE     IS DYNAMIC
000016         RECORD KEY      IS PERSONNEL-RECORD-NUMBER
000017         RESERVE         20 AREAS.
000018 I-O-CONTROL.
000019     SAME AREA FOR EMPLOYEE-FILE EMPLOYEE-FILE
000020     SAME AREA FOR PERSONNEL-FILE PERSONNEL-FILE.
000021 DATA DIVISION.
```

Figure 2-11. Multiple Buffer Pools

```
000022 FILE SECTION.
000023 FD  EMPLOYEE-FILE
000024     LABEL RECORDS ARE STANDARD.
000025 01  EMPLOYEE-RECORD.
000026     03  EMPLOYEE-NUMBER              PIC 9(5).
000027     03  EMPLOYEE-NAME                PIC X(20).
000028 FD  PERSONNEL-FILE
000029     LABEL RECORDS ARE STANDARD.
000030 01  PERSONNEL-RECORD.
000031     03  PERSONNEL-RECORD-NUMBER      PIC 9(5).
000032     03  DEPARTMENT                   PIC X(20).
000033 PROCEDURE DIVISION.
000034 START-PROGRAM.
000035     OPEN I-O PERSONNEL-FILE.
000036     OPEN I-O EMPLOYEE-FILE.
000037     DISPLAY "TWO BUFFER POOLS ARE IN USE.".
000038     CLOSE PERSONNEL-FILE.
000039     CLOSE EMPLOYEE-FILE.
000040 NO-BUFFER-POOL.
000041     STOP RUN.
```

Figure 2-11. Multiple Buffer Pools (continued)

2.9 HANDLING FILE-RELATED ERROR CONDITIONS IN COBOL

Occasionally, an operation on a file (OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START) may not be successful. This can be caused by:

- An error in the COBOL program. A READ operation for a file opened in output mode, or a REWRITE of a record that has not been read with the hold option, are examples of typical COBOL file-related programming errors.

- An error in access of the data file. A sequential READ operation on a file in which the end-of-file condition has been reached, or a WRITE operation of a record of an indexed file that has a duplicate primary key value, are examples of typical file-related data access errors.

- A system-related error. A WRITE operation to a file in which there is no further room for expansion or a permanent I/O error on the file resulting from a hardware malfunction are examples of typical system-related errors.

These errors can be treated as cancel conditions in which the only option is to cancel the program after the system issues a message. Options in VS COBOL, however, allow the program to intercept the error conditions, allowing the program to continue with the possibility that the operation can be reissued successfully. For example, in a data entry application, the COBOL program displays a screen requesting information regarding an employee. After the information has been validated, the program issues a WRITE to the indexed file EMPLOYEE-FILE, which has a primary key of EMPLOYEE-NAME. If a record having the value in EMPLOYEE-NAME already exists, this is a file-related error that will produce a system message. At that point, the only option is to cancel the program.

If, on the other hand, the program could recognize the error and, if the error occurs, redisplay the screen with an error message and request a different value for EMPLOYEE-NAME, a subsequent WRITE of the record of EMPLOYEE-FILE may be successful and the program can proceed.

VS COBOL provides the following facilities for intercepting file-related errors:

- The AT END exit for a sequential READ. For a sequential READ (a READ for a file with ACCESS MODE IS SEQUENTIAL, or a READ NEXT for a file with ACCESS MODE IS DYNAMIC), if the end-of-file condition is encountered (there are no more records in the file), the program will perform the imperative statement coded with the AT END exit. For example, to display the number of records in the file when the end-of-file condition is encountered for the file FILE1, code the following statement:

  ```
  READ FILE1 AT END DISPLAY "NUMBER OF RECORDS = " RECORD-NUMBER.
  ```

- The INVALID KEY exit for a random READ, a DELETE, a REWRITE, a START, or a WRITE. If any operation other than a sequential read is attempted and is unsuccessful because of a data file access error, the program will perform the imperative statement coded with the INVALID KEY exit. For example, to display a message when a duplicate key value for a record is encountered for the record EMPLOYEE-RECORD, code the following statement:

```
    WRITE EMPLOYEE-RECORD INVALID KEY DISPLAY "Invalid key
encountered."
```

The INVALID KEY exit can be coded with READ, DELETE, REWRITE, START, or WRITE to detect such data file access errors.

- A USE procedure in the DECLARATIVES for a system-related error on the file. If a system-related error, such as a permanent I/O error or a WRITE operation is attempted on a file which has no room for expansion, or if an INVALID KEY or AT END exit is not coded, a USE procedure in the DECLARATIVES (at the beginning of the Procedure Division) can be coded to handle this condition. The system message will still be produced. After the message is produced, the program branches to the DECLARATIVES logic. For example, to code statements handing I/O errors if they are detected on EMPLOYEE-FILE, code the following:

```
 PROCEDURE DIVISION.
 DECLARATIVES.
 I-O-ERROR SECTION.
     USE AFTER STANDARD ERROR PROCEDURE ON EMPLOYEE-FILE.
 INVALID-FUNCTION-PARAGRAPH.
*    Code I/O error logic here.
 END DECLARATIVES.
```

The USE AFTER STANDARD ERROR PROCEDURE is executed only if a system-related error is detected on EMPLOYEE-FILE as the result of an operation (OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, or START). The system error message is first displayed, then a branch is taken to INVALID-FUNCTION-PARAGRAPH, where the program-directed error handling is executed. END DECLARATIVES signifies the end of the error logic and causes a branch to the statement after the statement that caused the branch to the DECLARATIVES.

Figure 2-12 is a complete COBOL program that demonstrates the use of the INVALID KEY exit, the AT END exit, and the DECLARATIVES logic to process file-related errors without causing the program to cancel.

The FILE-CONTROL entry and record description entry define an indexed file FILE1. In all the COBOL programs analyzed up to this point, the program starts executing the first statement in the Procedure Division. The only exception is DECLARATIVES. DECLARATIVES are only executed in case of system-related errors that are to be processed by the program. In Figure 2-12, the DECLARATIVES start on Line 22 and end with the END DECLARATIVES statement on Line 28. If DECLARATIVES appear in a program, the first statement executed is the statement after END DECLARATIVES -- here, the paragraph GENERATE-INVALID-KEY, starting on Line 30.

The paragraph GENERATE-INVALID-KEY, coded on Lines 30 - 35, opens FILE1 in output mode, and then issues two WRITE statements. The first WRITE statement, coded on Line 32, successfully writes a record to FILE1. The second WRITE statement, coded on Line 33, generates an INVALID KEY condition because the entire record, and thus the primary key, is a duplicate of the record that was written to the file. Since the INVALID KEY exit was coded for the WRITE statement, the imperative statement associated with the INVALID KEY exit -- the DISPLAY statement on Line 34 -- is executed, and the message "INVALID KEY condition encountered." displays. After the message displays, if ENTER is pressed, the next statement is executed and FILE1 is closed. FILE1 now contains one record: the record written by the WRITE statement of Line 32.

The paragraph GENERATE-AT-END-CONDITION, coded on Lines 36 - 40, demonstrates the use of the AT END exit. FILE1 is opened in input mode on Line 37. The READ NEXT statement, coded on Line 38, reads the first (and only) record of FILE1 successfully. The second READ NEXT statement, coded on Lines 39 - 40, generates the end-of-file condition, since FILE1 contains only one record. The AT END exit is coded for this READ NEXT; if the end-of-file condition is encountered, the imperative statement associated with the AT END exit is executed; here, the DISPLAY statement on Line 40 displays the message "AT END condition encountered.".

The paragraph GENERATE-DECLARATIVE-BRANCH, coded on Lines 41 - 44, demonstrates a condition for forcing a branch to the DECLARATIVES. FILE1 was opened in input mode on Line 37 and remains open in input mode. A WRITE statement is invalid for files opened in input mode. The WRITE statement on Line 42 therefore cannot be executed for FILE1. This file-related error causes a branch to the DECLARATIVES. In the DECLARATIVES, the USE AFTER STANDARD ERROR statement specifies that, if an error should occur on FILE1, the DECLARATIVES branch should be taken. After the unsuccessful WRITE, the paragraph INVALID-FUNCTION-PARAGRAPH in the DECLARATIVES is executed and the message "Invalid function file status should = 95. It = 95" is displayed. If ENTER is pressed, the END DECLARATIVES is encountered, the CLOSE statement on Line 43 is executed, and the program terminates after successful execution of the STOP RUN on Line 44.

Every file-related operation sets a 2-byte field called the File Status. The FILE STATUS clause in the FILE-CONTROL entry for a file specifies a 2-byte field to contain the value of the File Status after every file-related operation. On Line 10 of the FILE-CONTROL entry for FILE1, the data name FILE-STATUS (defined in Working-Storage on Line 20) is to receive this value.

Appendix E describes the File Status in detail. A branch to the DECLARATIVES occurs if the File Status has a value equal to or greater than 30. The File Status for "invalid function" is 95; therefore, a branch to the DECLARATIVES is taken. If a File Status equal to or greater than 30 occurs in a COBOL program that does not have DECLARATIVES, a system error message will appear and the program must be cancelled.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  FILEERRS.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE1
        ASSIGN TO "FILE1", "DISK",     NODISPLAY,
        ORGANIZATION    IS INDEXED
        ACCESS MODE     IS DYNAMIC
        FILE STATUS     IS FILE-STATUS
        RECORD KEY      IS RECORD-KEY.
DATA DIVISION.
FILE SECTION.
FD  FILE1
    LABEL RECORDS ARE STANDARD.
01  FILE1-RECORD.
    03  RECORD-KEY                 PIC 9(010).
    03  FILLER                     PIC X(070).
WORKING-STORAGE SECTION.
77  FILE-STATUS                    PIC XX.
PROCEDURE DIVISION.
DECLARATIVES.
I-O-ERROR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON FILE1.
INVALID-FUNCTION-PARAGRAPH.
    DISPLAY "Invalid function file status should = 95. It = "
        FILE-STATUS.
END DECLARATIVES.
NON-DECLARATIVES SECTION.
GENERATE-INVALID-KEY.
    OPEN OUTPUT FILE1.
    WRITE FILE1-RECORD.
    WRITE FILE1-RECORD  INVALID KEY
        DISPLAY "INVALID KEY condition encountered.".
    CLOSE FILE1.
GENERATE-AT-END-CONDITION.
    OPEN INPUT FILE1.
    READ FILE1 NEXT.
    READ FILE1 NEXT AT END
        DISPLAY "AT END condition encountered.".
GENERATE-DECLARATIVE-BRANCH.
    WRITE FILE1-RECORD.
    CLOSE FILE1.
    STOP RUN.
```

Figure 2-12. File Error Handling

The FILE STATUS data name can also be tested directly after every operation. In the program illustrated by Figure 2-12, FILE-STATUS is updated after every file-related operation. Therefore, if testing of the FILE STATUS data item for a particular operation is required, the test must be performed before another file-related operation is issued. FILE-STATUS contains the following values after each file-related operation.

FILE-RELATED OPERATION	Value of FILE-STATUS
OPEN OUTPUT FILE1.	00 (This operation was successful.)
WRITE FILE1-RECORD.	00 (This operation was successful.)
WRITE FILE1-RECORD with INVALID KEY exit.	21 (Duplicate key value on indexed file creation.)
CLOSE FILE1.	00 (This operation was successful.)
OPEN INPUT FILE1.	00 (This operation was successful.)
READ FILE1 NEXT.	00 (This operation was successful.)
READ FILE1 NEXT with AT END exit.	10 (End of file on sequential read.)
WRITE FILE1-RECORD.	95 (Invalid function. A WRITE is not permitted for a file opened in input mode.)
CLOSE FILE1.	00 (This operation was successful.)

CHAPTER 3
DMS/TX

3.1 INTRODUCTION

DMS/TX is a transaction recovery system. An extension of DMS Sharing, DMS/TX provides multiple user sharing and rollback recovery of indexed data files processed in Record Access Method (RAM). Files used with DMS/TX are organized into a named set of indexed data files called a database. A DMS/TX database can exist on more than one volume. DMS/TX file updates performed by a VS COBOL program are grouped into units called transactions. A transaction is a related set of record updates that are posted as a group to preserve database consistency.

DMS/TX is available for all Wang VS computers that run a Release 6.0 or subsequent Operating System. This chapter is an overview of DMS/TX, highlighting its use with VS COBOL. The user is advised to read the DMS/TX Reference Manual before proceeding with this chapter.

3.1.1 Principal Features

Three main features of DMS/TX are DMS/TX File Sharing, Transaction Rollback Recovery, and Structural Integrity Monitoring.

DMS/TX File Sharing

DMS/TX provides a high level of file sharing, allowing multiple users simultaneous access to the same files. DMS/TX Sharing is more sophisticated than, yet fully compatible with, DMS Sharing.

DMS/TX allows each task on the system to hold a number of resources (records, groups of records, and/or files). Object programs hold these resources for update on a claim-as-needed basis. The claim-as-needed function allows:

- Any object program to claim records while already holding other records
- More than one program to do this simultaneously.

Each program can hold multiple resources for the duration of a transaction. All resources held by a program are released by the system at the conclusion of a transaction, as identified by the execution of a FREE ALL statement from within the program.

Transaction Rollback Recovery

Transaction rollback recovery ensures that transactions are fully applied to the data file(s) or not applied at all (rolled-back). If a transaction is rolled-back, all updates made to the data file(s) are removed, returning the file(s) to its previous consistent state. Consistency is maintained both within a file and between files whose updates must be coordinated. Rollback is automatically performed by the system when necessary and can be initiated as a program-invoked function.

Structural Integrity Monitoring

Structural integrity monitoring automatically monitors each update made to a file to detect impaired structural integrity. A file has structual integrity if, when a data record in an indexed file is updated, needed updates to that record's primary and alternate key index blocks are also performed.

DMS/TX performs this function by maintaining an indicator as part of each DMS/TX file. The indicator is updated each time a record is updated. If a system failure occurs, DMS/TX automatically checks each file's record update indicator. The user must then reorganize any files with impaired structural integrity before performing rollback recovery.

3.2 IMPLEMENTING DMS/TX IN COBOL

VS COBOL programs do not directly invoke DMS/TX. A program issues an OPEN statement on a file attached to a DMS/TX database and the system automatically initiates DMS/TX processing. The only requirement of the program is that it define its transactions by means of FREE ALL statements. The remaining DMS/TX related syntax is optional.

The same program can process files attached to a database, ordinary DMS files, and files attached to different databases. VS COBOL syntactical support for DMS/TX is as follows:

- Three VALUE OF clauses in the FILE SECTION of the DATA DIVISION. The three clauses are:

  ```
  VALUE OF RECOVERY-BLOCKS IS
  VALUE OF RECOVERY-STATUS IS
  VALUE OF DATABASE-NAME IS
  ```

  The VALUE OF RECOVERY-BLOCKS IS clause allocates the Recovery Blocks in output mode. All of the clauses retrieve the DMS/TX file information for existing files. They are optional.

- The ROLLBACK statement returns DMS/TX files to their previous consistent state if a transaction failed to complete. The ROLLBACK statement is optional.

- The FREE ALL statement ends a transaction, releasing all held resources. The FREE ALL statement is required.

- The Deadlock Declarative allows a task to override the system default deadlock handling when a deadlock occurs, returning control to the object program. The Deadlock Declarative is optional.

3.2.1 The VALUE OF RECOVERY-BLOCKS IS Clause

The VALUE OF RECOVERY-BLOCKS IS clause serves two purposes. For existing files, it returns the file's status with respect to DMS/TX. For new files created in output mode, it specifies whether the file can be attached to a DMS/TX database.

RECOVERY-BLOCKS can have three different values. They are:

N -- No Recovery Blocks
    The file is a DMS file and cannot be attached to a DMS/TX database.

A -- Recovery Blocks Allocated
    The file is a DMS file and can be attached to a DMS/TX database.

U -- Recovery Blocks Used
    The file is part of a DMS/TX database.

For new files, the only acceptable values are N and A. Existing files can have Recovery Blocks added through the DMS/TX utility. The correct syntax is:

```
                               data-name
VALUE OF RECOVERY-BLOCKS IS    literal
```

3.2.2. The VALUE OF RECOVERY-STATUS Clause

The VALUE OF RECOVERY-STATUS IS clause indicates whether the file is opened with transaction recovery. This value is only returned if the program opens the file in I-O or Shared mode. Possible values and their meanings for this field are as follows:

N -- No recovery
S -- Softcrash recovery
F -- Full recovery

The correct syntax is:

```
VALUE OF RECOVERY-STATUS IS  data-name
```

The value of data-name must be alphanumeric with a declared length of one character.

3.2.3 The VALUE OF DATABASE-NAME IS Clause

The VALUE OF DATABASE-NAME IS clause contains the name of the database the file is attached to. The correct syntax is:

```
VALUE OF DATABASE-NAME IS  data-name
```

The value of data-name must be alpha or numeric with a declared length of six characters.

3.2.4 Attaching Files to a DMS/TX Database

Files can be attached to a DMS/TX database through the DMSTX utility or through a program. To be attached, a file must first have Recovery Blocks allocated. Though not directly supported in VS COBOL, program-invoked attachment can be accomplished in two ways. Both methods require a call to a subroutine.

In the first method, attachment is accomplished at run-time by using an Assembler subroutine to call the SETRECOV SVC. The file to be attached must be closed at the time of the SETRECOV execution. VS COBOL programs access an Assembler subroutine containing SETRECOV with the CALL statement. Refer to the DMS/TX Reference manual for information on developing this subroutine.

Programs that create, attach, and use DMS/TX files should be written to observe the following sequence at run time:

1. Create the file with Record Blocks by opening it in Output mode
2. Close the file
3. CALL the Assembler subroutine to use SETRECOV to attach the file to a database with the SOFT recovery option
4. Re-open the file in I-O or Shared mode
5. Write records to the file

The second method of creating a DMS/TX file at run-time is for the program to issue a CALL to a subroutine that uses the DMSTX utility to attach a file to a database. With this method, a program first calls the subroutine to create and attach the file. After control is returned to the original program, it opens the file in I-O or Shared mode. The program can then write to the file. Refer to the DMS/TX Reference manual for details on the DMSTX utility.

3.2.5 Opening and Closing Files

The programming procedures for opening and closing files is the same for DMS/TX files as it is for DMS files. Programs still open a file in I-O or Shared mode to update records, and use the CLOSE statement to close it. In addition to terminating DMS/TX transactions, the CLOSE statement invokes an implicit FREE ALL statement, ending the current transaction.

3.2.6 Holding and Releasing Resources

DMS/TX allows each task to exclusively hold multiple resources. DMS/TX uses the same function requests as DMS. Programs with files opened in Shared mode can hold records or generic key groups of records as needed.

Holding Resources

The READ WITH HOLD and HOLD FOR UPDATE statements hold resources exclusively. The HOLD FOR RETRIEVAL statement provides a nonexclusive, shared hold. A program can hold a number of records in shared files. A READ statement without the HOLD option allows a program to read resources without locking those resources. In addition, a READ statement without the HOLD option allows a program to read resources currently locked by another task. Because of this, care should be used when coding the READ statement without the HOLD option.

DMS/TX support for resource holding provides additional support for programs updating records in DMS/TX files which have alternate index keys which do not allow duplicate values. In certain situations, exclusive locks are automatically applied by the system to the alternate index values as well as to the primary key values. If a task deletes a record from a non-duplicate alternate key path -- either by deleting the record or by removing the record from access by that key path -- the key value is locked, i.e., the value itself is prevented from being written by any other task.

Tasks check for locks on a nonduplicate alternate key value when they add a record to the path. If the value is locked, the task is queued until such time as the lock is removed or the time specified in the TIMEOUT phrase is exceeded. This allows other tasks to read the value. Tasks can add a record to the path by writing a record that is accessible by that path to the file, or by rewriting an accessible record with a new value for the alternate key, or with the record's bit mask reset to enable accessibility by that key path.

Releasing Resources

A FREE ALL statement ends the current transaction and begins the next transaction. It causes the system to commit all updates performed during the current transaction and releases all resources held during the transaction.

Resources are also released when a CLOSE statement for DMS/TX files opened in I-O or Shared mode is executed. When this occurs, the system executes an implicit FREE ALL.

3.2.7 DEADLOCK

A deadlock occurs when two programs each request a resource held the other program. The deadlock prevents both programs from proceeding. If one of the tasks does not free the held resources in the allotted time (as specified in the GENEDIT procedure), DMS/TX rolls back the current transaction of the waiting task and frees its locks.

Following successful deadlock processing DMS/TX returns control to the program if the programmer has included a deadlock exit routine in the DECLARATIVES section of the PROCEDURE DIVISION. The Deadlock Declarative is optional. If a deadlock situation is detected and the declarative is not present in a program, the system issues an error message and terminates the program run. Programs coded with the Deadlock Declarative avoid the extra step of having to restart the program run.

The syntax for coding a deadlock exit in VS COBOL is as follows:

```
USE AFTER DEADLOCK.
```

This statement is followed by a user-designed routine to restart the transaction or perform some other function.

3.2.8 Program-Initiated Rollback

DMS/TX offers the programmer the option of coding rollback recovery in the program logic. Upon execution of a program-initiated ROLLBACK statement, DMS/TX reverses updates, leaving files open and resources held. At the conclusion of the rollback operation, control is returned by the system to the program at the next instruction following the ROLLBACK statement.

The capability to rollback a transaction from a program is particularly useful for interactive data-entry applications. For example, if a data entry transaction involves keying a number of updates to a single screen, an error posted in one field could invalidate all of that screen's updates.

The syntax for coding a rollback is as follows:

<u>ROLLBACK</u> [ON <u>ERROR</u> imperative-statement].

The ON ERROR clause provides an executable routine in case of an unsuccessful execution of the ROLLBACK statement. Without the clause, the system cancels a program upon unsuccessful execution of the ROLLBACK statement.

Return codes for the ROLLBACK statement are contained in the special register RETURN-CODE. Refer to the ROLLBACK entry of Chapter 11 for a list of the return code values.

Programs containing rollback routines should be coded with the following considerations:

- All resources held by a transaction remain held following a program-initiated rollback.

- Open and Close statements, and VTOC operations are not rolled back. The programmer must be careful not to have these functions performed twice if, for instance, a screen has to be reprocessed.

- The contents of the user record area are unaffected by the rollback operation.

- Positional currency in database files opened for update is unpredictable. If the transaction performs consecutive processing, the program should be coded to re-establish currency by using a START statement.

3.2.9 Rollback Following a Program Cancel

If a program aborts or is cancelled by the user, the system automatically rolls back the current transaction as part of the cancel processing. During the rollback operation, the "File Cleanup in Progress" message is displayed on the workstation screen. Upon successful completion of the rollback, the system invokes a FREE ALL statement, releasing all resources. The cancel processing then closes the files based on the link level at which they were opened, invoking the DMS/TX close processing as appropriate.

3.3 PROGRAM EXAMPLE

Figure 3-1 is a sample COBOL program demonstrating the DMS/TX functionality. The program accesses two files attached to a database for interactive update. The data name DB1, described in the Working-Storage Section, is used by the program to reference the database the files are attached to.

The program accepts and posts updates to the employee and payroll files. The files are opened in the Shared mode, the new data is accepted from the Workstation, and the resource (file) is held. The record is then rewritten. The program then holds the payroll file, testing for an invalid key. If an invalid key is returned, the update to the employee file is rolled back, and the logic returns to the beginning of the transaction after prompting the user to try again. If the read and hold of the payroll file is successful, the payroll file is updated and the transaction is complete.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT EMPLOYEE-FILE
     ASSIGN TO "EMP1" "DISK"
     ORGANIZATION IS INDEXED
     ACCESS MODE IS DYNAMIC
     RECORD KEY IS   E-KEY-1.

     SELECT PAYROLL-FILE
     ASSIGN TO "PAY1" "DISK"
     ORGANIZATION IS INDEXED
     ACCESS MODE IS DYNAMIC
     RECORD KEY IS   P-KEY-1.

DATA DIVISION.
FILE SECTION.
FD  EMPLOYEE-FILE
     LABEL RECORD IS STANDARD
     RECORD CONTAINS 55 CHARACTERS
     VALUE OF FILENAME   IS "EMP1"
     LIBRARY IS "DWBS"
     VOLUME IS "ZENITH"
     RECOVERY-BLOCKS IS "A"
     RECOVERY-STATUS IS S1
     DATABASE-NAME   IS DB1.
01  EMP-REC.
     05  E-KEY-1   PIC 9(5).
 05  EMP-NAME      PIC X(25).
 05  EMP-TITLE     PIC X(25).

FD  PAYROLL-FILE
     LABEL RECORD IS STANDARD
     RECORD CONTAINS 55 CHARACTERS
     VALUE OF FILENAME   IS "PAY1"
     LIBRARY IS "DWBS"
     VOLUME IS "ZENITH"
     RECOVERY-BLOCKS IS "A"
     RECOVERY-STATUS IS S2
     DATABASE-NAME   IS DB1.
01   PAY-REC.
 05  P-KEY-1   PIC 9(5).
 05  PAY-NAME      PIC X(25).
 05  PAY-TITLE     PIC X(25).

```

Figure 3-1. Use of DMS/TX in VS COBOL

```
 WORKING-STORAGE SECTION.

 77   S1        PIC X(1).
 77   S2        PIC X(1).
 77   DB1       PIC X(6).

 01  FLAG       PIC X(3)  VALUE "   ".

 01   WS-RECORD.
  05  EMPID       PIC 9(5).
  05  EMPNAME     PIC X(25).
  05  EMPTITLE        PIC X(25).

 PROCEDURE DIVISION.
 DECLARATIVES.
 DEADLOCK-SEC SECTION.
   USE AFTER DEADLOCK.
 DLOCK-PAR.
   DISPLAY "DEADLOCK HAS OCCURRED.  PLEASE REENTER."
   GO TO TRANS-START.
 END  DECLARATIVES.

 MAIN-PROCESSING SECTION.
 BEGIN.
  OPEN SHARED EMPLOYEE-FILE
              PAYROLL-FILE.
  PERFORM TRANS-START THRU TRANS-EXIT UNTIL FLAG = "END".
  GO TO TRANS-END.

 TRANS-START.
  ACCEPT WS-RECORD.
  IF EMPID = "99999"
      MOVE "END" TO FLAG
      GO TO TRANS-EXIT.

  READ EMPLOYEE-FILE WITH HOLD
*     UPDATE EMP-REC
  REWRITE EMP-REC FROM WS-RECORD
  READ PAYROLL-FILE WITH HOLD
      INVALID KEY
          ROLLBACK
          DISPLAY "ERROR - TRY AGAIN"
          GO TO TRANS-START.
*    UPDATE PAY-REC
  REWRITE PAY-REC FROM WS-RECORD
  FREE ALL
  GO TO TRANS-START.

 TRANS-EXIT.
  EXIT.
```

Figure 3-1. Use of DMS/TX in VS COBOL (continued)

```
009000
009100 TRANS-END.
009200  CLOSE EMPLOYEE-FILE
009300        PAYROLL-FILE.
009400  STOP RUN.
```

Figure 3-1. Use of DMS/TX in VS COBOL (continued)

3.4 DMS/TX vs DMS SHARING

The following chart summarizes the functions of both DMS/TX and DMS Sharing.

DMS/TX

Syntax	Function
VALUE OF	
RECOVERY BLOCKS	Allocates Recovery blocks
RECOVERY-STATUS	Retrieves DMS/TX File Information
DATABASE-NAME	
HOLD	Requests resources (records and a range of records and/or files) to be held at once.
TIMEOUT	Used with HOLD, READ WITH HOLD, and WRITE to specify how many seconds a program will wait to acquire resources.
FOR RETRIEVAL	Used with HOLD to acquire resources for the purpose of reading only. Other programs can also read the resource simultaneously.
FOR UPDATE	Used with HOLD to acquire resources for a write, rewrite and/or delete operation. Other programs can access the resource for read without hold operations, but are denied any attempt to hold or update it.
ROLLBACK	Reverses a transaction

Syntax	Function
FREE ALL	Ends the transaction Releases the held records
DEADLOCK DECLARATIVE	A user supplied deadlock exit address which takes precedence over the system's default deadlock handling, returning control to the program.

DMS SHARING

Syntax	Function
HOLD	Requests resources (records, a range of records and/or files) to be held at once.
HOLD LIST	Adds a resource request to a list of existing requests.
HOLD EXTENSION-RIGHTS	Requests exclusive right to resources, on a claim-as-needed basis.
FREE ALL	Frees all resources immediately.
FREE EXTENSION-RIGHTS	Frees EXTENSION-RIGHTS only.
TIMEOUT	Used with HOLD, HOLD EXTENSION-RIGHTS, READ WITH HOLD, and WRITE to specify how many seconds a program will wait to acquire resources.
HOLDER-ID	Used with HOLD and HOLD EXTENSION-RIGHTS to identify the user holding resources.

Syntax	Function
FOR RETRIEVAL	Used with HOLD to acquire resources for the purpose of reading only. Other programs can also read the resource simultaneously.
FOR UPDATE	Used with HOLD to acquire resources for a write, rewrite and/or delete operation. Other programs can access the resource for read without hold operations, but are denied any attempt to hold or update it.

CHAPTER 4
WORKSTATION FILE PROCESSING

4.1 INTERACTIVE PROCESSING WITH VS COBOL

The Wang VS is an interactive system, which means that each workstation user can communicate directly with the system. For the VS COBOL programmer, such interactive communication greatly facilitates the processes of program creation, compilation and testing. The interactive capability is also useful when designing a system that requires on-line processing. The operator can query or input information to the system and get immediate response. Systems can be implemented that are operator response-driven. On the VS, these systems can be written in BASIC, RPGII, Assembler or COBOL.

This chapter describes the necessary steps for the COBOL programmer to follow in order to define, use and control the workstation. Two approaches to workstation processing are explained. The first approach uses DISPLAY AND READ, a Wang extension to COBOL, to control automatically the order and activity of information transfer. The programmer may accept all the default conditions provided by DISPLAY AND READ or override default values only for those cases where alternative processing is desired. Display characteristics, cursor position, and Field Attribute Characters are some of the workstation characteristics that are under programmer control using DISPLAY AND READ.

The second approach is more complex, since it assumes no default actions. The workstation area is treated as a record; REWRITEs are issued for screen displays from the record area, and READSs are issued for the program to transfer screen information into the record area. Whereas DISPLAY AND READ only requires fields used on the screen to be defined, the READ/REWRITE method of controlling the workstation requires either that definition of the entire screen (full screen I/O) or definition of one screen row (row-oriented I/O). Error conditions detected by DISPLAY AND READ induce automatic cursor positioning to the first field in error, as well as automatic blinking of the field; under the READ/REWRITE method the program must manually set the appropriate bytes with the required hexadecimal figurative constants. In addition, under the direct control method, the program is responsible for initializing the screen area, whereas DISPLAY AND READ automatically performs this housekeeping task.

4.2 VS INTERACTIVE EXTENSIONS

Wang has implemented extensions to the COBOL language that accommodate the responsive programming environment of the VS. These extensions are tailored to facilitate the full-screen programming capabilities of the VS workstation. The entire screen can be programmed at once, thereby allowing large blocks of information (an entire employee record, for example) to be displayed or modified at one time on the workstation.

The Procedure Division statement DISPLAY AND READ, with its affiliated phrases, facilitates programming in the VS interactive environment, providing such capabilities as controlling PF keys, positioning the cursor, sounding the workstation alarm, and transferring data between the program and the screen. DISPLAY AND READ uses the USAGE IS DISPLAY-WS screen format description, defined in Working-Storage, to format the screen.

Workstation coding requirements are grouped in this chapter according to program division.

- The Environment Division may require a Figurative-Constants paragraph if displaying nondefault attributes, controlling the cursor, or sounding the workstation alarm are desired. Each workstation file must have a FILE-CONTROL entry with a designated device-type of DISPLAY. The Environment Division requirements for DISPLAY AND READ are discussed in Subsection 4.3.1.

- The Data Division must include a File Description (FD) for the workstation file in the File Section, and USAGE IS DISPLAY-WS screen format descriptions in the Working-Storage Section. Modifying clauses are needed with these descriptions depending on the field validation requirements (for example, for range or table validation of fields). The Data Division requirements for DISPLAY AND READ are discussed in Subsection 4.3.2.

- The Procedure Division statement for controlling the workstation file is DISPLAY AND READ. The actions performed by it are dependent on those clauses coded with the USAGE IS DISPLAY-WS screen format descriptions in the Data Division. The Procedure Division requirements for DISPLAY AND READ are discussed in Subsection 4.3.3.

- An understanding of DISPLAY AND READ is all that is needed to write an interactive VS COBOL program; however, the programmer may want or need to control more workstation operations, such as setting Field Attribute Characters (via the FAC OF phrase), setting the order area (via the ORDER-AREA OF phrase), or testing a Program Function Key after a DISPLAY AND READ (via the PFKEY clause or the FILE STATUS clause). These additional Procedure Division statements are discussed in Subsection 4.3.4.

An alternative to using DISPLAY AND READ is to issue READ and REWRITE statements using the workstation file. Either full screen I/O or row-oriented I/O can be used. Full screen I/O is explained in Section 4.4; row-oriented I/O is explained in Section 4.5.

It is possible to code both DISPLAY AND READ and READ/REWRITE statements in the same program. A program may require DISPLAY AND READ for the majority of screen interactions; however, there may be an occasional use for direct control of the workstation. The rules for coexistence are described in Section 4.6.

4.3 CODING REQUIREMENTS FOR DISPLAY AND READ

An FD for the workstation file must be defined in the program. To define an FD for the workstation, code a FILE-CONTROL entry for the workstation, specifying device type as DISPLAY. In the File Section, code an FD entry corresponding to the FILE-CONTROL entry. The workstation file is coded as a consecutive file with one record of 1924 bytes: the first 4 bytes are the order area (used to control screen attributes), while the remaining 1920 bytes are the mapping area (used to display the screen).

The FD entry for the workstation is used by DISPLAY AND READ to open the workstation. Screen formatting is performed by the special USAGE IS DISPLAY-WS screen format description, specified in Working-Storage, which pass information to DISPLAY AND READ.

When a DISPLAY AND READ is issued from the COBOL program, access is automatically made to a subroutine that controls workstation processing. If the workstation is not open at the time the DISPLAY AND READ is issued, DISPLAY AND READ will automatically open the workstation. The DISPLAY AND READ subroutine is automatically included at compile time in the object program if a DISPLAY AND READ statement is encountered by the COBOL compiler. The USAGE IS DISPLAY-WS screen format description, along with DISPLAY AND READ and its associated phrases, define the screen format and the actions to be taken when the DISPLAY AND READ statement is executed.

4.3.1 Environment Division Requirements for DISPLAY AND READ

FILE-CONTROL Paragraph

If the workstation is used, either by DISPLAY AND READ or by READ/REWRITE, an FD for it must be defined within the program. Every FILE-CONTROL entry for a workstation file must include ACCESS MODE IS RANDOM and device type of DISPLAY. Since ORGANIZATION IS SEQUENTIAL is the default, the ORGANIZATION IS clause need not be coded.

The minimum Input-Output Section for a workstation is as follows.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT THE-WORKSTATION
        ASSIGN TO "SCREEN", "DISPLAY",
        ACCESS MODE IS RANDOM.
```

Other clauses can be added to the FILE-CONTROL entry for controlling the workstation, ènabling the program to control such features as cursor positioning, examining the PF key, and testing the file status after workstation I/O.

4.3.2 Data Division Requirements for DISPLAY AND READ

File Section

The workstation file must have a File Description (FD) entry. If DISPLAY AND READ is used, the screen is formatted using the USAGE IS DISPLAY-WS screen format description defined in Working-Storage. The record description entry for the workstation, although required, is not used by DISPLAY AND READ.

The USAGE IS DISPLAY-WS Screen Format Description

The screen format used by DISPLAY AND READ is specified by the USAGE IS DISPLAY-WS screen format description. Each time a DISPLAY AND READ is issued, a USAGE IS DISPLAY-WS screen format description is specified. This screen format definition is used by DISPLAY AND READ to control screen formatting and data transfer between the screen and the program data area. Each element in the USAGE IS DISPLAY-WS screen format description maps data to a particular row and column on the screen, specifying type, contents, and entry characteristics. The USAGE IS DISPLAY-WS screen format description contains modifying clauses that offer control capabilities for Procedure Division actions.

The USAGE IS DISPLAY-WS screen format description is used in the ORDER-AREA OF phrase in the Procedure Division to refer to the order area of that screen format definition. The ORDER-AREA OF phrase moves the specified figurative constant settings to the order area of the USAGE IS DISPLAY-WS screen format description to control setting of the order area before a DISPLAY AND READ. The USAGE IS DISPLAY-WS screen format description can only be referenced in the Procedure Division by the MOVE TO ORDER-AREA OF statement or by the DISPLAY AND READ statement.

Use of Data Name in a USAGE IS DISPLAY-WS Screen Format Description

A data name can be used to specify the name of a display element in the USAGE IS DISPLAY-WS screen format description. If modification of the FAC associated with the screen location is desired, specification of a data name is necessary. A figurative constant representing the desired Field Attribute Character can be moved using the FAC OF phrase in the Procedure Division. FILLER can be used if reference to the data name is not required.

The COLUMN Clause

The COLUMN clause specifies that the USAGE IS DISPLAY-WS screen element starts at the designated column on the screen. Valid values in the COLUMN clause are the integers are 1 - 80. The Field Attribute Character is mapped to the column immediately preceding the specified column; for example, the FAC of a screen element specified as starting in Column 8 is in Column 7. If a field starts in Column 1, the FAC does not actually occupy a screen location but is treated as if it were in the preceding column. Correct specification of the COLUMN clause, using two screen elements, and taking into account the column reserved for the FAC, is as follows:

```
05  FILLER     COLUMN 1      ROW 10     PIC X(7)   VALUE IS "CORRECT".
05  FILLER     COLUMN 9      ROW 10     PIC X(7)   VALUE IS "EXAMPLE".
```

The word "CORRECT" appears starting at Row 10 Column 1 of the screen. The word "CORRECT" has 7 letters; Row 10 Column 8 is reserved for the FAC of the next screen element. The word "EXAMPLE" then appears starting at Row 10 Column 9 of the screen. The following example shows the result of not taking the position for the Field Attribute Character into consideration.

```
05  FILLER     COLUMN 1      ROW 10     PIC X(9)   VALUE IS "INCORRECT".
05  FILLER     COLUMN 10     ROW 10     PIC X(7)   VALUE IS "EXAMPLE".
```

The word "INCORRECT" appears starting at Row 10 Column 1 of the screen. The word "INCORRECT" has 9 letters; the screen element for "EXAMPLE" does not take into account the FAC for the field. Therefore, the FAC for the screen element for "EXAMPLE" overlays the final "T" in "INCORRECT", producing the value "INCORREC EXAMPLE" displaying starting at Row 10 Column 1.

The ROW/LINE Clause

The ROW/LINE clause specifies that the USAGE IS DISPLAY-WS screen element starts at the designated row on the screen. Valid values for row or line number are the integers 1 - 24. Every screen element with a ROW/LINE clause must have a COLUMN clause as well; however, if a field has a COLUMN clause but no ROW/LINE clause, the element is assumed to start at the row specified in the previous screen element. For the first USAGE IS DISPLAY-WS screen element, if the ROW/LINE clause is not coded, Row 1 is assumed. If more than one USAGE IS DISPLAY-WS screen element uses the same screen position, the specification for the last element overlays all previous specifications. The following entries

```
05  FILLER     COLUMN 1      ROW 24     PIC X(13)  VALUE IS
    "NOT DISPLAYED".
05  FILLER     COLUMN 1                 PIC X(7)   VALUE IS "EXAMPLE".
```

result in the value "EXAMPLE" overlaying the value "NOT DIS" in the first screen element, producing the text "EXAMPLEPLAYED" starting at Row 24 Column 1.

The PICTURE Clause

The PICTURE clause for the USAGE IS DISPLAY-WS screen element has the same capabilities as the PICTURE clause used for an elementary data item. The PICTURE clause determines the format and length of the data as it appears on the screen.

The VALUE, SOURCE, and OBJECT Clauses

The VALUE clause and the SOURCE clause must be used independently of one another. Both specify the value to be displayed at a particular screen location. While the VALUE clause specifies a literal, the SOURCE clause specifies the contents of a particular data name to be displayed.

The OBJECT clause specifies the data name into which data is moved by DISPLAY AND READ. If an OBJECT clause is not coded, then the contents of the displayed screen field are not modifiable. SOURCE and OBJECT data names may be the same. (Moving a modifiable Field Attribute Character to the screen by using the FAC OF phrase makes the displayed screen field modifiable; however, if no OBJECT clause is coded, the modifiable data is not moved from the screen area into the program data area and it is therefore lost.)

The VALUE, SOURCE, and OBJECT clauses can only be used in certain combinations. The combination of SOURCE, OBJECT, and VALUE clauses produce different effects, such as display characteristics, default FACs, and data movement when DISPLAY AND READ reads the screen data into the program. Table 4-1 describes the effects of combinations of SOURCE, OBJECT, and VALUE clauses on screen display.

Table 4-1. Effects of VALUE, SOURCE, and OBJECT Clauses on USAGE IS DISPLAY-WS Screen Elements

SOURCE	OBJECT	VALUE	EFFECT
Yes	Yes	Yes	Not permitted.
Yes	Yes	No	Element displays at coded ROW and COLUMN with data moved from the SOURCE field to the element. A modifiable FAC is placed in the column before the element. DISPLAY AND READ moves the element to the OBJECT field.
Yes	No	Yes	Not permitted.
Yes	No	No	Element displays at coded ROW and COLUMN with data moved from the SOURCE field to the item. A dim-protected FAC is placed in the column before the element.

Table 4-1. Effects of VALUE, SOURCE, and OBJECT Clauses on USAGE IS DISPLAY-WS Screen Elements (continued)

SOURCE	OBJECT	VALUE	EFFECT
No	Yes	Yes	Element displays at coded ROW and COLUMN with VALUE literal and modifiable FAC. DISPLAY AND READ moves the element to the OBJECT field.
No	Yes	No	Element displays at coded ROW and COLUMN with pseudoblanks for the length of the element. DISPLAY AND READ moves the element to the OBJECT field.
No	No	Yes	Element displays at coded ROW and COLUMN with VALUE literal and dim-protected FAC.
No	No	No	Element cannot be displayed or modified. However a figurative constant can be moved to the FAC of the element by the FAC OF phrase.

Range Validation: The RANGE Clause

The RANGE clause provides, without programming effort, validation of data entered at the workstation during program processing. If an error occurs, the field in error blinks and the cursor is positioned to it. If more than one error occurs, all fields in error blink, and the cursor is positioned to the first field in error. Specification of a RANGE clause implies that the field is modifiable; therefore, an OBJECT clause is required when specifying a RANGE clause. Specification of a RANGE clause for a field without an OBJECT clause results in a warning message from the COBOL compiler.

The RANGE clause options for automatic data validation are as follows:

- Checking for negative values. For a numeric field, if RANGE IS NEGATIVE is specified, only values less than zero are accepted.

- Checking for positive values. For a numeric field, if RANGE IS POSITIVE is specified, only values greater than zero are accepted.

- Checking for a range of values. Range checking from one particular value to another value can be specified. The value can either be a literal or the contents of a data name. Checking is performed according to the COBOL comparison rules.

- Checking for a list of values. If RANGE IS table-name is specified, the table is searched and the field is validated if the value on the screen corresponds with a table element.

Only one validation criterion is permitted for a screen element. For example, multiple ranges, multiple table checking, or a combination of range and table lookups for a field are not allowed.

The FROM phrase is used to specify a range of values; that range being defined either by the specific literals or by data names representing the literals. Validation proceeds according to the rules for COBOL comparison operations. For example, following code validates data entered to guarantee that it is in the range "ABC" to "ABE".

```
01  SCREENREC USAGE IS DISPLAY-WS.
        03  DISPRNGE        ROW 3 COLUMN 10   PICTURE IS X(3)
        SOURCE IS FIELDA    OBJECT IS FIELDA
        RANGE  IS FROM      "ABC" TO "ABE".
```

When the DISPLAY AND READ is issued for SCREENREC, the value in FIELDA appears starting at Row 3 Column 10. Since DISPRNGE has a RANGE clause FROM "ABC" TO "ABE", the only acceptable values are "ABC", "ABD", and "ABE". If any other value is entered, SCREENREC is redisplayed with the FAC of DISPRNGE (at Row 3 Column 9) set by DISPLAY AND READ to blinking, high intensity, and modifiable. If a valid value is entered, the value is moved to FIELDA (the OBJECT IS data name) by DISPLAY AND READ.

Table Validation: The RANGE IS TABLE-NAME Clause

The RANGE IS TABLE-NAME clause allows the programmer to specify a predefined table of values. If the value entered is not a table entry, the value is rejected and the screen is redisplayed by DISPLAY AND READ with the field in error blinking with high intensity.

To specify automatic validation of a 10-byte screen field that contains one of five legitimate states (IDAHO, DELAWARE, NEW YORK, WYOMING, or OREGON), code the following table in Working-Storage:

```
01  STATE-TABLE.
    03  ENTRIES.
       05   FILLER      VALUE "IDAHO"       PICTURE IS X(10).
       05   FILLER      VALUE "DELAWARE"    PICTURE IS X(10).
       05   FILLER      VALUE "NEW YORK"    PICTURE IS X(10).
       05   FILLER      VALUE "WYOMING"     PICTURE IS X(10).
       05   FILLER      VALUE "OREGON"      PICTURE IS X(10).
    03  FILLER REDEFINES ENTRIES.
       05   TABLE-ENTRIES OCCURS 5 TIMES    PICTURE IS X(10).
```

The OBJECT field, containing the validated state name after DISPLAY AND READ successfully finds the name in the table, is coded as follows:

```
01  VALIDATED-STATE                         PICTURE IS X(10).
```

Finally, the RANGE clause may be used with the screen element STATE-VERIFY. The RANGE is the table TABLE-ENTRIES, as previously defined.

```
01  SCREENREC USAGE IS DISPLAY-WS.
    03  FILLER          ROW 3 COLUMN 8       PIC X(26)
        VALUE IS "STATE VALIDATION BY TABLE.".
    03  STATE-VERIFY    ROW 5 COLUMN 8       PIC X(10)
        OBJECT IS VALIDATED-STATE
        RANGE IS TABLE-ENTRIES.
```

The DISPLAY AND READ for SCREENREC displays 10 bytes of pseudoblanks starting in Row 5 Column 8 and waits for operator response. After the operator responds, the value (after pseudoblanks have been changed to spaces) in the field starting at Row 5 Column 8, for a length of 10 bytes, is compared with the entries in STATE-TABLE. If the value is on the table, the validated field is moved to VALIDATED-STATE (the OBJECT field); if the value is not on the table, a blinking-modifiable FAC will be moved automatically to Row 5, Column 7 and the DISPLAY AND READ redisplays SCREENREC.

Repetition of Fields: The OCCURS Clause for Screen Format Elements

The function of the OCCURS clause for screen format elements is to display and validate repeating occurrences of fields across and down the screen. Some examples of the use of the OCCURS clause for screen formatting are as follows:

- Display and validate table elements across a screen row. Figure 4-1 is the screen to be produced; Figure 4-2 is the COBOL program that produces the screen.

- Display and validate table elements down a screen row. Figure 4-3 is the screen to be produced; Figure 4-4 is the COBOL program that produces the screen.

- Display and validate table elements both across and down a screen row. Figure 4-5 is the screen to be produced; Figure 4-6 is the COBOL program that produces the screen.

- Use combinations of screen format elements containing the OCCURS clause to produce a well-formatted screen. Figure 4-7 is an example of an order entry screen; Figure 4-8 is the COBOL program that produces the screen.

Using the OCCURS Clause to Repeat Fields Across

The program illustrated in Figure 4-2 produces the screen illustrated in Figure 4-1. The screen consists of the title "FOUR FIELDS OCCURRING ACROSS" starting on Row 5, Column 26, and of four fields, each field 8 bytes in length, occurring across Row 7, starting at Column 23.

In the program illustrated in Figure 4-2, the FILE-CONTROL entry for the workstation is coded on Lines 6 - 8, while the File Description entry for the workstation is coded on Lines 11 - 13. These entries are needed by DISPLAY AND READ to open the workstation.

```
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
*  *                                                                                  *  *
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
* 5*                          FOUR FIELDS OCCURRING ACROSS                            * 5*
* 6*                                                                                  * 6*
* 7*                       ELEMENT1 ELEMENT2 ELEMENT3 ELEMENT4                        * 7*
* 8*                                                                                  * 8*
* 9*                                                                                  * 9*
*10*                                                                                  *10*
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
* 5*                                                                                  * 5*
* 6*                                                                                  * 6*
* 7*                                                                                  * 7*
* 8*                                                                                  * 8*
* 9*                                                                                  * 9*
*20*                                                                                  *20*
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
*  *                                                                                  *  *
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
```

Figure 4-1. Screen For Displaying Four Fields Across a Row

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  ACROSS.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT THE-WORKSTATION
000007         ASSIGN TO "WSFILE",   "DISPLAY"
000008         ACCESS MODE IS RANDOM.
000009 DATA DIVISION.
000010 FILE SECTION.
000011 FD  THE-WORKSTATION
000012     LABEL RECORDS ARE OMITTED.
000013 01  CRTREC  PICTURE IS X(1924).
000014 WORKING-STORAGE SECTION.
000015 01  FOUR-FIELDS  USAGE IS DISPLAY-WS.
000016     03  FILLER                       PICTURE X(28)
000017                 ROW 5    COLUMN 26
000018             VALUE IS "FOUR FIELDS OCCURRING ACROSS".
000019     03  FILLER  ROW 7.
000020         05  FILLER  OCCURS 4 TIMES   PICTURE X(8)
000021                 ROW 7    COLUMN 23
000022             SOURCE IS ELEMENT-TABLE  OBJECT IS ELEMENT-TABLE.
000023
000024 01  FILLER.
000025     03  ELEMENTS.
000026         05  FILLER  VALUE "ELEMENT1" PICTURE X(8).
000027         05  FILLER  VALUE "ELEMENT2" PICTURE X(8).
000028         05  FILLER  VALUE "ELEMENT3" PICTURE X(8).
000029         05  FILLER  VALUE "ELEMENT4" PICTURE X(8).
000030     03  ELEMENT-TABLE REDEFINES ELEMENTS
000031                     OCCURS 4 TIMES   PICTURE X(8).
000032 PROCEDURE DIVISION.
000033 DISPLAYIT.
000034     DISPLAY AND READ FOUR-FIELDS ON THE-WORKSTATION.
000035     STOP RUN.
```

Figure 4-2. Displaying Elements Across a Row

The USAGE IS DISPLAY-WS screen format description, FOUR-FIELDS, is coded on Lines 15 - 22 and consists of two screen format elements. The first element, coded on Lines 16 - 18, defines the literal "FOUR FIELDS OCCURRING ACROSS" to display starting at Row 5, Column 26, for a length of 28 bytes. The second element, coded on Lines 19 - 22, defines four fields, each having a length of 8 bytes, to display starting at Row 7, Column 23. The source of the data to be displayed is contained in the table ELEMENT-TABLE. ELEMENT-TABLE, defined on Lines 30 - 31, is a table that occurs four times, with each element 8 bytes long. The table is initialized by redefinition to the values "ELEMENT1", "ELEMENT2", "ELEMENT3", and "ELEMENT4". The initialization is accomplished by the entries on Lines 26 - 29.

The DISPLAY AND READ statement on Line 34 formats the screen according to the description of FOUR-FIELDS. The first DISPLAY AND READ statement in a program opens the workstation file THE-WORKSTATION -- an OPEN statement for THE-WORKSTATION is not needed. The four fields occurring across Row 7 each have a length of 8 bytes; however, an extra byte is reserved for the Field Attribute Character for each field. Therefore, each 8-byte table element maps onto a 9-byte screen area (the extra byte being reserved for the FAC).

Since the OBJECT field is ELEMENT-TABLE, each of the 8-byte modifiable screen fields are moved to ELEMENT-TABLE after the operator selects the ENTER key.

Using the OCCURS Clause to Repeat Fields Down

The program illustrated in Figure 4-4 produces the screen illustrated in Figure 4-3. The screen consists of the title "FOUR FIELDS OCCURRING DOWN" starting on Row 5, Column 27, and of four fields, each 10 bytes in length, occurring on 4 rows (Rows 7, 8, 9, and 10), with each occurrence starting at Column 35.

```
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
*  *                                                                                  *  *
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
* 5*                          FOUR FIELDS OCCURRING DOWN                              * 5*
* 6*                                                                                  * 6*
* 7*                                  ELEMENT1**                                      * 7*
* 8*                                  ELEMENT2**                                      * 8*
* 9*                                  ELEMENT3**                                      * 9*
*10*                                  ELEMENT4**                                      *10*
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
* 5*                                                                                  * 5*
* 6*                                                                                  * 6*
* 7*                                                                                  * 7*
* 8*                                                                                  * 8*
* 9*                                                                                  * 9*
*20*                                                                                  *20*
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*                                                                                  * 4*
*  *                                                                                  *  *
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
```

Figure 4-3. Screen For Displaying Four Fields Down a Row

The program illustrated in Figure 4-4 is coded almost exactly like the program illustrated in Figure 4-2 which produced the fields occurring across the row, with the exception that the SOURCE field is the element of the table, rather than the table itself, and that each element is initialized in the Procedure Division. The specification of the table element FIELD1 as the SOURCE field produces repetition down the screen when the DISPLAY AND READ is issued. In the Procedure Division, the paragraph INIT-TABLE, performed four times, initializes the table elements to "ELEMENT1", "ELEMENT2", "ELEMENT3", and "ELEMENT4". Then the DISPLAY AND READ is issued for the screen format OCCURS-DOWN and the fields repeat down the screen as shown in Figure 4-3.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  DOWN.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT THE-WORKSTATION
000007         ASSIGN TO "WSFILE",   "DISPLAY"
000008         ACCESS MODE IS RANDOM.
000009 DATA DIVISION.
000010 FILE SECTION.
000011 FD  THE-WORKSTATION
000012     LABEL RECORDS ARE OMITTED.
000013 01  CRTREC  PICTURE IS X(1924).
000014 WORKING-STORAGE SECTION.
000015 01  OCCURS-DOWN  USAGE IS DISPLAY-WS.
000016     03  FILLER                     PICTURE X(26)
000017                 ROW 5    COLUMN 27
000018             VALUE IS "FOUR FIELDS OCCURRING DOWN".
000019     03  FILLER  OCCURS 4 TIMES   ROW 7.
000020         05  FILLER                 PICTURE X(10)
000021                 ROW 7    COLUMN 35
000022             SOURCE IS FIELD1       OBJECT IS FIELD1.
000023
000024 01  FILLER.
000025     03  FILLER   OCCURS 4 TIMES.
000026         05  FIELD1.
000027             07  LITERAL1           PICTURE IS X(7).
000028             07  COUNTER            PICTURE IS 9.
000029 77  SUB                            PICTURE IS 9   VALUE IS 0.
000030
000031 PROCEDURE DIVISION.
000032 DISPLAYIT.
000033     PERFORM INIT-TABLE VARYING SUB FROM 1 BY 1 UNTIL SUB > 4.
000034     DISPLAY AND READ OCCURS-DOWN ON THE-WORKSTATION.
000035     STOP RUN.
000036
000037 INIT-TABLE.
000038     MOVE "ELEMENT" TO FIELD1  (SUB).
000039     MOVE SUB        TO COUNTER (SUB).
```

Figure 4-4. Displaying Elements Down a Row

Using the OCCURS Clause to Repeat Fields Across and Down

The program illustrated in Figure 4-6 produces the screen illustrated in Figure 4-5. The screen consists of the title "DISPLAYING A TABLE OCCURRING ACROSS AND DOWN" starting on Row 5, Column 17, and a 2-dimensional table (3 rows by 6 columns) starting on Row 7, Column 8. Each entry is 10 bytes in length and contains the value "ENTRY(n,m)" where n is the first occurrence number and m is the second occurrence number of the entry.

In the program illustrated in Figure 4-6, the SOURCE field is LEVEL-2, an entry of a table (TWO-LEVEL-TABLE) with two levels of OCCURS. The first level (LEVEL-1) occurs three times and indicates the number of repetitions of fields down the screen. The second level (LEVEL-2) occurs six times and indicates the number of repetitions of fields across the screen.

In the Procedure Division, the paragraph INIT-TABLE, coded on Lines 41 - 46, is performed 18 times, initializing each entry to the value "ENTRY(n,m)" using two nestings of the PERFORM VARYING statement. The DISPLAY AND READ of the USAGE IS DISPLAY-WS screen format TWO-OCCURS-LEVELS, coded on Line 37, maps the table onto the screen 6 entries across and 3 entries down, producing the screen shown in Figure 4-5.

Since TWO-LEVEL-TABLE is also an OBJECT field, after the screen is modified and the operator selects the ENTER key, each element on the screen is moved to the corresponding element on the table by the READ component of DISPLAY AND READ.

```
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
*  *                                                                                 *  *
* 1*                                                                                 * 1*
* 2*                                                                                 * 2*
* 3*                                                                                 * 3*
* 4*                                                                                 * 4*
* 5*                 DISPLAYING A TABLE OCCURRING ACROSS AND DOWN                    * 5*
* 6*                                                                                 * 6*
* 7*        ENTRY(1,1) ENTRY(1,2) ENTRY(1,3) ENTRY(1,4) ENTRY(1,5) ENTRY(1,6)        * 7*
* 8*        ENTRY(2,1) ENTRY(2,2) ENTRY(2,3) ENTRY(2,4) ENTRY(2,5) ENTRY(2,6)        * 8*
* 9*        ENTRY(3,1) ENTRY(3,2) ENTRY(3,3) ENTRY(3,4) ENTRY(3,5) ENTRY(3,6)        * 9*
*10*                                                                                 *10*
* 1*                                                                                 * 1*
* 2*                                                                                 * 2*
* 3*                                                                                 * 3*
* 4*                                                                                 * 4*
* 5*                                                                                 * 5*
* 6*                                                                                 * 6*
* 7*                                                                                 * 7*
* 8*                                                                                 * 8*
* 9*                                                                                 * 9*
*20*                                                                                 *20*
* 1*                                                                                 * 1*
* 2*                                                                                 * 2*
* 3*                                                                                 * 3*
* 4*                                                                                 * 4*
*  *                                                                                 *  *
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
```

Figure 4-5. Screen For Displaying Table Across and Down

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ACRSDOWN.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT THE-WORKSTATION
        ASSIGN TO "WSFILE",   "DISPLAY"
        ACCESS MODE IS RANDOM.
DATA DIVISION.
FILE SECTION.
FD  THE-WORKSTATION
    LABEL RECORDS ARE OMITTED.
01  CRTREC  PICTURE IS X(1924).
WORKING-STORAGE SECTION.
01  TWO-OCCURS-LEVELS  USAGE IS DISPLAY-WS.
    03  FILLER                         PICTURE X(45)
                                   ROW 5    COLUMN 17
     VALUE IS "DISPLAYING A TABLE OCCURRING ACROSS AND DOWN".
    03  FILLER  OCCURS 3 TIMES     ROW 7.
        05  FILLER  OCCURS 6 TIMES ROW 7 COLUMN 8    PIC X(10)
            SOURCE IS LEVEL-2      OBJECT IS LEVEL-2.

01  TWO-LEVEL-TABLE.
    03  LEVEL-1  OCCURS 3 TIMES.
        05  LEVEL-2 OCCURS 6 TIMES.
            07  TABLE-ENTRY.
                09  FILLER        PICTURE IS X(6).
                09  FIRST-INDEX   PICTURE IS 9.
                09  COMMA-LITERAL PICTURE IS X.
                09  SECOND-INDEX  PICTURE IS 9.
                09  RIGHT-PAREN   PICTURE IS X.
77  SUB1                          PICTURE IS 9    VALUE IS 0.
77  SUB2                          PICTURE IS 9    VALUE IS 0.
PROCEDURE DIVISION.
DISPLAYIT.
    PERFORM INIT1 VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 3.
    DISPLAY AND READ TWO-OCCURS-LEVELS ON THE-WORKSTATION.
    STOP RUN.
INIT1.
    PERFORM INIT-TABLE VARYING SUB2 FROM 1 BY 1 UNTIL SUB2 > 6.
INIT-TABLE.
    MOVE "ENTRY("   TO TABLE-ENTRY   (SUB1, SUB2).
    MOVE SUB1       TO FIRST-INDEX   (SUB1, SUB2).
    MOVE ","        TO COMMA-LITERAL (SUB1, SUB2).
    MOVE SUB2       TO SECOND-INDEX  (SUB1, SUB2).
    MOVE ")"        TO RIGHT-PAREN   (SUB1, SUB2).
```

Figure 4-6. Displaying Table Elements Across and Down

Using the OCCURS Clause for Complex Screen Formatting

The program illustrated in Figure 4-8 produces the screen illustrated in Figure 4-7. The screen is a typical order entry screen that might be required by a company's shipping department. No new concepts are introduced here; the program uses screen formatting features already discussed and illustrated. The purpose of the program is to illustrate the flexibility of VS screen formatting in COBOL. Using one Procedure Division statement, all the information required to process an order (which may consist of multiple items, terms, and shipping dates) can be formatted on the screen and automatically transferred to the program.

```
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
*  *                                                                                  *  *
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                         ORDER ENTRY SCREEN FOR ACME WIDGET COMPANY               * 3*
* 4*                                                                                  * 4*
* 5*                                                                                  * 5*
* 6*                                                                                  * 6*
* 7*           SHIP TO:   **********************************************              * 7*
* 8*                      **********************************************              * 8*
* 9*                      **********************************************              * 9*
*10*                                                                                  *10*
* 1*                                                                                  * 1*
* 2*    ITEM: ************************ TERMS:  *** *** ***      SHIP DATE:  ********  * 2*
* 3*          ************************         *** *** ***                  ********  * 3*
* 4*          ************************         *** *** ***                  ********  * 4*
* 5*          ************************         *** *** ***                  ********  * 5*
* 6*          ************************         *** *** ***                  ********  * 6*
* 7*                                                                                  * 7*
* 8*                                                                                  * 8*
* 9*                                                                                  * 9*
*20*                                                                                  *20*
* 1*                                                                                  * 1*
* 2*                                                                                  * 2*
* 3*                                                                                  * 3*
* 4*           DATE OF ORDER:  MM-DD-YY                                               * 4*
*  *                                                                                  *  *
******************************************************************************************
****          1         2         3         4         5         6         7         8 ****
**** 12345678901234567890123456789012345678901234567890123456789012345678901234567890 ****
******************************************************************************************
```

Figure 4-7. Sample Order Entry Screen

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.  WSOCCUR.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT THE-WORKSTATION
000007         ASSIGN TO "CRT",   "DISPLAY",
000008         ACCESS MODE     IS RANDOM.
000009 DATA DIVISION.
000010 FILE SECTION.
000011 FD  THE-WORKSTATION
000012     LABEL RECORDS ARE OMITTED.
000013 01  CRTREC                          PIC X(1924).
000014 WORKING-STORAGE SECTION.
000015*
000016 77  SHIP-NAME      PICTURE IS X(46)       VALUE SPACES.
000017 77  SHIP-CITY      PICTURE IS X(46)       VALUE SPACES.
000018 77  SHIP-STATE     PICTURE IS X(46)       VALUE SPACES.
000019 77  ORDER-DATE     PICTURE IS X(08)       VALUE SPACES.
000020*
000021 01  THE-TABLE.
000022     05  ITEM-NAME  PICTURE IS X(24)       OCCURS 5 TIMES.
000023     05  FILLER  OCCURS 5 TIMES.
000024         07  TERMS  PICTURE IS 9(3)        OCCURS 3 TIMES.
000025     05  SHIP-DATE  PICTURE IS X(8)        OCCURS 5 TIMES.
000026*
000027 01  DISPLAY-REC USAGE IS DISPLAY-WS.
000028     05  FILLER         PICTURE IS  X(42)       ROW 03 COLUMN 25
000029         VALUE IS "ORDER ENTRY SCREEN FOR ACME WIDGET COMPANY".
000030     05  FILLER         PICTURE IS  X(08)       ROW 07 COLUMN 11
000031         VALUE IS "SHIP TO:".
000032     05  ROW07-COL22    PICTURE IS  X(46)       ROW 07 COLUMN 22
000033         SOURCE IS SHIP-NAME      OBJECT IS SHIP-NAME     .
000034     05  ROW08-COL22    PICTURE IS  X(46)       ROW 08 COLUMN 22
000035         SOURCE IS SHIP-CITY      OBJECT IS SHIP-CITY     .
000036     05  ROW09-COL22    PICTURE IS  X(46)       ROW 09 COLUMN 22
000037         SOURCE IS SHIP-STATE     OBJECT IS SHIP-STATE    .
000038     05  FILLER         PICTURE IS  X(05)       ROW 12 COLUMN 04
000039         VALUE IS "ITEM:".
000040     05  FILLER         PICTURE IS  X(06)       ROW 12 COLUMN 35
000041         VALUE IS "TERMS:".
000042     05  FILLER         PICTURE IS  X(10)       ROW 12 COLUMN 60
000043         VALUE IS "SHIP DATE:".
000044*
000045     05  FILLER OCCURS 5 TIMES ROW 12.
000046         07  FILLER     PICTURE IS  X(24)       ROW 12 COLUMN 10
000047         SOURCE IS ITEM-NAME      OBJECT IS ITEM-NAME.
000048         07  FILLER     PICTURE IS  9(03)       ROW 12 COLUMN 43
000049                                  OCCURS 3 TIMES
```

Figure 4-8. Producing Sample Order Entry Screen

```
000050         SOURCE IS TERMS         OBJECT IS TERMS.
000051         07  FILLER     PICTURE IS  X(08)       ROW 12 COLUMN 72
000052         SOURCE IS SHIP-DATE     OBJECT IS SHIP-DATE.
000053*
000054      05 FILLER         PICTURE IS  X(14)       ROW 24 COLUMN 12
000055         VALUE IS "DATE OF ORDER:".
000056      05 FILLER         PICTURE IS X(08)        ROW 24 COLUMN 28
000057                                 OBJECT IS ORDER-DATE
000058         VALUE IS "MM-DD-YY".
000059 PROCEDURE DIVISION.
000060 START-PROGRAM.
000061      MOVE SPACES TO THE-TABLE.
000062      DISPLAY AND READ DISPLAY-REC ON THE-WORKSTATION.
000063      STOP RUN.
```

Figure 4-8. Producing Sample Order Entry Screen (continued)

Up to three levels of OCCURS are allowed in USAGE IS DISPLAY-WS screen definitions. These screen tables may have as SOURCE or OBJECT fields a table defined in Working-Storage. Each element of a source table is moved to the screen table so that the index of the table element corresponds to the index of the screen element. Thus, the dimensions of the source and object tables must match with the dimensions of the USAGE IS DISPLAY-WS screen table, as shown in the previous illustrations.

Field Attribute Characters

When coding the screen definition entry, the programmer should be familiar with the rules related to the Field Attribute Characters. Refer to Appendix C for a detailed explanation of the rules for Field Attribute Characters.

If DISPLAY AND READ is used, a FAC is automatically provided for each screen format element depending upon what clauses are associated with it. If a screen format element has no OBJECT clause, then it is not modifiable, and the default FAC used is a hexadecimal "8C" (dim and protected). If a screen format element has an OBJECT clause, then it is modifiable. The default FAC allows for bright intensity with entry characteristics corresponding to the picture of the screen format element. If the screen format element has a numeric picture, the default FAC used is a hexadecimal "82" (bright, modifiable, and numeric-only input allowed); if the screen format element has an alphanumeric picture, the default FAC used is a hexadecimal "81" (bright, modifiable, and uppercase only input allowed).

Since the default FAC for an alphanumeric screen format element allows only uppercase input, to allow both uppercase and lowercase input, the programmer must construct a figurative constant allowing both uppercase and lowercase input and move the figurative constant to the FAC of the screen format item using the FAC OF phrase. A FAC of hexadecimal "80" allows both uppercase and lowercase input.

If an error is detected on a screen format item by DISPLAY AND READ (for example, a RANGE validation is violated), the field in error blinks. DISPLAY AND READ automatically sets the blink bit in the FAC of the field in error and redisplays the screen format.

4.3.3 Procedure Division Requirements for DISPLAY AND READ

The events occurring during a DISPLAY AND READ are categorized into the following five steps:

1. Format the screen.
2. Rewrite the screen.
3. Read the operator response.
4. Validate the data.
5. Transfer the data into the object field(s).

The following operations occur automatically with one invocation of DISPLAY AND READ:

Step 1 -- Format the Screen

When the workstation screen is formatted, the literal values specified in the VALUE clause and the values assigned to the data names identified in the SOURCE clause for the USAGE IS DISPLAY-WS screen format elements are moved to the workstation screen.

The screen formatting process moves one of the following to the screen:

- If the screen format element has a VALUE clause, the VALUE IS literal is moved.
- If the screen format element has a SOURCE clause, the contents of the source field is moved.
- If the screen format element has an OBJECT clause but no SOURCE clause, pseudoblanks are moved.
- Default FACs are set for the screen format element.

Step 2 -- Rewrite the Screen

After the screen record is formatted DISPLAY AND READ issues a REWRITE to display the screen.

Step 3 -- Read the Operator Response

DISPLAY AND READ issues a READ, which waits for operator response. The operator responds by entering data in any modifiable field(s) and selecting any of the enabled PF keys or the ENTER key (if ENTER is enabled). If the PFKEYS option of DISPLAY AND READ is not specified, only the ENTER key is valid; selecting any other PF key results in rewriting the DISPLAY AND READ screen and sounding the workstation alarm. IF the PFKEYS option of DISPLAY AND READ is specified, those PF keys specified in the PFKEYS option are valid; selecting any other PF key results in rewriting the DISPLAY AND READ screen and sounding the workstation alarm. DISPLAY AND READ will not proceed unless an enabled PF key or the ENTER key (if ENTER is enabled) is selected.

The READ step next determines which PF key (or the ENTER KEY) was selected. The ONLY phrase of DISPLAY AND READ is used to specify valid PF key responses. If ONLY PFKEYS 1, 2, 3, 16 is coded, then only PF Keys 1, 2, 3, and 16 are acceptable responses. However, if PFKEYS 1, 2, 3, 16 is coded (the ONLY phrase without the keyword ONLY), then PF Keys 1, 2, 3, 16 and the ENTER key are acceptable responses.

If an ON phrase has been specified for any PF key, the READ step causes transfer of control to the imperative statement associated with the ON phrase after DISPLAY AND READ has determined that the PF key associated with the ON phrase has been selected.

Step 4 -- Validate the Data

The response is validated to check agreement of the data type entered by the operator with the usage defined in the PICTURE clause for the modified field. If a RANGE clause is coded, the entered data is also tested for the legal range limit; if a RANGE IS table-name clause is coded, the data is tested for a value in the table. If invalid data is entered, DISPLAY AND READ automatically returns control to the rewrite step (Step 2) and the cycle repeats until valid data is entered. When the screen containing invalid data is rewritten, the workstation alarm sounds and the FACs of those invalid fields are changed to blink so the field(s) in error can be identified. In addition, the cursor is automatically positioned at the first field in error.

Step 5 -- Transfer the Data Into the Object Fields.

If the data is successfully validated, all modifiable fields are moved from the screen area to OBJECT field. The transfer of data is in accordance with the rules of the MOVE statement. The ALTERED option of DISPLAY AND READ moves only those fields that have been changed and is recommended if reducing the amount of data transfer from the screen area to the program data area is a consideration.

DISPLAY AND READ Options

DISPLAY AND READ provides several options for enabling/disabling PF key(s) and the ENTER key, and for program-controlled actions based on the selected PF key. These options are:

1. DISPLAY AND READ DISPLAY-REC ON SCREEN.

   A DISPLAY AND READ with no options displays the USAGE IS DISPLAY-WS screen format description DISPLAY-REC on the workstation. The ENTER key is enabled; if any PF key is selected, the workstation alarm sounds and DISPLAY-REC is redisplayed.

2. DISPLAY AND READ DISPLAY-REC ON SCREEN
    PFKEY 16.

   This statement displays DISPLAY-REC on the workstation. The PFKEY phrase enables PF 16 as well as the ENTER key; if any other key is selected, the workstation alarm sounds and DISPLAY-REC is redisplayed.

3. DISPLAY AND READ DISPLAY-REC ON SCREEN
    ONLY PFKEY 16.

   This statement displays DISPLAY-REC on the workstation. The specification of the ONLY PFKEY phrase specifies that the ENTER key is <u>disabled</u> and that <u>only</u> PF16 is enabled. Selecting any PF key other than PF16 causes the workstation alarm to sound and DISPLAY-REC to be redisplayed.

4. DISPLAY AND READ DISPLAY-REC ON SCREEN
    PFKEY 16
    ON PFKEY 16 DISPLAY "PF16 has been selected.".

   This statement displays DISPLAY-REC on the workstation. The PFKEY 16 phrase, coded as in case 2, enables the ENTER key and PF16. The specification of the ON PFKEY 16 phrase causes automatic transfer of control to the imperative statement associated with the ON phrase. Therefore, if PF16 is selected, the message "PF16 has been selected." displays. Selecting the ENTER key causes DISPLAY AND READ to continue with data validation and transfer; selecting any other PF key causes the workstation alarm to sound and DISPLAY-REC to be redisplayed.

   The ON phrase specifies an immediate action to be taken if the PF key is selected. This action overrides data transfer of screen fields to any specified object fields in the USAGE IS DISPLAY-WS screen format.

5. DISPLAY AND READ ALTERED DISPLAY-REC ON SCREEN
```
    PFKEY 16
    NO-MOD DISPLAY "No screen fields have been modified.".
```

   This statement displays DISPLAY-REC on the workstation and enables the ENTER key and PF16 (as in Case 2 and Case 4). The ALTERED option of DISPLAY AND READ indicates that only those screen fields modified by the operator are transferred from the screen to the OBJECT field. The ALTERED option of DISPLAY AND READ significantly reduces the amount of screen data to be transferred to the program, since only screen items with OBJECT fields that have been altered (changed) will be affected.

   The NO-MOD phrase is a program-defined action to be taken if no screen fields have been modified. In the previous example, if no fields have been modified, the message "No screen fields have been modified." displays.

Under DISPLAY AND READ ALTERED, VS COBOL provides a method of testing whether a particular display item has been changed. This method uses the following statement:

```
IF FAC OF display-item ALTERED imperative-statement
```

Thus, the particular field(s) that have been changed can be tested and program-defined actions based on the alteration of the field can be implemented.

DISPLAY AND READ phrases have the following precedence rules:

1. If no DISPLAY AND READ options are coded, then execution falls through to the next statement after an enabled PF key is selected.

2. If the ON phrase is coded, execution passes to the statement indicated by the ON imperative statement and no transfer of data occurs, even if a field has been modified. For example, if the following statement is coded.

   ```
   DISPLAY AND READ DISPLAY-REC ON SCREEN
    ONLY PFKEY 2, 5
    ON   PFKEY 2 GO TO 100-EXIT.
   ```

   selection of PF2 passes control to paragraph 100-EXIT, selection of PF5 (the only other valid PF key) allows DISPLAY AND READ to validate the screen data and transfer it to the program, and selection of any other PF key causes the workstation alarm to sound.

3. If the NO-MOD phrase is coded for a DISPLAY AND READ ALTERED, then execution is passed to the NO-MOD imperative statement only when no modification of displayed data has occurred. Although ALTERED must be coded if the NO-MOD phrase is coded, ALTERED can be coded without using the NO-MOD phrase.

4. If both the NO-MOD phrase and the ON phrase are coded, and no modifications are made but a PF key associated with the ON phrase is selected, the ON phrase has precedence and is executed. For example, if the following statement

```
DISPLAY AND READ ALTERED DISPLAY-REC ON SCREEN
 ONLY PFKEY 2, 4
 ON PFKEY 2 GO TO 100-EXIT
 NO-MOD GO TO 200-NEXT-LEVEL.
```

   is executed, with no modifications made to modifiable screen fields, and PF 2 is selected, then control passes to 100-EXIT (the imperative statement associated with the ON phrase) rather than to 200-NEXT-LEVEL (the imperative statement associated with the NO-MOD phrase).

5. RANGE validation is performed before PF key validation, except that PF key validation associated with the ON phrase is performed immediately. Therefore, if a enabled PF key with no associated ON phrase is selected, and at least one screen field has violated a RANGE check, the field(s) in error blink and the screen is redisplayed.

   Therefore, any PF key that is enabled but has no ON PFKEY exit is not honored unless all of the screen fields pass the RANGE checks. If the PF key is enabled and has an associated ON phrase, the imperative statement of the ON phrase will be taken. If nothing has been modified, even if one of the fields violates a RANGE check, the NO-MOD imperative statement is executed.

4.3.4 Coding Requirements for Additional Workstation File Control

All of the capabilities discussed thus far occur automatically and give the programmer enough control over the workstation to handle most application situations. If, however, the programmer wishes to exercise more control in conjunction with the DISPLAY AND READ facilities, the option does exist. These additional programming capabilities allow the programmer to:

- Specify and test the Field Attribute Characters of any field that is displayed on the workstation screen, implemented by the FAC OF phrase.

- Specify and test order area bits for controlling keyboard locking, alarm sounding, and cursor positioning, implemented by the ORDER-AREA OF phrase.

- Determine location of cursor, implemented by examining the CURSOR POSITION IS data name as specified in the workstation FILE-CONTROL entry.

- Test user's PF key response, implemented by examining either the PFKEY IS data name or the FILE STATUS IS data name as specified in the workstation FILE-CONTROL entry.

Control of Field Attribute Characters Using the FAC OF Phrase

Field Attribute Characters, as described in Subsection 4.3.2, define the attributes for each displayed field. The programmer can control the characteristics of fields displayed on the workstation screen by manipulating FAC values and thereby not accept the defaults provided by DISPLAY AND READ. Each FAC is a 1-byte character that can be altered by moving a new value to it. The values must be provided in hexadecimal and must be identified in the Environment Division as a figurative constant. For an extensive discussion of Field Attribute Characters, refer to Appendix C.

For example, to define a Field Attribute Character for a protected field, displaying with bright intensity, a possible value is "86". The FIGURATIVE-CONSTANTS paragraph would read:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
FIGURATIVE-CONSTANTS.
   PRO-BRITE IS "86".
```

Assume the following USAGE IS DISPLAY-WS definition:

```
01 DISPLAY-REC USAGE IS DISPLAY-WS.
   05   DISP-FIELD    PIC 9(5)  ROW 10   COLUMN 7.
```

PRO-BRITE characteristics could be given to DISP-FIELD by moving PRO-BRITE to the figurative constant associated with DISP-FIELD. This is accomplished by coding the following statement in the Procedure Division:

```
MOVE PRO-BRITE TO FAC OF DISP-FIELD.
```

A FAC of a displayed data item can also be tested in an IF statement. For example, to test whether DISP-FIELD has the PRO-BRITE FAC, code the following:

```
IF FAC OF DISP-FIELD = PRO-BRITE
    DISPLAY "This field is bright and protected.".
```

A FAC can also be tested to see if the field it describes had been altered by a user responding to DISPLAY AND READ with the ALTERED option. For example, to test whether the screen area associated with DISP-FIELD had been altered by the previous DISPLAY AND READ, code the following:

```
IF FAC OF DISP-FIELD ALTERED
    THEN DISPLAY "This field has been changed.".
```

Control of the Order Area Using The ORDER-AREA OF Phrase

The order area is a 4-byte control area for the workstation and can, if necessary, be manipulated under program control. The first byte of the order area contains the row number at which screen processing is to begin. The second byte is the Write Control Character (WCC) that controls keyboard locking, alarm sounding, and cursor positioning. The last two bytes contain the cursor column number and cursor row number, respectively, after a READ; in addition, if the "position cursor" bit is on in the WCC, these bytes indicate at what column and row the cursor is to be positioned. Refer to Appendix D for a detailed discussion of the order area.

The order area values may be provided or tested under program control. To manipulate these hexadecimal values, the programmer can move the appropriate figurative constants to the order area bytes.

The USAGE IS DISPLAY-WS screen format description is used in the ORDER-AREA OF phrase as the order area of the screen. Each USAGE IS DISPLAY-WS screen format description has, in effect, its own order area, which is mapped onto the workstation order area when the DISPLAY AND READ is issued. To illustrate, assume the following screen format description:

```
01  DISPLAY-REC  USAGE IS DISPLAY-WS.
```

A 4-byte group item ORDERAREA, composed of four 1-byte elementary items, defines the actual order area. This is coded:

```
01  ORDERAREA.
    03  ROW-NUMBER               PICTURE IS X.
    03  WRITE-CONTROL-CHARACTER  PICTURE IS X.
    03  CURSOR-COLUMN-ADDRESS    PICTURE IS X.
    03  CURSOR-ROW-ADDRESS       PICTURE IS X.
```

After the appropriate figurative constants have been moved to the elementary items composing ORDERAREA, the order area of the USAGE IS DISPLAY-WS screen format description is set to the desired values by coding the following:

```
MOVE ORDERAREA TO ORDER-AREA OF DISPLAY-REC.
```

Figure 4-9 is a complete COBOL program illustrating controlling the cursor and sounding the workstation alarm using the ORDER-AREA OF phrase. The figurative constants POSITION-CURSOR, SOUND-THE-ALARM, ONE, TWENTY-FOUR, and EIGHTY, coded on Lines 4 - 9, define the hexadecimal values that are to be moved into the order area for the appropriate USAGE IS DISPLAY-WS screen format. The order area to be initialized (ORDERAREA) is a 4-byte group item consisting of four 1-byte elementary items, which will be initialized in the Procedure Division by moving in figurative constants.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ORDRAREA.
ENVIRONMENT DIVISION.
FIGURATIVE-CONSTANTS.
    POSITION-CURSOR   IS "A0",
    SOUND-THE-ALARM   IS "C0",
    ONE               IS "01",
    TWENTY-FOUR       IS "18",
    EIGHTY            IS "50".
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SCREEN ASSIGN TO "SCREEN", "DISPLAY",
        ACCESS MODE     IS RANDOM.
DATA DIVISION.
FILE SECTION.
FD  SCREEN
    LABEL RECORDS ARE STANDARD.
01  CRTREC   PICTURE X(1924).
WORKING-STORAGE SECTION.
01  CURSOR-CONTROL USAGE IS DISPLAY-WS.
    05  FILLER         PICTURE IS  X(35)       ROW 04 COLUMN 08
        VALUE IS "The cursor is at row 24, column 80.".
01  ALARM-SCREEN USAGE IS DISPLAY-WS.
    05  FILLER         PICTURE IS  X(34)       ROW 04 COLUMN 08
        VALUE IS "The workstation alarm has sounded.".
01  ORDERAREA.
    03  ROW-NUMBER              PICTURE IS X.
    03  WRITE-CONTROL-CHARACTER PICTURE IS X.
    03  CURSOR-COLUMN-ADDRESS   PICTURE IS X.
    03  CURSOR-ROW-ADDRESS      PICTURE IS X.
PROCEDURE DIVISION.
CONTROL-THE-CURSOR.
    MOVE ONE                TO ROW-NUMBER.
    MOVE POSITION-CURSOR    TO WRITE-CONTROL-CHARACTER.
    MOVE EIGHTY             TO CURSOR-COLUMN-ADDRESS.
    MOVE TWENTY-FOUR        TO CURSOR-ROW-ADDRESS.
    MOVE ORDERAREA          TO ORDER-AREA OF CURSOR-CONTROL.
    DISPLAY AND READ CURSOR-CONTROL ON SCREEN.
SOUND-ALARM.
    MOVE SOUND-THE-ALARM    TO WRITE-CONTROL-CHARACTER.
    MOVE ORDERAREA          TO ORDER-AREA OF ALARM-SCREEN.
    DISPLAY AND READ ALARM-SCREEN    ON SCREEN.
    CLOSE SCREEN.
    STOP RUN.
```

Figure 4-9. Control of Order Area Using the ORDER-AREA OF Phrase

In the paragraph CONTROL-THE-CURSOR, coded on Lines 32 - 38, ORDERAREA is set as follows:

1. ROW-NUMBER is set to a hexadecimal figurative constant ONE ("01") by the statement MOVE ONE TO ROW-NUMBER, coded on Line 33. This setting of the row number instructs DISPLAY AND READ to display the entire screen, starting at Row 1 (the default action of DISPLAY AND READ).

2. WRITE-CONTROL-CHARACTER is set to the hexadecimal figurative constant POSITION-CURSOR ("A0") by the statement MOVE POSITION-CURSOR TO WRITE-CONTROL-CHARACTER, coded on Line 34. This setting of the Write Control Character instructs DISPLAY AND READ to position the cursor to the column and row address specified in the next 2 bytes of the order area.

3. CURSOR-COLUMN-ADDRESS is set to the hexadecimal figurative-constant EIGHTY ("50") by the statement MOVE EIGHTY TO CURSOR-COLUMN-ADDRESS, coded on Line 35. The value "50" in hexadecimal is "80" in decimal; therefore the setting of hexadecimal 50 in CURSOR-COLUMN-ADDRESS will define a column number of 80.

4. CURSOR-ROW-ADDRESS is set to the hexadecimal figurative-constant TWENTY-FOUR ("18") by the statement MOVE TWENTY-FOUR TO CURSOR-ROW-ADDRESS, coded on Line 36. The value "18" in hexadecimal is "24" in decimal; therefore, the setting of hexadecimal 18 in CURSOR-ROW-ADDRESS defines a row number of 24.

Finally, the initialized order area (ORDERAREA) is moved to the order area of the USAGE IS DISPLAY-WS format definition CURSOR-CONTROL by the MOVE statement using the ORDER-AREA OF phrase, coded on Line 37. The DISPLAY AND READ of CURSOR-CONTROL, coded on Line 38, sets the cursor at Row 24, Column 80.

The workstation alarm is sounded in the paragraph SOUND-THE-ALARM as follows:

1. WRITE-CONTROL-CHARACTER is set to the hexadecimal figurative constant SOUND-THE-ALARM ("C0") by the statement MOVE SOUND-THE-ALARM TO WRITE-CONTROL-CHARACTER, coded on Line 40.

2. ORDERAREA is moved to the USAGE IS DISPLAY-WS screen format definition ALARM-SCREEN by the statement MOVE ORDERAREA TO ORDER-AREA OF ALARM-SCREEN, coded on Line 41.

3. The DISPLAY AND READ of ALARM-SCREEN, coded on Line 42, causes the alarm the alarm to sound. DISPLAY AND READ controls the screen based on the Write Control Character; in this case, the bit associated with sounding the alarm is set.

Determining Cursor Position Using the CURSOR POSITION IS Clause

As illustrated in Figure 4-9, setting the cursor on the screen can be controlled through the ORDER-AREA OF phrase. However, determining the position of the cursor after a DISPLAY AND READ can be accomplished by a special clause in the FILE-CONTROL entry for the workstation. This clause is the CURSOR POSITION IS clause.

The CURSOR POSITION IS clause in the FILE-CONTROL entry for the workstation can be used to define a data name having the value of the cursor column and the cursor row after the READ option of DISPLAY AND READ. To define a data name called CURSOR-POS to receive the cursor position value after a READ, code the following in the FILE-CONTROL entry for the workstation:

```
    CURSOR POSITION IS CURSOR-POS
```

CURSOR-POS is a group item composed of two elementary items, defined in the Data Division as follows:

```
    01  CURSOR-POS.
         03  COLUMN-SETTING              BINARY.
         03  ROW-SETTING                 BINARY.
```

COLUMN-SETTING contains the cursor column number after a READ; ROW-SETTING contains the cursor row number after a READ.

The CURSOR POSITION IS data name is assigned a value by the operating system when a DISPLAY AND READ is issued. The value is in the form of two 2-byte binary data items; the first value corresponds to a cursor column location and the second corresponds to a cursor row location. Valid column values are 1 - 80 (inclusive) and valid row values are 1 - 24 (inclusive). This clause only permits reading the current cursor position. Setting the cursor to another position can only be accomplished through the ORDER-AREA OF phrase, as illustrated in Figure 4-9.

Testing PF Key Response Using the PFKEY or the FILE STATUS Clauses

Two special clauses in the FILE-CONTROL entry for the workstation are used to direct action based upon user PF key response. They are the PFKEY and FILE STATUS clauses.

The PF key can be tested by the PFKEY IS clause in the FILE-CONTROL entry for the workstation file. To code a PFKEY clause, code the following in the FILE-CONTROL entry for the workstation file:

```
     PFKEY IS PF-KEY
```

In Working-Storage code the following:

```
     01  PF-KEY        PIC 99.
```

The PFKEY IS data name (PF-KEY) is a 2-character numeric field that receives the numeric value of the selected PFKEY following execution of a DISPLAY AND READ statement or of a workstation READ statement.

PF-KEY can be tested in the Procedure Division to initiate action based on selection of a particular PF key. After the READ function of DISPLAY AND READ, the value of the PF key selected is stored in the PFKEY IS data name. A number between 0 and 32 is stored, with 0 representing the ENTER key and 1 through 32 representing the corresponding PF key. For example, to perform a routine called CALCULATION based on selection of the ENTER key, code the following:

```
    IF PF-KEY = 0   PERFORM CALCULATION.
```

The PF key can also be tested after a workstation READ by examining the workstation's file status. To define a FILE STATUS clause, code in the FILE-CONTROL entry for the workstation as follows:

```
    FILE STATUS IS FILE-STATUS
```

In Working-Storage code:

```
    01  FILE-STATUS.
         03  STATUS-BYTE-1          PICTURE IS X.
         03  PFK-BYTE               PICTURE IS X.
```

The second byte of the data item associated with the FILE STATUS clause holds the value corresponding to the PF key of the operator's response after a workstation READ. The rightmost character, PFK-BYTE, will contain "@" if the ENTER key is selected, an uppercase letter in the range A through P if one of the PF keys between 1 and 16 is selected, or a lowercase letter in the range a through p if one of the PF keys between 17 and 32 is selected.

FILE-STATUS can be tested in the Procedure Division to initiate action based on selection of a particular PF key. For example, to perform a routine called CALCULATION based on selection of the ENTER key being, code the following:

```
    IF PFK-BYTE = "@" PERFORM CALCULATION.
```

The PFKEY IS clause and the FILE STATUS clause thus perform equivalent functions (testing the PF key after a DISPLAY AND READ or a workstation READ statement). However, use of the PFKEY IS clause is recommended because the value returned in the data item directly corresponds to the number of the PF key selected (with 0 representing the ENTER key); whereas the value returned in the FILE STATUS data name is a letter returned in the second byte, and the <u>letter</u> must be translated into the PF key <u>number</u>. The letter, called the AID character, is discussed in detail in Appendix E. Using the FILE STATUS method makes the code more difficult to read because the programmer must be conscious of which letter in the FILE STATUS data name corresponds to which PF key number.

In summary, the three methods of testing for a PF key value are:

1. Use of the ON PFKEY phrase of DISPLAY AND READ.

2. Testing the PFKEY IS data name after a workstation READ.

3. Testing the second byte of the FILE STATUS data name (after translating the letter into a PF key number) after a workstation READ.

4.4 PROGRAMMING THE WORKSTATION THROUGH FULL SCREEN I/O

The workstation can be treated as a file, affording the programmer direct control of the screen area. The workstation is viewed as a consecutive file in random access mode, each row being one record of the file.

Figure 4-10 is a complete COBOL program illustrating the technique of direct file processing of the workstation, using full screen I/O. The program produces the screen (four fields occurring across) as shown in Figure 4-1. The same screen is produced by the COBOL program illustrated in Figure 4-2, but whereas that program used DISPLAY AND READ to produce the screen, this program uses full screen I/O. A comparison of the programs illustrated in Figure 4-2 and Figure 4-10 demonstrates the coding differences between DISPLAY AND READ and full screen I/O.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ACROSS.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
FIGURATIVE-CONSTANTS.
    POSITION-CURSOR IS "A0",
    MODCHR          IS "81",
    DIM             IS "8C".
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT THE-WORKSTATION
        ASSIGN TO "WSFILE",   "DISPLAY"
        RELATIVE KEY IS ROW-NUMBER
        ACCESS MODE IS RANDOM.
DATA DIVISION.
FILE SECTION.
FD  THE-WORKSTATION
    LABEL RECORDS ARE OMITTED.
01  CRTREC.
    03  SCREEN-ORDER-AREA.
        05  ROWNUMBER                PICTURE X.
        05  WCC                      PICTURE X.
        05  CURSOR-COLUMN            PICTURE X.
        05  CURSOR-ROW               PICTURE X.
```

Figure 4-10. Displaying Elements Across a Row Using Full Screen I/O

```
000025     03  SCREEN-MAPPING-AREA         PICTURE X(1920).
000026 WORKING-STORAGE SECTION.
000027 77  ROW-NUMBER                      PICTURE 9(2)   VALUE 1.
000028 01  FOUR-FIELDS.
000029     03  ROWS-1-THRU-4               PICTURE X(320) VALUE SPACES.
000030     03  ROW-5.
000031         05  FILLER                  PICTURE X(25)  VALUE SPACES.
000032         05  FILLER                  PICTURE X(28)
000033             VALUE IS "FOUR FIELDS OCCURRING ACROSS".
000034         05  REST-OF-ROW-5           PICTURE X(27)  VALUE SPACES.
000035     03  ROW-6                       PICTURE X(80)  VALUE SPACES.
000036     03  ROW-7.
000037         05  FILLER                  PICTURE X(21)  VALUE SPACES.
000038         05  FILLER                  PICTURE X      VALUE MODCHR.
000039         05  FIELD1  VALUE "ELEMENT1" PICTURE X(8).
000040         05  FILLER                  PICTURE X      VALUE MODCHR.
000041         05  FIELD2  VALUE "ELEMENT2" PICTURE X(8).
000042         05  FILLER                  PICTURE X      VALUE MODCHR.
000043         05  FIELD3  VALUE "ELEMENT3" PICTURE X(8).
000044         05  FILLER                  PICTURE X      VALUE MODCHR.
000045         05  FIELD4  VALUE "ELEMENT4" PICTURE X(8).
000046         05  FILLER                  PICTURE X      VALUE DIM.
000046 PROCEDURE DIVISION.
000047 OPEN-THE-WORKSTATION.
000048     OPEN I-O THE-WORKSTATION.
000049 INITIALIZE-THE-ORDER-AREA.
000050     MOVE POSITION-CURSOR TO WCC.
000051     MOVE LOW-VALUES      TO CURSOR-COLUMN CURSOR-ROW.
000052 DISPLAY-THE-SCREEN.
000053     MOVE FOUR-FIELDS TO SCREEN-MAPPING-AREA.
000054     REWRITE CRTREC.
000055 READ-THE-SCREEN.
000056     READ THE-WORKSTATION MODIFIABLE.
000057 MOVE-TO-WORKING-STORAGE.
000058     MOVE SCREEN-MAPPING-AREA TO FOUR-FIELDS.
000059     CLOSE THE-WORKSTATION.
000060     STOP RUN.
```

Figure 4-10. Displaying Elements Across a Row
Using Full Screen I/O (continued)

Comparing the program illustrated in Figure 4-10 with the program illustrated in Figure 4-2, the differences between full screen I/O and DISPLAY AND READ are:

1. The order area and the mapping area of the workstation must be initialized. The record description entry for the workstation (CRTREC), coded on Lines 18 - 24, defines the 4-byte order area as well as the 1920-byte mapping area for the screen. The order area (SCREEN-ORDER-AREA) is initialized by moving appropriate figurative constants to the individual order area bytes. The mapping area (SCREEN-MAPPING-AREA) is initialized by moving the screen description.

2. The entire screen must be initialized. Whereas using DISPLAY AND READ only those screen areas that are to be used need to be defined, using full screen I/O the screen is treated as a record; unused areas must be defined as FILLER and initialized to SPACES. In Working-Storage, the group item FOUR-FIELDS, coded on Lines 27 - 45, defines the screen format. The following coding requirements should be noted.

   a. Even though the first field (the literal "FOUR FIELDS OCCURRING ACROSS") appears on Row 5 Column 26, the previous 345 bytes are initialized to spaces by definition of the data names ROWS-1-THRU-4 and FILLER on Lines 28 and 30.

   b. All Field Attribute Characters (except for the dim-protected FAC which is the default for the beginning of every screen row) must be defined and initialized. The four elements occurring across as well as their associated FACs, which in the program illustrated in Figure 4-2 were defined by one screen format item definition, are defined here using 9 coding lines (Lines 37 - 45). The Field Attribute Character MODCHR, defined as a figurative constant on Line 6 as a bright, modifiable field allowing uppercase input, must be moved to the byte immediately before each element. In addition, at the end of the last element, a "trailing FAC" must be defined as dim-protected (DIM), to prevent the last screen field from being modifiable to the end of the screen row.

3. The workstation must be opened explicitly in the Procedure Division. This is accomplished by the OPEN statement coded on Line 48. The program in Figure 4-2 was not required to open the workstation explicitly, since DISPLAY AND READ automatically opens the workstation.

4. The order area must be initialized in the Procedure Division. This is accomplished in the paragraph INITIALIZE-THE-ORDER-AREA, coded on Lines 49 - 51. The figurative constants used position the cursor to the first modifiable field by setting the "position cursor" bit in the Write Control Character byte and setting cursor column and cursor row addresses to zero. The row number (the first byte of the order area) is set to 1 using the RELATIVE KEY IS data name, ROW-NUMBER, defined on Line 26 with a value of 1. For full-screen I/O, the row number must be set to 1.

5. The mapping area must be initialized before the screen is displayed using REWRITE. This is accomplished by the paragraph DISPLAY-THE-SCREEN, coded on Lines 52 - 54.

6. A READ must be issued if operator response is required. (If the screen is <u>only</u> to be displayed, the READ is omitted.) This is accomplished by the READ statement on Line 56. The MODIFIABLE option means that only modifiable portions of the screen are transferred to the record description area.

7. The record description area is moved to Working-Storage, thus providing the equivalent of the transfer to the OBJECT fields provided by DISPLAY AND READ. This is accomplished by the MOVE statement on Line 58.

The one DISPLAY AND READ statement in the program illustrated by Figure 4-2 performs the equivalent functions of steps 1 - 7. Using DISPLAY AND READ is therefore the recommended method of programming the workstation. Only in exceptional cases should full screen I/O be used. The most common exceptional case is when only a display of the screen (with no operator action required) is required. DISPLAY AND READ always performs the READ, requesting operator response -- it does not have a "display only" mode. Therefore, to display an entire screen without requiring a read, a REWRITE of the screen, using full screen I/O as described, is required.

Setting the Order Area Using Extensions to REWRITE

The REWRITE statement for the workstation contains options to set the order area. These options allow setting the order area without the tedious and error-prone procedure of defining the order area, specifying figurative constants, and initializing the order area to the figurative constants. Combinations of REWRITE options are allowed, permitting virtually all combinations of order area settings at REWRITE time. The REWRITE options are as follows:

- ALARM. Sound the workstation alarm.

- SETTING CURSOR COLUMN/ROW. Set the cursor to a designated column (1 - 80) and row (1 - 24).

- ROLL DOWN. Copy each row to the next lower row. ROLL DOWN is valid only for row-oriented I/O, to be discussed in Section 4.5.

- ROLL UP. Copy each row to the next higher row. ROLL UP is valid only for row-oriented I/O, to be discussed in Section 4.5.

- ERASE PROTECT. Erase and protect the screen at and after the cursor row address specified in the order area. The order area must be initialized to the desired row address for ERASE PROTECT.

- ERASE MODIFY. Set all modifiable locations after the specified row address to blanks. The order area must be initialized to the desired row address for ERASE MODIFY.

An example of using the REWRITE extension for setting the cursor is the following:

```
REWRITE CRTREC  SETTING CURSOR COLUMN  1 ROW 8.
```

This statement both rewrites the workstation screen and sets the cursor at Row 8 Column 1.

4.5 <u>PROGRAMMING THE WORKSTATION THROUGH ROW-ORIENTED I/O</u>

The workstation can be programmed to rewrite one or more rows at a time. To do this, the 4-byte order area must be defined and initialized correctly; the mapping area, however, is 80 bytes instead of 1920 bytes long.

To specify the row for the workstation rewrite, specify a RELATIVE KEY IS data name in the FILE-CONTROL entry, and move the desired row number to the data name before issuing the REWRITE. For example, if one 80-byte row is to be rewritten on Row 11, move 11 to the RELATIVE KEY IS data name before issuing the REWRITE of the 01-level record area (84 bytes). For rollup or rolldown, the RELATIVE KEY IS data name specifies at what row the function is to begin. Therefore to roll down all screen rows starting with Row 11, move 11 to the RELATIVE KEY IS data name before issuing the REWRITE with the ROLL DOWN option.

Figure 4-11 is a complete COBOL program that produces the same results as the program using DISPLAY AND READ in Figure 4-9 -- positioning the cursor and sounding the workstation alarm. The program additionally demonstrates the following features previously discussed:

- Use of row-oriented I/O.
- REWRITE options that allow setting the cursor position and sounding the workstation alarm without having to set the order area.
- Use of the PFKEY IS clause to determine which PF key was selected after a workstation READ.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ORDRAREA.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT SCREEN ASSIGN TO "SCREEN", "DISPLAY",
        CURSOR POSITION IS CURSOR-POSITION
        RELATIVE KEY    IS ROW-NUMBER
        PFKEY           IS PF-KEY
        ACCESS MODE     IS RANDOM.
DATA DIVISION.
FILE SECTION.
FD  SCREEN
    LABEL RECORDS ARE OMITTED.
01  WORKSTATION-REC.
    03  FILLER                     PICTURE IS X(4).
    03  WS-MAPPING-AREA            PICTURE IS X(80).
WORKING-STORAGE SECTION.
77  ROW-NUMBER   VALUE IS 1        PICTURE IS 99.
01  CURSOR-POSITION.
    03  COLUMN-SETTING             BINARY.
    03  ROW-SETTING                BINARY.
01  CURSOR-LINE.
    05  FILLER         PICTURE IS X(8)    VALUE SPACES.
    05  FILLER         PICTURE IS  X(35)
        VALUE IS "The cursor is at row 24, column 80.".
01  ALARM-LINE.
    05  FILLER         PICTURE IS  X(8)   VALUE SPACES.
    05  FILLER         PICTURE IS  X(34)
        VALUE IS "The workstation alarm has sounded.".
01  PFKEY-LINE.
    05  FILLER         PICTURE IS X(8)    VALUE SPACES.
    05  FILLER         PICTURE IS X(7)    VALUE "PF key".
    05  PF-KEY         PICTURE IS 99      VALUE 0.
    05  FILLER         PICTURE IS X(9)    VALUE " was hit.".
PROCEDURE DIVISION.
CONTROL-THE-CURSOR.
    OPEN I-O SCREEN.
    MOVE 4                  TO ROW-NUMBER.
    MOVE CURSOR-LINE        TO WS-MAPPING-AREA.
    REWRITE WORKSTATION-REC SETTING CURSOR COLUMN 80 ROW 24.
    READ SCREEN.
CHECK-PF-KEY.
    MOVE PFKEY-LINE         TO WS-MAPPING-AREA.
    REWRITE WORKSTATION-REC.
    READ SCREEN.
SOUND-ALARM.
    MOVE ALARM-LINE         TO WS-MAPPING-AREA.
    REWRITE WORKSTATION-REC ALARM.
    READ SCREEN.
    CLOSE SCREEN.
    STOP RUN.
```

Figure 4-11. Setting the Cursor, Checking the PF Key, and Sounding the Alarm

In the program illustrated in Figure 4-11, the FILE-CONTROL entry for the workstation, coded on Lines 6 - 10, contains the CURSOR POSITION clause (coded on Line 7), the RELATIVE KEY phrase (coded on Line 8), and the PFKEY clause (coded on Line 9).

The CURSOR POSITION IS data name, CURSOR-POSITION, coded in Working-Storage on Lines 20 - 22, is a group item containing two elementary binary items, COLUMN-SETTING and ROW-SETTING. COLUMN-SETTING contains the value of the cursor column after a READ; ROW-SETTING contains the value of the cursor row after a READ.

The RELATIVE KEY IS data name, ROW-NUMBER, coded in Working-Storage on Line 19, contains the row number (from 1 to 24) to be rewritten. ROW-NUMBER should therefore be initialized to the value of the row to be rewritten before the REWRITE is issued for the workstation.

The PFKEY IS data name, PF-KEY, coded on Line 34, contains the number of the PF key after a READ. PF-KEY is embedded in a message indicating the PF key number. The message is rewritten after the READ.

In row-oriented I/O, only one row of the screen need be defined. The record description area for the workstation, WORKSTATION-REC, is defined as an 84-byte area on Lines 15 - 17 -- four bytes of FILLER for the order area and 80 bytes for the row to be rewritten.

In the Procedure Division the paragraph CONTROL-THE-CURSOR, coded on Lines 37 - 42, after opening the workstation, sets the cursor at Row 24 Column 80 and displays the message "The cursor is set at Row 24, Column 80." on workstation Row 4. The RELATIVE KEY IS data name, ROW-NUMBER, is set to 4, indicating that Row 4 is to be rewritten, on Line 39. The message is moved to the mapping area on Line 40. Finally, the REWRITE is issued with the SETTING CURSOR option on Line 41. The SETTING CURSOR option sets the cursor to Row 24, Column 80 before the REWRITE is issued.

The paragraph CHECK-PF-KEY, coded on Lines 43 - 46, checks the PF key that was pressed. PF-KEY, the PFKEY IS data name, contains the PF key number. The message containing the value "PF key nn was selected." is moved to the mapping area on Line 44, and the REWRITE is issued. Since the RELATIVE KEY IS data name, ROW-NUMBER, is still set to 4, the message displays on Row 4.

The paragraph SOUND-ALARM, on Lines 47 - 52, sounds the workstation alarm after the message "The workstation alarm has sounded." is moved to the screen mapping area. The ALARM option of the REWRITE statement, coded on Line 49, accomplishes sounding the alarm.

4.6 COEXISTENCE OF DISPLAY AND READ AND FULL SCREEN I/O

It is possible to issue both DISPLAY AND READ, full screen I/O, and row-oriented I/O operations (READ and REWRITE) in the same COBOL program. DISPLAY AND READ and full screen I/O require the same FILE-CONTROL and File Section entries to define the workstation file. DISPLAY AND READs and conventional READs and REWRITEs can be coded in any order in the Procedure Division; in effect, the two methods of workstation programming operate independently of each other. There are some differences in the runtime operation of the two methods, however. These are:

1. It is not necessary to open the workstation file using DISPLAY AND READ. The first DISPLAY AND READ operation will automatically open the workstation file. However, if the workstation file is not opened before the first REWRITE (either explicitly via the OPEN statement or implicitly via DISPLAY AND READ), the REWRITE fails and the program cancels.

2. If workstation interaction is to be performed from more than one module (either using DISPLAY AND READ or using full screen I/O), the workstation must be explicitly closed before the CALL is issued. Each module requires its own FILE-CONTROL entry for the workstation. The FILE-CONTROL entry cannot be passed from one module to another. Before workstation interaction can be performed, the workstation must be opened by that module. If the workstation is open when a module is called, and that module attempts workstation I/O, the workstation will not be properly opened for the called module.

3. DISPLAY AND READ both displays a screen and waits for a workstation response, thereby requiring response using a PF key. DISPLAY AND READ performs both the workstation read and the rewrite. To display a message without requiring operator intervention, REWRITE the workstation record after moving the message to the screen record area.

CHAPTER 5
PRINT FILE PROCESSING

5.1 DEFINING A COBOL PRINT FILE

This chapter discusses print file processing in VS COBOL, and the techniques for skipping lines before printing, skipping lines after printing, and advancing to the next page. Producing well-formatted reports is a requirement for many data processing applications; VS COBOL provides, through extensions of the WRITE statement, powerful facilities to accomplish this task.

VS COBOL treats a print file as a consecutive file with variable-length records and device type of PRINTER. (For a discussion of consecutive file processing in COBOL, refer to Section 2.3.) To specify a COBOL print file, PRTFILE, code the following FILE-CONTROL entry:

```
SELECT PRTFILE ASSIGN TO "PRINT"  "PRINTER".
```

Since the default file organization is a consecutive file, and the other clauses are optional, this FILE-CONTROL entry is sufficient for defining a print file.

The FD entry for a print file is also simple to specify. To specify an FD entry for PRTFILE, code the following statement:

```
FD  PRTFILE
    RECORD CONTAINS 55 CHARACTERS
    LABEL RECORDS ARE OMITTED.
01  PRTLINE    PIC X(55).
```

The record description entry for PRTFILE actually defines a print file containing records of _57_ characters in length. The COBOL program describes _only_ the actual data portion of the print file. The WRITE statement for a print file appends a 2-byte printer control area to the data portion of the record. The statement WRITE PRTLINE in the Procedure Division causes a 57 character print record to be written to the print file PRTFILE: the 2-byte printer control area concatenated with the 55-byte print record defined in the record description entry for PRTFILE.

A print file is typically opened in output mode. The default library used is the pound sign (#), concatenated with the user-ID, concatenated with the characters "PRT"; the default volume is the spool volume (or the system volume if SPOOLVOL is equal to spaces). The default number of records is 1000. All of these defaults can be overridden using appropriate VALUE OF clauses in the FD entry for the print file.

Options of the WRITE statement for print files allow setting of the printer control area. To produce well-formatted reports, it is necessary to control line spacing (how many lines to skip before or after printing the line) and page advancing (whether to skip to the next page). VS COBOL provides two methods of doing this. One method, discussed in Section 5.2, is to use integers or data names with integer values in the BEFORE/AFTER ADVANCING phrase of the WRITE statement. The other method, discussed in Section 5.3, is to use hexadecimal figurative constants in the BEFORE/AFTER ADVANCING phrase to specify print control information and to activate printer hardware-dependent features (such as expanded print).

The length of the print line depends on the printer used. The print line cannot have a length greater than the number of characters-per-line allowed by the printer. Since at compile-time the printer on which the file will print is unknown, the COBOL compiler will not produce any messages if a record length greater than the length supported on the system printer(s) is specified. However, attempting to print a file with a record length greater than that allowed by the printer will produce an error condition that depends on the particular type of printer and whether the file is being printed on-line or spooled.

5.2 USING THE BEFORE/AFTER ADVANCING CLAUSE FOR PRINTER CONTROL

The BEFORE/AFTER ADVANCING phrase of the WRITE statement enables printer control by automatically setting the printer control area. The following functions are available:

Function	Clause
Write a print line, then advance printer.	WRITE print-record BEFORE ADVANCING integer LINES.
Advance printer, then write a print line.	WRITE print-record AFTER ADVANCING integer LINES.
Skip to next page	WRITE print-record AFTER ADVANCING PAGE.

In the BEFORE/AFTER ADVANCING phrase, "integer" can also be a data name having an integer value, or a figurative constant. The use of integers in the BEFORE/AFTER ADVANCING phrase is discussed in this section; the use of figurative constants is discussed in Section 5.3.

A WRITE without the BEFORE/ADVANCING phrase will advance the printer one line and print. Therefore, the following two statements are equivalent:

```
WRITE PRTFILE AFTER ADVANCING 1 LINES.
WRITE PRTFILE.
```

Figure 5-1 is a complete COBOL program that illustrates the BEFORE/AFTER ADVANCING options for printer control.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.   PRNTFILE.
000003 ENVIRONMENT DIVISION.
000004 INPUT-OUTPUT SECTION.
000005 FILE-CONTROL.
000006     SELECT PRTFILE ASSIGN TO "PRINT"  "PRINTER".
000007 DATA DIVISION.
000008 FILE SECTION.
000009 FD  PRTFILE
000010     RECORD CONTAINS 132 CHARACTERS
000011     LABEL RECORDS ARE OMITTED.
000012 01  PRTLINE        PICTURE X(132).
000013 WORKING-STORAGE SECTION.
000014 01  LINE1          PICTURE X(132) VALUE
000015      "THIS LINE PRINTS. THEN THE PRINTER ADVANCES 10 LINES.".
000016 01  LINE2          PICTURE X(132) VALUE
000017      "THE PRINTER WILL ADVANCE 25 LINES. THEN THIS LINE PRINTS.".
000018 01  LINE3          PICTURE X(132) VALUE
000019            "    THIS IS THE FIRST PART".
000020 01  LINE4          PICTURE X(132) VALUE
000021        "THE PRINTER WILL ADVANCE 1 LINE. THEN THIS LINE PRINTS.".
000022 PROCEDURE DIVISION.
000023 OPEN-PRINT-FILE.
000024     OPEN OUTPUT PRTFILE.
000025 WRITE-PRINT-LINES.
000026     WRITE PRTLINE FROM LINE1    BEFORE ADVANCING 10 LINES.
000027     WRITE PRTLINE FROM LINE2    AFTER  ADVANCING 25 LINES.
000028     WRITE PRTLINE FROM LINE3    AFTER ADVANCING PAGE.
000029     MOVE   "                            THIS IS THE SECOND PART "
000030         TO LINE3.
000031     WRITE PRTLINE FROM LINE3    AFTER ADVANCING 0.
000032     WRITE PRTLINE FROM LINE4.
000033 CLOSE-PRINT-FILE.
000034     CLOSE PRTFILE.
000035     STOP RUN.
```

Figure 5-1. Use of BEFORE/AFTER ADVANCING

The FILE-CONTROL entry for PRTFILE, coded on Line 6, specifies a consecutive file with device type of "PRINTER". The device type of PRINTER identifies PRTFILE as a print file.

The record description entry for PRTFILE, coded on Lines 9 - 12, specifies a print record with 132 print positions. The actual record length is 134 bytes because two bytes are added for the printer control area. Space for the printer control area is not defined in the record description entry; rather, it is appended to the print record when the WRITE is executed.

PRTFILE is opened in output mode by the successful execution of the OPEN statement on Line 24. Since the BEFORE ADVANCING option of the WRITE statement means "write the record, then advance the printer", the WRITE statement on Line 26 prints LINE1 and then advances the printer 10 lines. Since the AFTER ADVANCING option of the WRITE statement means "advance the printer, then write the record," the WRITE statement on Line 27 advances the printer 25 lines, and then prints LINE2. Since the PAGE option of the WRITE statement means "skip to the next page", the WRITE statement on Line 28 advances the printer to the next page and prints LINE3, which has the value "THIS IS THE FIRST PART".

Using the number 0 in the AFTER ADVANCING option of the WRITE statement does not cause the printer to advance any lines; the next line will overprint if 0 is used. The MOVE and WRITE statements coded on Lines 29 - 31 print the second part of the heading, with the value "THIS IS THE SECOND PART", to the same line. The message "THIS IS THE FIRST PART THIS IS THE SECOND PART" appears on that line. This message is the result of <u>two</u> WRITE statements -- the WRITE on Line 28 writes the first part of the message, and the WRITE on Line 31 writes the second part of the message.

The WRITE statement on Line 32 writes LINE4 after advancing to the next line and is the equivalent of WRITE LINE4 AFTER ADVANCING 1 LINES.

5.3 USING FIGURATIVE CONSTANTS FOR PRINTER CONTROL

Another method of printer control is to define figurative constants for use in the BEFORE/AFTER ADVANCING phrase. Use of figurative constants enables program control of the printer control area. The figurative constants define a value for the printer control area, which will be moved to the first two bytes of the printer record when a WRITE statement is issued. In addition to printer spacing and page ejecting, hardware-specific features, such as expanded print and sounding the printer alarm, can be activated.

Not all VS printers support expanded print or the hardware alarm; if a particular printer supports these attributes, using the figurative constant method will activate these features. Refer to Appendix F for the appropriate bits to set when defining the figurative constant.

In the Environment Division, construct a 2-byte figurative constant to enable the functions desired. The printer must support the feature (such as expanded print, actuating the hardware alarm, or channel skipping) desired. For example, to sound the hardware alarm on a VS printer that supports this feature, code the following:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
FIGURATIVE-CONSTANTS.
    SOUND-THE-ALARM   IS "1000".
```

As shown in Appendix F, Bit 3 of the first byte must be on to sound the hardware alarm. The figurative constant SOUND-THE-ALARM has Bit 3 of the first byte set. In the Procedure Division, to sound the alarm after writing the line PRTLINE on the file PRTFILE, code the following:

```
WRITE PRTLINE AFTER ADVANCING SOUND-THE-ALARM.
```

The figurative constant used in the WRITE statement determines all the printer control, including control of printer advancing and advancing to the next page. Therefore, the "BEFORE ADVANCING" and "AFTER ADVANCING" phrases are ignored by the WRITE when figurative constants are used. The setting of the "space before printing" or the "space after printing" bit in the figurative constant determines whether printing precedes spacing or spacing precedes printing, regardless of whether the "BEFORE ADVANCING" or "AFTER ADVANCING" phrase is used. In addition, the words "PAGE" and "LINES" as WRITE statement options are ignored; page skipping and line advancing are governed solely by the settings of the figurative constant.

The program in Figure 5-2 is identical to the program in Figure 5-1, with the exception of the figurative constants defined on Lines 4 - 8 and their use in the WRITE statement options. The figurative constants BEFORE-TEN, AFTER-25, TOP-OF-FORM, and ZERO-LINES, are constructed so that when used in the WRITE statement, the record is written and the printer advances 10 lines, the printer advances 25 lines and the record is written, the printer skips to the top of the next page, and the printer does not advance before or after the record is written. The figurative constant values define 2 bytes to be moved to the printer control area by the WRITE. Refer to Appendix F for further discussion of definition of these values.

The paragraph WRITE-PRINT-LINES in the program of Figure 5-2 generates the identical output as the paragraph WRITE-PRINT-LINES in the program of Figure 5-1 except that in Figure 5-2, the settings of the figurative constants determine printer action. For example, the WRITE statement on Line 31 prints LINE1 and advances the printer 10 lines because the figurative constant BEFORE-TEN used in the WRITE statement is set to the values that cause this action.

```
000001 IDENTIFICATION DIVISION.
000002 PROGRAM-ID.   PRNTFILE.
000003 ENVIRONMENT DIVISION.
000004 CONFIGURATION SECTION.
000005 FIGURATIVE-CONSTANTS.
000006     BEFORE-TEN   IS   "400A"
000007     AFTER-25     IS   "0019"
000008     TOP-OF-FORM  IS   "8001"
000009     ZERO-LINES   IS   "0000".
000010 INPUT-OUTPUT SECTION.
000011 FILE-CONTROL.
000012     SELECT PRTFILE ASSIGN TO "PRINT"  "PRINTER".
000013 DATA DIVISION.
000014 FILE SECTION.
000015 FD  PRTFILE
000016     RECORD CONTAINS 132 CHARACTERS
000017     LABEL RECORDS ARE STANDARD.
000018 01  PRTLINE    PIC X(132).
000019 WORKING-STORAGE SECTION.
000020 01  LINE1         PICTURE X(132) VALUE
000021      "THIS LINE PRINTS. THEN THE PRINTER ADVANCES 10 LINES.".
000022 01  LINE2         PICTURE X(132) VALUE
000023      "THE PRINTER WILL ADVANCE 25 LINES. THEN THIS LINE PRINTS."
000024 01  LINE3         PICTURE X(132) VALUE
000025           "    THIS IS THE FIRST PART".
000026 01  LINE4         PICTURE X(132) VALUE
000027        "THE PRINTER WILL ADVANCE 1 LINE. THEN THIS LINE PRINTS."
000028 PROCEDURE DIVISION.
000029 OPEN-PRINT-FILE.
000030     OPEN OUTPUT PRTFILE.
000031 WRITE-PRINT-LINES.
000032     WRITE PRTLINE FROM LINE1    BEFORE ADVANCING BEFORE-TEN.
000033     WRITE PRTLINE FROM LINE2    AFTER  ADVANCING AFTER-25.
000034     WRITE PRTLINE FROM LINE3    AFTER ADVANCING TOP-OF-FORM.
000035     MOVE   "                            THIS IS THE SECOND PART
000036         TO LINE3.
000037     WRITE PRTLINE FROM LINE3    AFTER ADVANCING ZERO-LINES.
000038     WRITE PRTLINE FROM LINE4.
000039 CLOSE-PRINT-FILE.
000040     CLOSE PRTFILE.
000041     STOP RUN.
```

Figure 5-2. Use of Figurative Constants to Control the Printer

CHAPTER 6
TAPE FILE PROCESSING

6.1 INTRODUCTION

This chapter discusses the process of creating and maintaining tape files using VS COBOL. Tape files are identified in the FILE-CONTROL entry for the file by the device type "TAPE". Thus, to define a tape file TAPEFILE, code the FILE-CONTROL entry as follows:

```
SELECT TAPEFILE
  ASSIGN TO "TAPEIN", "TAPE".
```

Consecutive files are the only file organization supported on tape. Any function supported for consecutive files is allowed for tape. For a detailed discussion of consecutive file processing in COBOL, refer to Section 2.3.

6.2 TAPE LABEL PROCESSING

Tape labels identify a tape file. Multiple files are allowed on a VS tape volume; a tape label will identify which file is to be processed. (The VS also supports nonlabelled tape files, in which case the file to be processed is identified by the order in which the file is located on the tape.)

To facilitate the transport of information from computers using industry-standard tape file formats, the VS supports three methods of tape label processing. These are:

1. ANSI Tape Labels. An ANSI labelled tape (AL) identifies a tape label format in accordance with the standards of the American National Standards Institute (ANSI). The ANSI standard tape label format is intended to be a common industry standard, facilitating file transfer across computers adhering to the industry standard conventions.

2. IBM Tape Labels. An IBM labelled tape (IL) identifies a tape label format in accordance with IBM 370 standards. The IBM tape label standard differs from the ANSI tape label standard. Therefore, to facilitate file transfer from IBM 370 systems to the VS, IBM tape label formats are supported.

3. No Tape Labels. A nonlabelled tape (NL) identifies a tape volume with no tape labels. The file to be processed is identified by its order on the tape. Nonlabelled tapes are the common mode of tape processing for some computer vendors. In instances where file transfer between a computer supporting neither AL nor IL tapes and the VS is required, the best alternative may be creating an NL tape on the system and transporting it to the VS.

The VS utility TAPEINIT enables initialization of a tape volume and specification of the type of tape label processing allowed. Through the TAPEINIT utility, the tape volume can be initialized as either an ANSI-labelled tape (AL), an IBM-labelled tape (IL), or a nonlabelled tape (NL). In VS COBOL, LABEL RECORDS ARE STANDARD implies either an AL or an IL tape, while LABEL RECORDS ARE OMITTED implies an NL tape. Refer to VS Utilities Reference Manual for further information on the TAPEINIT utility.

6.3 USE OF LABEL RECORDS CLAUSE FOR TAPE LABEL PROCESSING

6.3.1 ANSI and IBM Tape Label Processing

In the FD for the tape file, the LABEL RECORDS clause identifies whether or not the tape has labels. If tape labels are present, code in the FD for TAPEFILE as:

```
FD  TAPEFILE
    LABEL RECORDS ARE STANDARD.
```

In tape label processing, the program or procedure must supply the file name. An additional specification of a library name will provide qualification of the file name on the tape label. To specify the file name, use the VALUE OF FILENAME clause; to specify the library name, use the VALUE OF LIBRARY clause in the FD of the file. Refer to Section 2.2.4 for a detailed discussion on methods of supplying the file and library names for the tape file.

If an existing tape file is opened, the tape volume is scanned for a tape file name containing a match between the program-supplied tape file name and the file name on the label. If there is no match, an error message is displayed, giving the opportunity for respecifying the file, library, and volume. If a file is being created (opened in output mode), either an ANSI label or an IBM label is created.

The VS file naming convention permits up to three levels of name qualification -- file name, library name, and volume name. Other file naming conventions (for example, the convention for naming tape data sets on the IBM 370) may support more than three levels of name qualification. When the file label is read on the VS, the first three levels of qualification are converted to file, library, and volume names; any further qualification of names is ignored. For example, an IBM 370 tape data set with the name 111.222.333.444.555 is converted to a VS name of VOLUME=111, LIBRARY=222 and FILE=333. The fourth and fifth levels of qualification (444 and 555) are ignored.

If a tape contains files using a collating sequence different from the VS collating sequence, code translation is the responsibility of the programmer. This is primarily a consideration in conversion of IL tape files. IBM 370 tape data sets frequently contain files using the EBCDIC collating sequence. Such a file cannot be processed using the VS collating sequence, which is the ASCII collating sequence. To translate the file into the VS collating sequence, run the VS code translation utility (TRANSL) before processing the tape file using the COBOL program. Refer to VS Utilities Reference manual for detailed information on the TRANSL utility.

6.3.2 Nonlabelled Tape Processing

If tape labels are not present, or if the tape label is to be processed by the program as a file, code the following FD for TAPEFILE:

```
FD  TAPEFILE
    LABEL RECORDS ARE OMITTED.
```

Through the TAPEINIT utility, the tape volume has been initialized as a nonlabelled tape (NL). In VS COBOL, LABEL RECORDS ARE OMITTED implies an NL tape.

Since no identification for the file exists, it can be found only by specifying a file number representing the position of the file on the tape volume. Records are written to the tape file; when the file is closed, an end-of-file marker is written to the tape volume signifying the end of the tape file. By counting end-of-file markers, a particular file can be located.

To specify the file number, VS COBOL provides a clause -- the VALUE OF POSITION clause. The data name in the VALUE OF POSITION clause is an integer representing the ordinal number of the file to be processed. For example, to specify that TAPEFILE is the fifth file on the tape volume, code the following:

```
FD  TAPEFILE
    LABEL RECORDS ARE OMITTED
    VALUE OF POSITION IS POSITION-COUNTER.
```

In Working-Storage, code the following:

```
WORKING-STORAGE SECTION.
77  POSITION-COUNTER        PIC S9    COMPUTATIONAL    VALUE +5.
```

If a tape contains labels that do not adhere to ANSI or IBM label processing, the tape label itself can be processed as a file and the information contained in the label can be used by the program to process the file. The tape can be processed as an NL tape and the program can then access the label. This method should be used in conversion of tapes, from systems that adhere neither to ANSI nor to IBM tape labelling conventions, to the VS.

CHAPTER 7
SORT-MERGE PROCESSING

7.1 INTRODUCTION

Applications often find it necessary to have a COBOL program sort or merge data files while executing. Sorts and merges are performed in COBOL by the SORT-MERGE module. Programmers can include a sort or merge operation in a program by coding the procedures and syntax described in this chapter. Formats, syntax and general rules are presented in Part II.

7.2 SORTING

COBOL programs perform sort operations by collecting the records from the file or files to be sorted into a temporary file called a sort file. Once created, the sort file is processed according to the instructions defined in a SORT statement and released back to the program.

Sort processing can be accomplished by the SORT statement with the USING and GIVING phrases, or by defining an input and output procedure. The input procedure makes records available for sorting from the file or files to be sorted by means of the RELEASE statement. The SORT statement then arranges the entire set of records in the sort file. The reordering is performed according to the keys specified by the programmer in the SORT statement. When the reordering is complete, the output procedure makes the sorted records available to the program by means of the RETURN statement.

Sorting files within a VS COBOL program has an advantage over external sorts in that the COBOL sort allows the programmer to manipulate the individual records during the sort process. The manipulation may consist of addition, deletion, creation, or editing of the individual records. It may be necessary to apply the manipulation before or after the records are reordered, or even in both places. This special processing is applied to the records during execution of the input and output procedures. Specific rules governing what the programmer can and cannot do during these procedures are included in the RELEASE and RETURN statement sections in Chapter 12.

A COBOL program may contain more than one sort, each of which can have its own input and output procedures. The sort feature automatically causes execution of these procedures at the point specified by the programmer in such a way that extra passes over the sort file are not required. Sort files are named by a file control entry and are described by a sort or merge file description (SD) entry (refer to Chapter 11). Sort files can never be accessed directly. They can only be accessed from the input and output procedures. The SORT statement must name the file or procedure from which the input procedure acquires the records to be sorted and must name the file into which the sorted records are to be placed.

7.3 MERGING

Merge processing in a COBOL program is similar to sort processing. The major difference is that the MERGE function does not employ an input procedure. Files to be merged are accessed by the MERGE statement. Records from these files are placed into a temporary file (the merge file). The merge is accomplished according to the key specified by the programmer in th MERGE statement. With the reordering complete, the merge file is made available to the program. This can be accomplished by either a coding a GIVING phrase with the MERGE statement of by the RETURN statement in an output procedure.

As is true for sort files, merge files are accessed and referred to only by the MERGE statement.

7.4 IMPLEMENTATION

Implementation of a SORT or MERGE operation in a VS COBOL program requires entries in the Environment, Data, and Procedure Divisions of that program. The file to be sorted or merged is named and its file-related characteristics defined in the FILE-CONTROL Section of the Environment Division.

The SORT FILE DESCRIPTION (SD), is the Data Division entry. An SD file description contains information about the size and the names of the data records of the file(s) to be sorted or merged.

The Procedure Division statements are SORT, MERGE, RELEASE, and RETURN. The SORT statement defines the sort function in the following three steps:

1. Creates a sort file either by executing an input procedure or by transferring records from some other file.

2. Sorts the records in the sort file on a set of user-specified keys.

3. Makes available each record from the sort file in the sorted order to either an output procedure or an output file.

The MERGE statement combines two or more identically sequenced files on a set of user-specified keys. During this process the statement makes the merged records available, in merge order, to either an output procedure or an output file.

The RELEASE statement transfers records to the initial phase of a SORT operation. The RETURN statement obtains either sorted records from the final phase of a SORT operation or merged records during a MERGE operation.

7.5 COLLATING SEQUENCE AND SORT-MERGE LIMITS

The COBOL SORT-MERGE module links to the VS SORT utility. As a result, the collating sequence for a COBOL sort is the same as that of the SORT utility (refer to the VS System Utilities Reference).

A SORT-MERGE file may have a maximum of 8 keys. The maximum key length depends on the type of data. Binary data (USAGE BINARY) can have 2 positions. Character data (USAGE DISPLAY) can have from 1 to 256 character positions. Decimal data (USAGE DISPLAY) can have from 1 to 16, as can Packed (USAGE COMP), Zoned Decimal (USAGE DISPLAY), and Zoned Decimal sign leading (USAGE DISPLAY). Numeric data with USAGE DISPLAY or USAGE COMP can have key lengths from 1 to 16.

7.6 PROGRAM EXAMPLE

Figure 7-1 is a program example that implements a number of SORT operations.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  ST100.
000300 ENVIRONMENT DIVISION.
000400 INPUT-OUTPUT SECTION.
000500 FILE-CONTROL.
000600        SELECT FILE1
000700        ASSIGN TO "FILE1"  "DISK"
000800        ORGANIZATION IS SEQUENTIAL
000900        ACCESS MODE IS SEQUENTIAL.
001000
001100        SELECT FILE2
001200        ASSIGN TO "FILE2" "DISK"
001300        ORGANIZATION IS SEQUENTIAL
001400        ACCESS MODE IS SEQUENTIAL.
001500
001600        SELECT FILE3
001700        ASSIGN TO "FILE3"  "DISK"
001800        ORGANIZATION IS SEQUENTIAL
001900        ACCESS MODE IS SEQUENTIAL.
```

Figure 7-1. SORT Processing

```
002000
002100        SELECT FILE4
002200        ASSIGN TO "FILE4"  "DISK"
002300        ORGANIZATION IS INDEXED
002400        ACCESS MODE IS DYNAMIC
002500        RECORD KEY IS FILE4-KEY.
002600
002700        SELECT SORT1
002800        ASSIGN TO "FILE1" "DISK".
002900
003000        SELECT SORT2
003100        ASSIGN TO "FILE2" "DISK".
003200
003300        SELECT SORT3
003400        ASSIGN TO "FILE3" "DISK".
003500
003600 DATA DIVISION.
003700 FILE SECTION.
003800 FD   FILE1
003900        LABEL RECORDS ARE STANDARD
004000        DATA RECORD IS FILE1-RECORD.
004100 01     FILE1-RECORD                PIC X(50).
004200
004300 FD   FILE2
004400        LABEL RECORDS ARE STANDARD
004500        DATA RECORD IS FILE2-RECORD.
004600 01     FILE2-RECORD.
004700        05  FILE2-FIELD1            PIC X(5).
004800        05  FILE2-FIELD2            PIC X(5).
004900        05  FILLER                  PIC X(40).
005000
005100 FD   FILE3
005200        LABEL RECORDS ARE STANDARD
005300        DATA RECORD IS FILE3-RECORD.
005400 01     FILE3-RECORD.
005500        05  FILE3-FIELD1            PIC X(5).
005600        05  FILE3-FIELD2            PIC X(5).
005700        05  FILLER                  PIC X(40).
005800
005900 FD   FILE4
006000        LABEL RECORDS ARE STANDARD
006100        DATA RECORD IS FILE4-RECORD.
006200 01     FILE4-RECORD.
006300        05  FILE4-KEY.
006400            10  FILE4-FIELD1        PIC X(5).
006500            10  FILE4-FIELD2        PIC X(5).
006600            10  FILE4-FIELD3        PIC X(5).
006700        05  FILLER                  PIC X(35).
006800
```

Figure 7-1. SORT Processing (continued)

```
006900 SD   SORT1
007000      RECORD CONTAINS 50 CHARACTERS
007100      DATA RECORD IS SORT1-RECORD.
007200 01   SORT1-RECORD.
007300      05 S1-SORTKEY1            PIC X(5).
007400      05 S1-SORTKEY2            PIC X(5).
007500      05 FILLER                 PIC X(40).
007600
007700 SD   SORT2
007800      RECORD CONTAINS 50 CHARACTERS
007900      DATA RECORD IS SORT2-RECORD.
008000 01   SORT2-RECORD.
008100      05  S2-SORTKEY1               PIC X(5).
008200      05  S2-SORTKEY2               PIC X(5).
008300      05  FILLER                    PIC X(40).
008400
008500 SD   SORT3
008600      RECORD CONTAINS 50 CHARACTERS
008700      DATA RECORD IS SORT3-RECORD.
008800 01   SORT3-RECORD.
008900      05  S3-SORTKEY1               PIC X(5).
009000      05  S3-SORTKEY2               PIC X(5).
009100      05  S3-SORTKEY3               PIC X(5).
009200      05  S3-SORTKEY4               PIC X(5).
009300      05  S3-SORTKEY5               PIC X(5).
009400      05  FILLER                    PIC X(25).
009500
009600 WORKING-STORAGE SECTION.
009700
009800 01   WS-FILE1-RECORD.
009900      05  WS-FILE1-FIELD1           PIC X(5).
010000      05  WS-FILE1-FIELD2           PIC X(5).
010100      05  FILLER                    PIC X(40).
010200
010300 PROCEDURE DIVISION.
010400
010410 OPEN-RTN.
010420     OPEN OUTPUT FILE3 FILE4.
010430
010440 OPEN-RTN-EXIT.
010450     EXIT.
010455
010500 FIRST-SORT.
010600     SORT SORT1
010700         ON ASCENDING KEY S1-SORTKEY1
010800         USING FILE1
010900         GIVING FILE2.
011000 FIRST-SORT-EXIT.
011100     EXIT.
011200
```

Figure 7-1. SORT Processing (continued)

```
011300 SECOND-SORT.
011400     SORT SORT2
011500         ON ASCENDING KEY S2-SORTKEY1
011600         ON ASCENDING KEY S2-SORTKEY2
011700         WITH DUPLICATES IN ORDER
011800         INPUT PROCEDURE IS BUILD-INDEX-FILE1 THRU BUILD1-EXIT
011900         GIVING FILE3.
012000 SECOND-SORT-EXIT.
012100     EXIT.
012200
012300
012400 THIRD-SORT.
012450     OPEN INPUT FILE2.
012500     SORT SORT3
012600         ON ASCENDING KEY S3-SORTKEY4
012700         ON ASCENDING KEY S3-SORTKEY5
012800         INPUT PROCEDURE IS BUILD-INDEX-FILE2 THRU BUILD-2-EXIT
012900         OUTPUT PROCEDURE IS BUILD-INDEX-FILE3 THRU BUILD3-EXIT.
013000 THIRD-SORT-EXIT.
013100     EXIT.
013200
013300 EXIT-RTN.
013400     CLOSE FILE2  FILE3 FILE4.
013500     STOP RUN.
013600
013700 BUILD-INDEX-FILE1.
013800     READ FILE2 NEXT AT END
013900         GO TO BUILD1-EXIT.
014000     MOVE FILE2-FIELD1 TO S2-SORTKEY1.
014100     MOVE FILE2-FIELD2 TO S2-SORTKEY2.
014200     RELEASE SORT2-RECORD.
014300     GO TO BUILD-INDEX-FILE1.
014400 BUILD1-EXIT.
014500     EXIT.
014600
014700 BUILD-INDEX-FILE2.
014800     READ FILE3 NEXT AT END
014900         GO TO BUILD2-EXIT.
015000
015100     MOVE FILE3-FIELD1 TO S3-SORTKEY1.
015200     MOVE FILE3-FIELD2 TO S3-SORTKEY2.
015300     RELEASE SORT3-RECORD.
015400     GO TO BUILD-INDEX-FILE2.
015500 BUILD2-EXIT.
015600     EXIT.
```

Figure 7-1. SORT Processing (continued)

```
015700 BUILD-INDEX-FILE3.
015800     RETURN SORT3 AT END
015900         GO TO BUILD3-EXIT.
016000
016100     MOVE S3-SORTKEY1 TO FILE4-FIELD1.
016200     MOVE S3-SORTKEY2 TO FILE4-FIELD2.
016300
016400     WRITE FILE4-RECORD.
016500     GO TO BUILD-INDEX-FILE3.
016600 BUILD3-EXIT.
016700     EXIT.
016800
```

Figure 7-1. SORT Processing (continued)

The sample program in Figure 7-1 performs three different SORT operations. All input and output files are assumed to exist prior to running the object program.

Lines 010500 through 011100 display a SORT operation employing the USING and GIVING statements. These statements open both the input file (FILE1) and output file (FILE2). No other OPEN statements are necessary.

Lines 011300 through 012100 display a SORT operation employing an input procedure and more than one sort key. Lines 012400 through 013100 display a SORT operation employing both an input and output procedure.

PART II

REFERENCE

CHAPTER 8
GENERAL LINGUISTIC CONSIDERATIONS

8.1 INTRODUCTION

PART II explains the form and function of the various linguistic units of COBOL: characters, words, clauses, statements, sentences, entries, paragraphs, sections, and divisions. This chapter deals with the elements of COBOL, the combination of elements into more complex forms, general formatting considerations, and the notation that is used in subsequent chapters. The remainder of PART II will discuss the components of each of the divisions of a COBOL program.

8.2 COBOL CHARACTERS

The most basic unit of the language is the character. The set of characters used to form COBOL character-strings and separators includes the letters of the English alphabet, Arabic digits, and special characters. The complete VS COBOL character set consists of the following characters:

VS COBOL Character Set

Character	Meaning
0,1,...,9	digit
A,B,...,Z	letter
	space (blank)
+	plus sign
-	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma
;	semicolon
.	period (decimal point)
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol

The following characters are used for punctuation: space [ ], comma [,], semicolon [;], period [.], quotation mark ["], left parenthesis [(], and right parenthesis [)].

8.3 CHARACTER-STRINGS AND SEPARATORS

A COBOL character-string is a character or a sequence of continuous characters that forms a COBOL word, literal, PICTURE character-string, or comment-entry. Separators are strings of one or more punctuation characters that delimit a character-string. A character-string can only be concatenated with a separator. A separator can be concatenated with another separator or with a character-string. Concatenated separators and character-strings form the text of a COBOL source program.

The rules for the formation of separators are:

1. The punctuation character space (blank) is a separator. Anywhere a space is used as a separator, more than one space can be used.

2. The punctuation characters comma, semicolon, and period are separators only when immediately followed by a space. However, these particular separators are permissible only where designated by the general formats (refer to Section 8.7.1, Definition of a General Format), by format punctuation rules (refer to Section 8.7.4, Format Notation), by statement and sentence structure definitions, or by reference format rules for source programs (refer to Section 8.7.5, COBOL Source-Program Reference Format).

3. The punctuation characters left and right parentheses are separators. Parentheses must appear in balanced pairs of left and right parentheses delimiting subscripts, indices, arithmetic expressions, or conditions.

4. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or left parenthesis. A closing quotation mark must immediately precede a space, comma, semicolon, period, or right parenthesis. Quotation marks can appear only in balanced pairs of opening and closing quotes delimiting nonnumeric literals. The only exception to this rule is line continuation (refer to Section 8.7.6, Continuation Lines). To represent a single quotation mark within a nonnumeric literal, two contiguous quotation marks must be used.

5. The separator space (blank) can precede all other separators except the closing quotation mark. A space preceding a closing quotation mark is considered a part of the nonnumeric literal and not a separator.

6. The separator space may immediately follow any separator except the opening quotation mark. A space following an opening quotation mark is considered a part of the nonnumeric literal and not a separator.

7. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period. (PICTURE character-strings consist of combinations of COBOL characters used in the PICTURE clause as symbols defining data categories and editing features.)

The above rules do not apply to punctuation characters that appear as parts of numeric or nonnumeric literals, PICTURE character-strings, comment entries, or comment lines. Such punctuation characters are not considered separators.

8.4 PUNCTUATION

The following rules control punctuation in all four divisions of a COBOL program to increase readability, provide special forms of data, and delimit sentences.

1. The punctuation characters comma and semicolon are optional. They can be used interchangeably. Neither one can immediately precede the first clause of an entry or paragraph.

2. It is permissible to use a semicolon or comma between statements in the Procedure Division.

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

4. The parentheses provide for indexing and subscripting.

5. The quotation marks serve to delimit nonnumeric literals.

6. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except within nonnumeric literals.

7. A space must always precede and follow an arithmetic operator or an equal sign.

8.5 DIVISIONAL COMPONENTS

A COBOL program is made up of four divisions. Each division must begin with one of the division headers listed in Section 1.3, Structure of COBOL Programs. The composition of a division is as follows:

SECTIONS are composed of PARAGRAPHS, which are composed of

SENTENCES or ENTRIES, which are composed of

CLAUSES or STATEMENTS, which are composed of

PHRASES, which are composed of

USER-DEFINED and COBOL-DEFINED WORDS, which are composed of

COBOL CHARACTERS and PUNCTUATION.

8.5.1 Sections

The Environment, Data, and Procedure Divisions are organized into sections. A section consists of a section header, which terminates with a period, followed by zero, one, or more successive paragraphs. A section ends immediately before the next section or division, or at the end of the program, or, in the Declaratives Section of the Procedure Divison, at the key words END DECLARATIVES. Therefore, each section consists of the section header and the related section body.

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 header is composed of a section name, followed by the reserved word SECTION, a period, and a space.

8.5.2 Paragraphs

A paragraph consist of a paragraph header followed by zero, one, or more entries, or of a paragraph name followed by a period and a space and by zero, one, or more sentences. Comment entries. may be included within a paragraph. Each paragraph ends immediately before the next paragraph, section, or division, or at the end of the program, or, in the Declaratives Section of the Procedure Divison, at the key words END DECLARATIVES.

8.5.3 Sentences

A COBOL sentence consists of one or more statements and is terminated by a period followed by a space. Punctuation within sentences is permitted in certain places to improve readability. Sentences occur only in the Procedure Division.

8.5.4 Entries

An entry is any descriptive clause or set of consecutive descriptive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division.

8.5.5 Clauses

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

8.5.6 Statements

A statement is a syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

8.5.7 Phrases

A phrase is an ordered set of one or more consecutive COBOL character-strings that forms a portion of a COBOL statement or clause.

8.6 COBOL WORDS

A COBOL word consists of a combination of one or more characaters selected from the COBOL character set for words (0 through 9, A through Z, and the hyphen). Each word contains no more than 30 characters and neither begins nor ends with a hyphen.

The character space (blank) cannot appear in any word, although it can be used as a separator between words. Two or more spaces can occur anywhere a space serves as a separator.

A word is terminated by one of the following punctuation characters:

Character	Meaning
	space (blank)
.	period
,	comma
;	semicolon

In each case (except for space) the punctuation character must be followed by a space.

There are two kinds of COBOL words: user-defined and COBOL-defined.

8.6.1 User-defined Words

A user-defined word is a COBOL word that must be supplied by the user to satisfy the format of a clause or statement. Thus, it has a meaning specific to the program in which it is used.

There are 14 types of user-defined words. Except for level numbers, each user-defined word can belong to one and only one of these types within a given source program. All user-defined words, with the exception of paragraph names, section names, and level numbers, must contain at least one alphabetic character. Furthermore, all user-defined words of each type must be unique.

The 14 types of user-defined words are:

Alphabet name	Names a special character set and/or collating sequence in the OBJECT-COMPUTER and SPECIAL-NAMES paragraphs or the CODE-SET clause. Wang VS uses the ASCII code.

Condition name	Names 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.
	
	Condition-names are defined in the Data Division as 88 level items. For example,
	
	`01 END-OF-FILE-IND PIC X VALUE "0".`
	`   88 EOF                VALUE "1".`
	`   88 NOT-EOF            VALUE "0".`
	
	EOF and NOT-EOF are condition-names.

Data name	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 that can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.
File name	Names a file described in a file description entry within the File Section of the Data Division.
Index name	Names an index associated with a specific defined table.
Level number	Denotes the position of a data item in the hierarchy of a data description or indicates special properties of a data description entry. Level numbers need not be unique; a given specification of a level number may be identical to any other level number and may even be identical to a paragraph name or section name.

A level number is a 1- or 2-digit number chosen from the numbers 1 through 49, 77, and 88. The range of numbers 1 through 49 indicates the position of a data item in the hierarchical structure of a logical record. Level numbers 77 (refer to "Noncontiguous Working-Storage" in Section 11.3.2 and "Noncontiguous Linkage Storage" in Section 11.3.5) and 88 (refer to "Format 2" in Section 11.3.3) identify special properties of the associated data description .

Level numbers in the range 1 through 9 can occur as single digits or be preceded by zero. In this manual, the form 01, 02,....,09 is used to represent level numbers 1 through 9.

<u>Library</u> <u>name</u>	Names a COBOL library containing text to be included in a given source program by the compiler. It is used with the COPY statement.
<u>Mnemonic</u> <u>name</u>	Is associated with a specified implementor-name (a COBOL-defined word) in the SPECIAL-NAMES paragraph of the Environment Division.
<u>Paragraph</u> <u>name</u>	Begins a paragraph of the Procedure Division.
<u>Program</u> <u>name</u>	Identifies a source program and all listings pertaining to a particular program. It is assigned to a source program in the PROGRAM-ID paragraph of the Identification Divison.
<u>Record</u> <u>name</u>	Names a record described in a record description entry in the Data Division.
<u>Routine</u> <u>name</u>	Identifies a procedure written in a language other than COBOL. A routine name is used with the ENTER statement.
<u>Section</u> <u>name</u>	Begins a section of the Procedure Division.
<u>User-figurative constant</u>	Names a hexadecimal character. The user-figurative constant is assigned in the FIGURATIVE-CONSTANTS paragraph of the Environment Division. The hexadecimal character can then be referenced in the Procedure Division by means of the figurative constant name. For example,

```
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. WANG-VS.
OBJECT-COMPUTER. WANG-VS.
CONFIGURATION SECTION.
FIGURATIVE-CONSTANTS. ONE IS "01",
     NOTAB IS "80", DIM IS "8C",
     TAB IS "AO".
 PROCEDURE DIVISION.
     MOVE DIM TO FAC OF DATA-NAME-1.
     MOVE ONE TO CONSTANT-ONE.
```

8.6.2 COBOL-defined Words

COBOL-defined words and names have the same meanings in all COBOL programs and include words such as ADD, IS, or READ. COBOL-defined words can be classified as either reserved words or system names. Within a given source program, a COBOL word can belong to one, and only one, of these classes.

A system name is a COBOL word that communicates with the physical operating environment (hardware). There are two types of system names: computer names and implementor names. A computer name is used in the SOURCE-COMPUTER and OBJECT-COMPUTER paragraphs of the Environment Division to identify the computer on which a program is to be compiled or run. The computer name recognized by the VS compiler is WANG-VS. Implementor names refer to particular features of an implementor's computing system.

A reserved word is one of a specified list of words (refer to Appendix A, Reserved Words) that must not appear in programs as user-defined words or system names but can only be used as specified in the general formats. However, reserved words can appear as nonnumeric literals, that is, a reserved word can be enclosed in quotation marks. Used in this manner, they do not take on the preassigned meaning of reserved words.

There are six types of reserved words: key words, optional words, connectives, special registers, figurative constants, and special-character words. The six types are:

Key words	Are required when the formats using them appear in a source program. In the general formats in this manual, key words appear in uppercase characters that are underlined. For example, MOVE literal TO identifier In this case, MOVE and TO are key words.

Optional words	Improve the readability of the source code in which they appear, but do not affect the meaning of that code. For example, IS clarifies the statement: FILENAME IS "TAXFILE". Optional words appear in this manual in uppercase characters, but are distinguished from key words by the absence of an underline.
Connectives	Occur in three types: 1. Logical connectives, used in the formation of conditions: AND, OR (such as IF COUNT = 0 AND MAX GREATER THAN 10) 2. Series connectives, used to link two or more consecutive operands: ',' (separator comma) or ';' (separator semicolon) (such as, ADD 1 TO COUNT; GO TO KS1). 3. Qualifier connectives that are used to associate a file name with its qualifier library name: OF, IN.
Special registers	Name certain compiler generated storage areas whose primary use is to store information in conjunction with the use of specific COBOL features. These special registers include: DEBUG-ITEM (refer to Section 13.2.1, DEBUG-ITEM) and RETURN-CODE. RETURN-CODE is a special register with an implied PICTURE of 999. Any COBOL program can move a value to this special register, which can then be tested through a procedure. An example of special register usage is: MOVE 99 TO RETURN-CODE.
Figurative constants (system-defined)	Reserved words that are used to name and reference specific constant values. These words must not be bounded by quotation marks when used as figurative constants. The singular and plural forms are equivalent and may be used interchangeably. These figurative constant values are generated by the compiler and referenced through the use of the following reserved words:

Cobol Word	Meaning
ZERO ZEROS ZEROES	Represents the numeric value zero or one or more of the alphabetic character zero ('0') depending on the context.
SPACE SPACES	Represents one or more of the character space.
HIGH-VALUE HIGH-VALUES	Represents one or more of the hexadecimal character "FF", the character that has the highest ordinal position in the program collating sequence (ASCII).
LOW-VALUE LOW-VALUES	Represents one or more of the hexadecimal character "00", the character that has the lowest ordinal position in the program collating sequence (ASCII).
QUOTE QUOTES	Represents one or more quotation marks ("). The word QUOTE or QUOTES can be used wherever a system-defined figurative constant can be used, e.g., in a VALUE IS clause of a data description entry. But neither word can be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE PTL QUOTE is incorrect as a way of stating the nonnumeric literal "PTL".

When a system-defined 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 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 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 can be used wherever a literal appears in a format. Whenever the literal is restricted to having only numeric characters, however, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

When the figurative constants HIGH-VALUE(S) or LOW-VALUE(S) are used in the source program, the actual character associated with each figurative constant depends upon the ASCII collating sequence.

Special-character words	Reserved words that are arithmetic operators or relational characters. These characters include '+','-', '>', '<', '=', '*', and '/'.

8.6.3 Literals

A literal is a character-string with a constant value, a value not determined by COBOL or user definition, but by the ordered set of characters of which the literal is composed. Every literal belongs to one of two types: nonnumeric or numeric.

Nonnumeric Literals

A nonnumeric literal is a character-string delimited on both ends by quotation marks and consisting of any allowable character. The length of the nonnumeric literal can be from 1 to 120 characters. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used. The value of a nonnumeric literal in the object program is the string of characters itself, except:

1. The delimiting quotation marks are excluded.

2. Each embedded pair of contiguous quotation marks represents a single quotation mark character.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals belong to the alphanumeric category of data items (refer to "PICTURE Clause" in Section 11.3.3).

Numeric Literals

A numeric literal is a character-string whose characters are selected from the digits '0' through '9', the plus sign, the minus sign, and/or the decimal point. Numeric literals of 1 through 18 digits in length are allowed. The rules for the formation of numeric literals are:

1. A literal must contain at least one digit.

2. A literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, it is considered positive.

3. A literal must not contain more than one decimal point. The decimal point can appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is considered to be an integer.

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal belongs to the numeric category of data items. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

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

8.7 FORMAT AND NOTATION

This manual uses certain principles of notation to represent the COBOL language. The general formats use this notation in describing the elements and requirements of each COBOL clause and statement. The syntax rules and general rules then describe the meanings and relationships of individual elements. The conventions governing these methods of explanation are discussed in this section.

8.7.1 Definition of a General Format

A General Format represents the proper arrangement of elements in a specific COBOL clause or statement. Throughout this manual, a general format accompanies each description of a clause or statement. When more than one specific arrangement is permitted, separate, numbered formats are shown.

Within a general format, the elements of the clause or statement (key words, connectives, and special characters) are shown in their proper sequence. This sequence must be used unless otherwise stated. Optional clauses are also shown and identified as such.

8.7.2 Definition of Syntax Rules

Syntax rules define and/or clarify the order in which words or elements are arranged to form larger units such as phrases, clauses, or statements. Syntax rules also impose restrictions on individual words or elements.

8.7.3 Definition of General Rules

General rules define and/or clarify the meaning or relationship of the meanings of an element or set of elements. They explain in detail the semantics of a statement and the effect of the statement on compilation and/or execution.

8.7.4 Format Notation

The general formats in this manual have a uniform system of notation. Although this notation is not part of COBOL, it is useful in describing COBOL. The following paragraphs explain this system of notation.

1. Reserved words, which have preassigned meanings in COBOL, are printed entirely in uppercase letters. If any reserved word is spelled incorrectly, it cannot be recognized as a reserved word and may cause an error in the program.

2. All underlined reserved words are key words; key words are required unless the portion of the format containing them is itself optional or unless they occur as one of the options enclosed by braces (refer to Paragraph 8). The absence or incorrect spelling of any key word is considered an error in the program. Reserved words that are <u>not</u> underlined can be included or omitted at the discretion of the programmer. These are optional words used only to improve readability, but when used, they must be spelled correctly.

3. Even though the characters '+', '-', '<', '>', and '=' are not underlined in formats, they, or reserved words equivalent to them, are required when those formats are used.

4. Words printed in lowercase letters represent information to be supplied by the programmer. The text accompanying the format defines all such words.

5. To facilitate references to them in text, some lowercase words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.

6. Certain entries in the formats consist of a capitalized word(s) followed by the word "clause" or "statement". The reference is to clauses or statements that are described in other formats.

7. Square brackets ([]) indicate that the enclosed item can be used or omitted, depending on the requirements of the particular program. When brackets contain a vertical list of two or more items, one or none of the items can be used.

8. The use of braces ({}) indicates that one of the options contained within the braces must be selected. When braces enclose a portion of a format and only one possibility is shown, the braces delimit that portion of the format to which a following ellipsis applies. (Refer to Paragraph 9.)

   When braces enclose options containing key words and an option containing only reserved words that are not key words (are not underlined), the latter is the default option which is implicitly selected if none of the others is selected.

9. The ellipsis (...) can show the omission of a portion of a source program. This meaning becomes apparent in context.

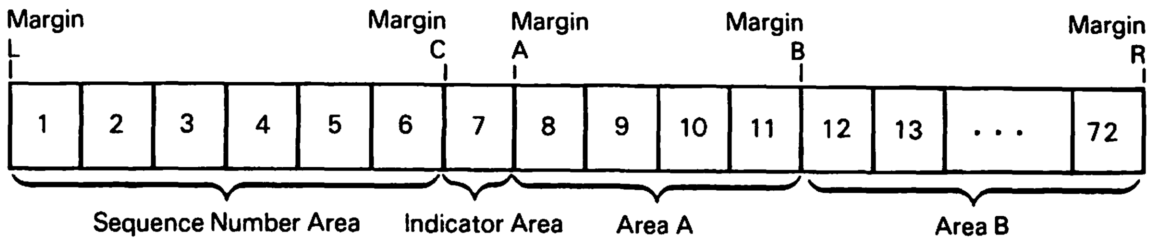
   In the general formats, an ellipsis indicates that the immediately preceding unit can occur one or more times in succession, at the programmer's option. A unit is either a single lowercase word, or a group of lowercase words and reserved words enclosed in brackets or braces. If a unit enclosed in brackets or braces is to be repeated, the entire unit must be repeated.

10. Since braces, brackets, and ellipses are not part of COBOL, their occurrence in a format does not indicate their occurrence in the COBOL code that employs that format. The occurrence of all other punctuation and special characters in a format does indicate their occurrence in the corresponding COBOL code. Additional punctuation can be used in COBOL code according to the rules specified in this manual.

11. Comments, restrictions, and clarifications on the use and meaning of every format appear in the associated text.

8.7.5 COBOL Source-program Reference Format

The COBOL reference format, which provides a standard method for describing COBOL source programs, is defined in terms of the character positions on a line on an input-output medium. The workstation screen is the medium for initial input to the VS COBOL compiler. Each 80-character line displayed corresponds conceptually to an 80-column punched card. The reference format for a line is:



The rules for spacing given in the discussion of the reference format take precedence over all other rules for spacing.

Margin L is immediately to the left of the leftmost character position of a line.

Margin C is between the 6th and 7th character positions of a line.

Area A occupies Character Positions 8 through 11. This area is reserved for the beginning of division headers, section names, paragraph names, level indicators, and certain level numbers.

Area B occupies Character Positions 12 through 72. It contains all remaining source code.

Margin R is immediately to the right of Column 72.

A sequence number, consisting of six digits in the sequence area, occupies Character Positions 1 through 6. This number labels each source program line (each 80-column record). The EDITOR assigns these numbers automatically when the COBOL source text is entered.

The program identification area, Columns 73 - 80 inclusively, may contain the 8-character program-ID (refer to Subsection 8.2.1, PROGRAM-ID Paragraph). The EDITOR offers this option at the time of source program creation.

The indicator area, Column 7, is used to indicate continuation lines and comment lines.

8.7.6 Continuation Lines

A contiunation line is indicated by a hyphen (-) in Column 7. The hyphen is used whenever a nonnumeric literal is extended onto the next line. The literal continues in Area B of the next line. The first nonblank character in Area B of the continuation line must be a quotation mark, and the continuation starts with the character immediately after that quotation mark. However, the line to be continued must not end in a quotation mark. The closing quotation mark appears only at the end of the entire nonnumeric literal that is being continued. For example,

```
          MOVE "THIS LITERAL IS CONT
     -      "INUED ON THE NEXT LINE".
```

Area A of the continuation line must be left blank.

8.7.7 Comment Lines

Any line that contains a '*' or '/' in Column 7 is a comment line. All characters following the '*' or '/' on the same line are reproduced on the source listing but serve as documentation only and do not affect compilation. The use of '/' causes that comment line to appear at the top of a new page.

A comment line can appear as any line in a source program after the Identification Division header. Any combination of characters from the ASCII character set may appear in Areas A and B of that line. Successive comment lines are allowed.

Whenever a percent sign, '%', is not part of a nonnumeric literal, any characters on a line after the percent sign are treated as comments. Thus comments can appear on the same line as source code.

8.7.8 Blank Lines

A blank line is one that is blank from Column 7 through Column 72, inclusively. Blank lines can appear anywhere in the source program except immediately following a continued line. They will appear as blank lines in a listing of the source code but have no effect on compilation.

8.7.9 Division, Section, and Paragraph Formats

The division header must be the first line in a division. The division header starts at the A margin with the division name, followed by a space, followed by the word DIVISION, followed by a period. Except in the case of the USING phrase (refer to Section 12.2.1, Procedure Division Header), no other text may appear on the same line as the division header.

A section name starts at the A margin of any line following the division header. The section name is followed by a space, the word SECTION, and a period. Except in the case of USE and COPY statements (refer to Section 12.5, Procedure Division Statements), no other text may appear on the same line as the section header.

Paragraph headers and paragraph names begin at the A margin of any line following the first line of a division or section and are terminated by a period followed by a space. The first sentence or entry in a paragraph begins either on the same line as the paragraph header or name, or at the B margin of the next nonblank line that is not a comment line. Successive sentences or entries either begin in the same line as the preceding sentence or entry, or at the B margin of the next nonblank line that is not a comment line.

8.7.10 Data Division Entries

Each Data Division entry begins either with the level indicator FD or with a level number, followed by a space, followed by the associated user-defined name, followed by a sequence of descriptive clauses. Each clause, except the last clause of an entry, can be terminated by either a comma or a semicolon. The last clause is always terminated by a period followed by a space.

In those Data Division entries that begin with the level indicator, the level indicator begins at the A margin, followed at the B margin by its associated user-defined name and appropriate descriptive clauses.

Those Data Division entries that begin with level numbers are called data description entries. At least one space must separate a level number from the word following the level number. In those data description entries that begin with a level number 01 or 77, the level number begins at the A margin, followed at the B margin by its associated user-defined name and appropriate descriptive clauses.

Successive data description entries may have the same format as the first or can be indented according to level number. Indentation does not affect the magnitude of the level number. When level numbers are to be indented, each new level number may begin any number of spaces to the right of Margin A.

CHAPTER 9
IDENTIFICATION DIVISION

9.1 GENERAL DESCRIPTION

The Identification Division must be the first division in every COBOL source program. This division identifies the source program and the resultant output listing. In addition, the user can include the date the program is written and such other comment information as is allowed in the paragraphs of the general format shown in Section 9.2.

9.2 ORGANIZATION

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 can be included in this division at the user's choice, in any order. (Section 9.2.1 describes the PROGRAM-ID paragraph. Section 9.2.2 describes the comment entry paragraphs.)

General Format

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

Syntax Rules

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

2. The comment entry can be any combination of the 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 can be contained on one or more lines.

3. The reserved words that may be used as headers for the comment-entry paragraphs are AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, and SECURITY.

9.2.1 PROGRAM-ID Paragraph

Function

The PROGRAM-ID paragraph gives the name by which a program is identified.

General Format

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

Syntax Rules

1. The program name must conform to the rules for the formation of a user-defined word. Each character of a user-defined word is selected from the set of characters 'A', 'B', 'C',...'Z', '0', '1',...'9', and the hyphen, except that the hyphen cannot be the first or last character. The hyphen used in any other position in the program name results in a compiler warning only. The compiler warns that the incorrect program name has been corrected. If a hyphen is used in the program name, the corrected object program name contains a '$' in place of the hyphen.

General Rules

1. The PROGRAM-ID paragraph must contain the name of the program and must be present in every program.

2. The program name identifies the source program and all listings pertaining to a particular program.

3. The first eight characters of the program name are used to identify the object program. The characters must be alphabetic or numeric.

4. The program name cannot be a reserved word.

5. If two or more programs are to be linked, the first seven characters of their program names must not be the same.

9.2.2 Comment-entry Paragraphs

Comment-entry paragraphs, designated in the general format description, serve as useful documentation, but are optional.

DATE-COMPILED Paragraph

Function

The DATE-COMPILED paragraph provides the compilation date in the Identification Division source program listing.

General Format

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

Syntax Rules

1. The comment entry can be any combination of the characters from the computer's character set. The continuation of the comment entry by the use of the hyphen in the continuation column (Column 7) is not permitted; however, the comment entry can be contained on one or more lines.

General Rules

1. The paragraph name DATE-COMPILED causes the current date to be inserted during compilation. If a DATE-COMPILED paragraph is present, it is replaced during compilation with a paragraph of the form:

   DATE-COMPILED. current date

   Current date is of the format:

   DAYXXXXX MMM DD, YYYY as in

   FRIDAY JAN 05, 1984.

9.3 Example of IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  SAMPLE1.

AUTHOR.  P. T. LORD.

INSTALLATION.  WANG LABORATORIES, INC.
               LOWELL, MASS.

DATE-WRITTEN. JAN. 5, 1984.

DATE-COMPILED.

SECURITY.  THIS PROGRAM MAY ONLY BE
             EXECUTED BY PAYROLL DEPARTMENT
             PERSONNEL.
```

CHAPTER 10
ENVIRONMENT DIVISION

10.1 GENERAL DESCRIPTION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows identification of the compiling computer and the executing computer. In addition, information relating to input/output control, special hardware characteristics and control techniques can be given.

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

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section.

The Configuration Section deals with the characteristics of the source computer and the object computer. This section is divided into four 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 the implementor names to user-defined names; and the FIGURATIVE-CONSTANTS paragraph, which associates data names with hexadecimal values.

The Input-Output Section deals with the information needed to control the transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph, which names each file used in the program and allows specification of other file-related information; and the I-O-CONTROL paragraph, which defines special control techniques to be employed in the object program.

10.2 ORGANIZATION

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

General Format

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

Syntax Rules

1. The Environment Division begins with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

10.2.1 Configuration Section

SOURCE-COMPUTER Paragraph

Function

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

General Format

```
SOURCE-COMPUTER.   WANG-VS [WITH DEBUGGING MODE].
```

Syntax Rule

1. If USE FOR DEBUGGING is present in the Declaratives Section of the Procedure Divison, the WITH DEBUGGING MODE clause must be included in the SOURCE-COMPUTER paragraph (refer to Section 13.2.2, Compile Time Switch--WITH DEBUGGING MODE).

OBJECT-COMPUTER Paragraph

Function

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed. The program collating sequence defaults to ASCII. This paragraph is optional and is treated as a comment.

General Format

```
                                                  ┌ WORDS      ┐
OBJECT-COMPUTER.   WANG-VS  [ MEMORY SIZE integer { CHARACTERS } ]
                                                  └ MODULES    ┘

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

Example of SOURCE-COMPUTER and OBJECT-COMPUTER Paragraphs

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.  WANG-VS.

OBJECT-COMPUTER.  WANG-VS.
```

SPECIAL-NAMES Paragraph

Function

The SPECIAL-NAMES paragraph provides a means of changing the representation of the currency sign and the use of decimal points and commas in PICTUREs and numeric literals. Thus, the SPECIAL-NAMES paragraph facilites the representation of foreign currency. It also specifies switches by relating implementor names used by the compiler to condition names used in the source program.

General Format

```
SPECIAL-NAMES.
    [SWITCH [IS mnemonic-name]                                                    ]
    [   { ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2] }        ] ...
    [   { OFF STATUS IS condition-name-3 [ON STATUS IS condition-name-4] }        ]
    [alphabet-name IS { STANDARD-1 }]
    [                 { NATIVE     }] ...
    [CURRENCY SIGN IS literal-1]
    [DECIMAL-POINT IS COMMA].
```

General Rules

1. No more than seven switches can be declared in a run unit.

2. If any switches are declared in a program, a screen listing the switch numbers appears on the workstation each time the program is executed. The screen prompts the user to indicate which switches are ON by placing a nonblank character beside those numbers. When a run unit is composed of a single program, this screen appears before the execution of any of the statements in the program. When a run unit is composed of main and subprograms and at least one switch is declared in the main program, this screen appears before the execution of any of the statements in the main program. In the case of a run unit in which switches are declared only in subprograms, the screen does not appear until the first time one of the subprograms that declares a switch is called.

3. A mnemonic name specified in a switch declaration serves a documentary purpose only and cannot be used elsewhere in the program.

4. The condition names assigned in the ON and OFF STATUS clauses of the switch declaration may be referenced by the IF and PERFORM...UNTIL statements of the Procedure Division. However, references in the PERFORM...UNTIL statement serve no purpose since the switch is always set either one way or the other. When a statement referring to a switch condition is executed, the setting of the switch is tested. If the switch condition refers to an ON status and the corresponding switch has been set on in the switch-option screen, the result of the test is true; if the switch condition refers to an OFF status and the corresponding switch has not been set on, the result is true; otherwise, the result of the test is false.

5. The alphabet-name clause is treated as a comment since the NATIVE character code of the Wang VS is STANDARD-1 ASCII (American National Standard Code for Information Interchange X3.4-1968).

6. The literal that 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.

   a. Digits 0 through 9

   b. Alphabetic characters 'A', 'B', 'C', 'D', 'L', 'P', 'R', 'S', 'V', 'X', 'Z', or the space

   c. Special characters '*', '+', '-', ',', '.', ';', '(', ')', '"', '/', '='

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

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

<u>Example of SPECIAL-NAMES Paragraph</u>

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SPECIAL-NAMES.

    SWITCH-1 ON STATUS IS S1-ON

      CURRENCY SIGN IS "Y"

      DECIMAL-POINT IS COMMA.
```

FIGURATIVE-CONSTANTS Paragraph

Function

The FIGURATIVE-CONSTANTS paragraph allows the user to define additional figurative constants by associating data names with hexadecimal values. User-figurative constants can be employed to control the order area and Field Attribute Characters (FACs) of the display screen. They can also be employed to control printer functions such as sounding the alarm and advancing lines.

General Format

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

General Rules

1. Data-name-1 can be used anywhere a figurative constant can be used.

2. A hexadecimal value can be either two or four hexadecimal characters. The hexadecimal value must be enclosed in quotation marks. A hexadecimal character is any of the characters:

   0 1 2 3 4 5 6 7 8 9 A B C D E F.

3. The value of data-name-1 will be the specified hexadecimal value. Note that it takes two hexadecimal characters to represent one character location's value.

4. If a 2-character hexadecimal value is used in a VALUE IS clause or as the object of a MOVE verb, as many copies of the value as are needed to fill the target of the VALUE IS clause or the MOVE are placed into the data item referenced by data-name-1.

5. If a hexadecimal value is four characters long, data-name-1 acts as a 2-character literal. If the target of a MOVE or a VALUE clause is longer than the two characters, the remaining right positions are filled with SPACES.

6. Refer to Appendix C for the use of figurative-constants to control Field Attribute Characters of the workstation screen. Refer to Appendix D for the use of figurative-constants to control the workstation screen order area. Refer to Appendix F for the use of figurative-constants to control printer functions.

Example of FIGURATIVE-CONSTANTS Paragraph

```
        ENVIRONMENT DIVISION.

        CONFIGURATION SECTION.

        FIGURATIVE-CONSTANTS.

            STARS    IS "2A",
            DASHES   IS "2D",
            EXCLM-PT IS "21".
```

10.2.2 Input-Output Section

FILE-CONTROL Paragraph

Function

The FILE-CONTROL paragraph names the files to be used in the program, associates them with external media, and can supply other information pertinent to the use of the files.

General Format

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

FILE-CONTROL Entry -- for Consecutive Files

Function

The FILE-CONTROL entry names a file and may specify other file-related information.

General Format

```
SELECT file-name

        ASSIGN TO "parameter-reference-name" ["DISK"   ] [NODISPLAY]
                                             ["DISPLAY"]
                                             ["PRINTER"]
                                             ["TAPE"   ]

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

            [FILE STATUS IS data-name-2]

            [CURSOR POSITION IS data-name-3]

            [BUFFER SIZE IS integer-1 BLOCKS]

            [PFKEY IS data-name-4].
```

Syntax Rules

1. The SELECT clause must be specified first in the FILE-CONTROL entry. The clauses that follow the SELECT clause can appear in any order.

2. Neither the SELECT clause nor the ASSIGN clause can appear in Margin A. Both must be indented to Margin B.

3. Each file described in the Data Division must be named once and only once by a 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.

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

5. If the ORGANIZATION clause is not specified, the ORGANIZATION IS SEQUENTIAL clause is implied.

6. Data-name-1, data-name-2, and data-name-4 may be qualified.

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

8. The device type may be "DISK", "TAPE", "PRINTER", or "DISPLAY". If no type is specified, DISK is assumed.

9. The RELATIVE KEY clause is not permitted with device type "TAPE".

10. The CURSOR POSITION clause applies only when "DISPLAY" has been specified as the device type, indicating a workstation file.

11. Data-name-3 must be defined in the Data Division as an 01 level structure with two USAGE IS BINARY data items subordinate to it.

12. The PFKEY clause can be specified only for files with the device type "DISPLAY".

13. Data-name-4 must be defined in the Data Division as a numeric display data item of length two.

14. If the device type is "TAPE", the only ACCESS MODE that can be specified is SEQUENTIAL.

15. Parameter-reference-name is the external name for the file and is used to identify the request for file information when opened. The first character must be alphabetic and the entire name cannot exceed eight alphanumeric characters.

General Rules

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

2. 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.

3. When the FILE STATUS clause is specified, a value is moved into the data item specified by data-name-2 after every attempt to execute an Input/Output statement that references that file. This value indicates the condition obtaining at the completion of the attempt to execute the Input/Output statement, as explained in Section 2.8 and in more detail in Appendix E.

For workstation files, the rightmost character of the FILE STATUS data item indicates which key was selected by the operator to signal completion of input from the workstation. When prompted for input, the workstation operator can enter the required response and then signal completion of the input by pressing the ENTER key or the appropriate Program Function key. The rightmost character of the FILE STATUS data item contains an "at" sign (@) if the ENTER key was pressed, or an uppercase or lowercase letter from A to P depending on which Program Function key was pressed. Refer to "AID Characters" in Appendix E. The PFKEY clause, described in Paragraph 14, provides another means of indicating which Program Function key was pressed.

4. The PRNAME is the external name of the file, the name used to identify the request for file information at OPEN time.

5. When ACCESS MODE is SEQUENTIAL, 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 the WRITE statements when the file is created or extended.

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

7. When the ACCESS MODE is DYNAMIC, records in the file can be accessed in SEQUENTIAL and/or RANDOM order.

8. All records stored in a consecutive 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 1, and subsequent logical records have relative record numbers of 2, 3, 4,... .

9. When the Workstation I/O method is used to control the workstation screen, the RELATIVE KEY data item identifies the CRT row at which a READ or REWRITE of a workstation file record begins. The legal values for this data item are 1 through 24. An attempt to write a record that extends beyond the maximum size of the workstation file will cause a runtime error.

10. The BUFFER SIZE clause specifies the size of the buffer to be used, in 2K blocks. If not specified, the default buffer size of one block is assumed. If specified, integer-1 must be between 1 and 9, inclusive.

11. The data item specified by data-name-1 is used to communicate a relative record number between the user and operating system. This data item must be defined as an unsigned integer.

12. When the CURSOR POSITION clause is specified, the operating system moves a value into the data item specified by data-name-3 after the execution of every statement that references the display screen. The value produced represents the current column (first byte) and row (second byte) position of the cursor.

13. If NODISPLAY is specified, no prompting occurs when all necessary file location information has been specified through the program or the Procedure language. PRINTER and DISPLAY files default to NODISPLAY.

14. The PFKEY clause identifies a data name that is to receive the numeric value of the selected PF (Program Function) key following execution of a DISPLAY AND READ statement. The range of values is 00 - 32, where 00 identifies the ENTER key.

Example of Input-Output Section for Consecutive Files

```
INPUT-OUTPUT SECTION.

  FILE-CONTROL.

      SELECT CRT

        ASSIGN TO "CRT" "DISPLAY"
        ORGANIZATION    IS SEQUENTIAL
        ACCESS MODE     IS RANDOM
        CURSOR POSITION IS CURSOR-POS
        FILE STATUS     IS FILE-STAT.
```

FILE-CONTROL Entry -- for Indexed Files

Function

The FILE-CONTROL entry names a file and can specify other file-related information.

General Format

```
SELECT file-name

    ASSIGN TO "parameter-reference-name"  [ "DISK" ]  [NODISPLAY]

        ORGANIZATION IS INDEXED
        [                 ⎧SEQUENTIAL⎫ ]
        [ ACCESS MODE IS  ⎨RANDOM    ⎬ ] RECORD KEY IS data-name-1
        [                 ⎩DYNAMIC   ⎭ ]
        [ALTERNATE RECORD KEY  integer-1 IS  data-name-2    [WITH DUPLICATES]
                                [[integer-2 IS] data-name-3 [WITH DUPLICATES]] ...]
        [FILE STATUS IS data-name-4]
        [                    [AREA ] ]
        [ RESERVE integer-3  [AREAS] ] .
```

Syntax Rules

1. The SELECT clause must be specified first in the FILE-CONTROL entry. The clauses that follow the SELECT clause can appear in any order.

2. Neither the SELECT nor the ASSIGN clause can appear in Margin A. Both clauses must be indented to Margin B.

3. Each file described in the Data Division must be named once and only once by a 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.

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

5. The ORGANIZATION clause must be specified for indexed files.

6. The RECORD KEY data item can be defined as an elementary or group DISPLAY item within a record description entry associated with the specified file name. Records are logically ordered for retrieval according to the byte-by-byte internal collating sequence.

7. Data-name-1 through data-name-4 may be qualified.

8. Data-name-4 must be defined in the Data Division as a 2-character data item of the category alphanumeric and must not be defined in the File Section.

9. The device type must be "DISK".

10. The data items referenced by data-name-1, data-name-2, and data-name-3 must each be defined within a record description entry associated with the specified file.

11. Data-name-2, data-name-3,... cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item reference by data-name-1 or by any other data-name-2, data-name-3,... associated with this file.

12. Integer-1 and integer-2 can be any number from 1 through 16.

13. Integer-3 can be any number from 3 through 60.

14. Parameter-reference-name is the external name for the file and is used to identify the request for file information when opened. The first character must be alphabetic and the entire name cannot exceed eight alphanumeric characters.

General Rules

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

2. 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.

3. When the ACCESS MODE is SEQUENTIAL, records in the file are accessed in the sequence dictated by the file order. For indexed files, this sequence is the order of ascending prime record key values.

4. If the ACCESS MODE is RANDOM, the value of the record key data item indicates the record to be accessed.

5. When the ACCESS MODE is DYNAMIC, records in the file can be accessed in SEQUENTIAL or RANDOM order.

6. When the FILE STATUS clause is specified, a value is moved into the data item specified by data-name-4 after the execution of every input/output statement that references that file. This value indicates the execution status of the statement. General use of this value is explained in Section 2.8. Refer to Appendix E for a detailed explanation.

7. The PRNAME is the external name of the file, the name used to identify the request for file information at OPEN time.

8. The RESERVE clause allows the user to improve performance when processing indexed files randomly or dynamically by specifying buffer pooling. Integer-3 specifies the number of blocks to be reserved for the buffer pool and to be used by the indexed files referred to in the SAME AREA clause of the I-O-CONTROL paragraph (refer to "SAME AREA clause" in "I-O-CONTROL paragraph", later in this subsection).

9. The RECORD KEY clause specifies the record key that is the primary key for the file. The values of the primary key must be unique among records of the file. The prime record key provides an access path to records in an indexed file.

10. The ALTERNATE RECORD KEY clause specifies the record keys within each record type that provide alternate access paths into the indexed file. Each key listed need appear in no more than one record type. However, a key that is present in more than one record type must be assigned the same ordinal number and must occupy the same relative position in each record where it occurs.

11. The data descriptions of data-name-2 and data-name-3,... their relative positions within a record, and the ordinal numbers assigned to key fields data-name-2 and data-name-3,... must be the same as those used when the file was created. The alternate keys listed for the file must be equivalent to, or a subset of, those keys used when the file was created.

12. The DUPLICATES clause specifies that the value of the associated alternate record key can be duplicated within any of the records in the file. If the DUPLICATES clause is not specified, the value of the associated alternate record key must not be duplicated among any of the records accessible through this alternate record key field.

13. The ALTERNATE RECORD KEYs identified by data-name-2 and data-name-3 can be of any data usage, but are in any case ordered within the file according to the byte-by-byte internal collating sequence.

14. The length of the primary RECORD KEY plus the longest ALTERNATE RECORD KEY cannot exceed 256 characters.

15. If NODISPLAY is specified, no prompting occurs when all necessary file location information has been specified through the program or the operating system's commands.

Example of Input-Output Section for Indexed Files

```
        INPUT-OUTPUT SECTION.

          FILE-CONTROL.

              SELECT CLIENT-FILE

                 ASSIGN TO "CLIFILE" "DISK"
                 ORGANIZATION IS INDEXED
                 ACCESS MODE IS RANDOM
                 RECORD KEY IS SS-NUMBER
                 ALTERNATE RECORD KEY 01 IS STATE-NAME WITH DUPLICATES
                 FILE STATUS IS FILESTAT
                 RESERVE 16 AREAS.
```

FILE-CONTROL Entry -- for Relative Files

Function

The FILE-CONTROL entry names a file and can specify other file-related information.

General Format

```
SELECT file-name

    ASSIGN TO "parameter-reference-name" ["DISK"] [NODISPLAY]

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

    ORGANIZATION IS RELATIVE

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

    [FILE STATUS IS data-name-2]

    [BUFFER SIZE IS integer-2 BLOCKS] .
```

Syntax Rules

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

2. Parameter-reference-name is the external name for the file and is used to identify the request for file information when opened. The first character must be alphabetic and the entire name cannot exceed eight alphanumeric characters.

3. "DISK" is the device type "parameter-reference-name" is assigned to and is the only device type allowed for relative files.

4. Each file described in the Data Division must be named once and only once by a 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.

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

6. Data-name-1 and data-name-2 may be qualified.

7. If a relative file is to be referenced by a START statement, the RELATIVE KEY phrase must be used.

8. Data-name-1 must not be defined in a record description entry associated with the file-name.

9. The data item referenced by data-name-1 must be defined as an integer.

10. Data-name-2 must be defined in the Data Division as a two-character, alphanumeric data item and must not be defined in the File Section.

11. The BUFFER SIZE clause specifies the size of the buffer to be used, in 2K blocks. If this clause is not used, the default buffer size of one block is assumed. If specified, integer-2 must be between 1 and 9, inclusive.

General Rules

1. The ASSIGN clause specifies the association of the file to a storage medium.

2. If NODISPLAY is used, no prompting occurs when all necessary and correct file location information has been specified through the program or procedure when the file is opened. If the information is incomplete or incorrect, prompting for a correction occurs regardless of whether NODISPLAY is specified.

3. The RESERVE clause is treated as a comment.

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

5. When the ACCESS MODE IS sequential, records in the file are accessed in the order of ascending relative record numbers of the existing records in the file.

6. If the access mode is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

7. If the access mode is dynamic, records in the file may be accessed sequentially and/or randomly.

8. When the FILE STATUS clause is specified, a value is moved by the operating system into the data item specified by data-name-2 after the execution of every I/O statement that references that file, either explicitly or implicitly. This value indicates the status of execution of the statement. Refer to Appendix E, File Status Key Values.

9. 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). Subsequent logical records have relative record numbers of 2, 3, 4, etc.

10. The data item specified by data-name-1 is used to communicate a relative record number between the user and the operating system.

Example of Input-Output Section for Relative Files

```
INPUT-OUTPUT SECTION.

    FILE-CONTROL.

    SELECT REL-FIL
    ASSIGN TO "R-FIL1"  "DISK"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS DYNAMIC
    RELATIVE KEY IS REL-KEY
    FILE STATUS R-STAT.
```

FILE-CONTROL Entry -- for Sort, Merge files

Function

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

General Format

```
SELECT file-name

    ASSIGN TO "parameter-reference-name" [ "DISK" ] [ NODISPLAY ]
                                         [ "TAPE" ]

    [BUFFER SIZE IS integer BLOCKS] .
```

Syntax Rules

1. Each Sort or Merge file described in the Data Division must be named only once in the FILE-CONTROL paragraph. Each sort or merge file specified must have a sort or merge file description entry in the Data Division.

General Rules

1. The ASSIGN clause specifies the association of the sort or merge file referenced by file-name to a storage medium. The default is "DISK".

2. If the file specified by file-name is specified in the USING phrase of the SORT statement, it must be described either implicitly or explicitly as having sequential or indexed organization.

3. If the file specified by file-name is specified in the GIVING phrase of a SORT statement, it must be described either implicitly or explicitly as having sequential organization.

4. The BUFFER SIZE clause specifies the size of the buffer to be used, in 2K blocks. If this clause is not used, the default buffer size of one block is assumed. If specified, integer must be between 1 and 9, inclusive.

I-O-CONTROL Paragraph

Function

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

General Format

```
I-O-CONTROL
     [ RERUN [ ON file-name-1 ] EVERY integer-1 RECORDS OF file-name-2 ]

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

Syntax Rules

1. The I-O-CONTROL paragraph is optional.

2. The RERUN clause is treated as a comment by the compiler.

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

4. A file-name must not appear in more than one SAME RECORD AREA clause.

5. The SAME AREA clause is treated as a comment for consecutive and relative files.

General Rules

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

2. The use of RECORD in the SAME RECORD AREA clause causes a single buffer area to be used for logical records of two or more files. This is not buffer pooling. Buffer pooling requires that RECORD be omitted. (Refer to General Rule 3.)

3. The SAME AREA clause (with RECORD omitted) indicates that the specified files are to share a buffer pool. All specified files should be INDEXED files, other files are ignored. One of these files should have a RESERVE clause in its SELECT statement. If more than one of these files has a RESERVE clause in its SELECT statement, the one with the largest number is used. If none of these files has a RESERVE clause in its SELECT statement, three buffers are reserved.

Example of I-O-CONTROL Paragraph

```
INPUT-OUTPUT SECTION.

I-O-CONTROL.

    SAME RECORD AREA FOR CLIENT-FILE, LIST-B-FILE.
```

CHAPTER 11
DATA DIVISION

11.1 COMPUTER INDEPENDENT DATA DESCRIPTION

To make data as computer independent as possible, the characteristics or properties of the data are described in terms of a standard data format oriented to the appearance of the data on a printed page, rather than to the electronic storage of data. This standard data format is designed for general data processing applications and uses the decimal system to represent numbers (regardless of the radix used by the computer) and the remaining characters in the COBOL character set to describe nonnumeric data items.
(Refer to "PICTURE Clause" in Section 11.3.3.)

11.1.1 Logical and Physical Records

A logical record is a group of related data that is uniquely identifiable as a unit. In COBOL, a record description entry defines the logical record. I/O processing in COBOL using the WRITE, REWRITE, and DELETE statements, manipulates the individual records. A READ or START issued against a file retrieves/locates one logical record at a time.

A single logical record can be contained within a single physical unit; or several logical records can be contained within a single physical unit.

A physical record is a physical unit of information whose size and recording mode are determined by the requirements of a particular hardware configuration (the computer and the input or output device employed). A physical record on the VS is called a block. For disk files, each block contains 2K bytes.

Buffering Concepts

Blocks are grouped into buffers. A buffer is a storage area that is used temporarily when performing I/O operations. When a file is opened, block(s) of data are loaded from the storage device into the buffer area. Each READ retrieves a record from the buffer area and each WRITE writes a record to the buffer.

VS COBOL allows the programmer to choose between three buffering strategies.

1. Accept the default strategy that allocates one 2K block per buffer for consecutive files and two 2K blocks per buffer for indexed files.

2. Specify buffer pooling for indexed files not opened in the shared mode through the RESERVE clause in the file-control entry and the SAME AREA clause in the I-O-CONTROL paragraph.

3. Select a large buffer strategy for consecutive files through the BUFFER SIZE clause of the file control entry. The large buffer strategy allows the programmer to override the default strategy of one block per buffer. Up to 18 2K blocks per buffer can be allocated for disk files.

Record Concepts

The record description consists of a set of data description entries that describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data name, if required, followed by a series of independent clauses, as required. A record description has a hierarchical structure, therefore, the clauses used with an entry can vary considerably, depending upon whether or not they are followed by subordinate entries.

For indexed files, the data names of alternate keys specified in a particular record description determine the alternate access paths (index structures) through which a record will become accessible when that particular description is specified in a WRITE or REWRITE statement.

11.1.2 Concept of Levels

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it can be further subdivided to permit more detailed data referral.

The most basic subdivisions of a record--those not further subdivided--are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself can be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, can be combined into groups of two or more groups. Thus, an elementary item can belong to more than one group.

Level Numbers

A system of level numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, record level numbers start at 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater in value than 49. The special level numbers 77 and 88 (discussed later in this subsection) are exceptions to this rule. Separate entries are written in the source program for each level number used.

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. All items immediately subordinate to a given group item must be described using identical level numbers greater than the level number used to describe that group item.

The level number of an item that immediately follows the last elementary item of a group must be equal to the number of one of the groups to which the elementary item belongs. If the level number of the following item is not equal to one of the group level numbers preceding it, it is interpreted as equal to the first preceding group level number that is larger than it. For example, in

```
01  EMPLOYEE-DATA.
    03 EMPLOYEE-NAME.
        05 EMPLOYEE-ADDRESS.
            07 STREET PIC X(20).
            07 CITY-STATE PIC X(20).
        04 EMPLOYEE-NUMBER PIC X(9).
```

04 is treated by the compiler as if it had been written as 05.

There is no true concept of level for entries that specify noncontiguous working-storage and linkage data items that are not subdivisions of other items and are not themselves subdivided. These entries have been assigned the special level number 77.

01 level entries are fullword aligned. 77 level entries may not be fullword aligned.

Entries that specify conditions names, to be associated with particular values of a conditional variable, have been assigned the special level number 88. A level 88 entry can specify either a single value or a range of values.

11.1.3 Classes of Data

The five categories of data items (refer to "PICTURE Clause" in Section 11.3.3) are grouped into three classes: alphabetic, numeric, and alphanumeric. The classes and categories are synonymous for alphabetic and numeric. The alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (not edited). Every elementary item except 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 table depicts the relationship of the class and categories of data items:

Classes and Categories of Data

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

11.1.4 Character Representation and Radix

The VS COBOL compiler represents the value of numeric items whose usage is DISPLAY as ASCII characters; numeric items whose usage is COMPUTATIONAL are represented in packed decimal format. Items whose usage is INDEX, and items whose usage is BINARY, are represented in half-word aligned 16-bit signed binary numbers.

The size of an elementary data item or a group item is the number of characters in standard data format of the item. Synchronization and usage can cause a difference between this size and the actual number of characters required for the internal representation.

11.1.5 Algebraic Signs

Algebraic signs fall into two categories: (1) operational signs, which are associated with signed numeric data items and signed numeric literals to indicate their algebraic properties; and (2) editing signs, which appear on edited reports to identify the sign of the item.

The SIGN clause permits the programmer to explicitly state the location of the operational sign. This clause is optional; if it is not used, usage of DISPLAY is assumed and operational signs are represented as if SIGN IS TRAILING SEPARATE CHARACTER was specified.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

11.1.6 Standard Alignment Rules

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

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

   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 described in Paragraph 1a.

2. If the receiving data item is numeric edited, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required. The only exception occurs when editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, data is moved to the receiving character positions and aligned at the leftmost character position in the receiving data item. Space fill or truncation to the right is supplied as required.

If the JUSTIFIED clause is specified for the receiving item, these standard rules are modified as described in "JUSTIFIED Clause" in Section 11.3.3.

11.2 METHODS OF DATA REFERENCE

11.2.1 Qualification, The CORRESPONDING Phrase, and Subscripting

Qualification

All user-specified names that define data elements must be unique. Uniqueness is achieved if each name has either unique spelling, hyphenation, or qualification. A data-name is qualified when it is part of a hierarchy of names such that references to the name can be unique by mentioning one or more of the higher levels of the hierarchy. Within the Data Division, all data-names used for qualification must be associated with a level indicator or level-number.

In a hierarchy, names associated with a level indicatior are the most significant. The next most significant are names associated with level-number 01, followed by level-number 02, and so on. A section-name is the highest (and only) qualifier for a paragraph name. Thus, the most significant name in a hierarchy must be unique and cannot be qualified.

Subscripted or indexed data-names, conditional variables, procedure-names, and data-names can be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names. Regardless of the available qualification, no name can be both a data-name and a procedure-name.

Referencing qualified names is accomplished by coding a data-name, condition-name, paragraph-name, or text-name followed by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

General Format

Format 1

$$\left\{\begin{array}{l}\text{data-name-1}\\ \text{condition-name}\end{array}\right\}\left[\left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\end{array}\right\}\text{data-name-2}\right]\ldots$$

Format 2

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

Format 3

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

General Rules

1. Each qualifier must be a successively higher level and be 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 or condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referenced in the Procedure, Environment, and Data Divisions.

4. Qualification cannot be used in a REDEFINES clause.

5. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION cannot appear. A paragraph-name need not be qualified when it is referred to from within the same section.

6. A data-name cannot be subscripted when it is being used as a qualifier.

7. A name can be qualified even if it is not necessary.

8. If more than one set of qualifiers exists that ensure uniqueness, any such set can be used for referrence. The complete set of qualifiers for a data-name must not be the same as any partial set for another data-name. Qualified data-names may have any number of qualifiers.

9. If more than one COBOL library is available to the compiler during compilation, text-names must be qualified each time they are referenced. If the volume and/or library is invalid, a GETPARM screen is generated by the system to allow the user to respecify the required information.

<u>Corresponding</u>

The CORRESPONDING phrase allows the COBOL programmer to add, substract, or move data items from one group item to some other group item. The CORRESPONDING phrase eliminates the necessity of coding a separate ADD, SUBTRACT, or MOVE Statement for each corrrsponding date item. To use this feature, the programmer codes the statements to add, subtract, or move <u>group items</u>. Upon execution, only those items that are corresponding are actually added, subtracted, or moved.

A pair of data items are corresponding if:

1. Each is subordinate to a separate group item.

2. They have the same data-name and qualifiers up to, but not including, the group item names.

3. Either of the group items they are subordinate to can have REDEFINES or OCCURS clauses. In addition, each of the group items can themselves be subordinate to data items with REDEFINES or OCCURS clauses.

4. At least one of the data items must be an elementary item when used in a MOVE statement with the CORRESPONDING phrase.

5. Both data items must be elementary items when used in the ADD or SUBTRACT statements with the CORRESPONDING phrase.

A pair of data items are <u>not</u> corresponding if:

1. The common data name is FILLER.
2. Either data name contains a REDEFINES, OCCURS, or USAGE IS INDEXED clause.
3. Either data item is subordinate to a data item containing the REDEFINES, OCCURS, or USAGE IS INDEXED clauses.
4. The description of either group item contains a level-number of 77, 88, or the USAGE IS INDEXED clause.

Subscripting

Subscripts can be used only when referring to an individual element within a list or table of like elements that have not been assigned individual data names (refer to "OCCURS Clause" in Section 11.3.3).

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.

The subscript can be signed; 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 2,3,4... . 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 SUBCHK compiler option generates special code that checks the ranges of subscripts during program execution and causes a program interrupt if a subscript exceeds its defined length.

The subscript, or set of subscripts, that identifies the table element is delimited by the balanced pair of separators, the left 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.

The format for subscripting is:

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

11.2.2 Indexing

References can be made to individual elements within a table of like elements by specifying an index 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.

For example, a variable having one index could be established by the following:

```
01  ACCOUNTS-TABLE.
    03  ACCOUNT PIC 9(5) OCCURS 20 TIMES
        INDEXED BY ACCOUNT-INDEX.
```

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 that element's occurrence number. An index name must be initialized before it is used as a table reference. An index name can be given an initial value by either a SET or a Format 4 PERFORM statement.

Direct indexing is specified by using an index name in the form of a subscript. For example,

```
MOVE ACCOUNT (ACCOUNT-INDEX) TO ACCOUNT-OUT.
```

This will cause the entry in ACCOUNTS-TABLE (specified by the current value of ACCOUNT-INDEX) to be moved to ACCOUNT-OUT.

Relative indexing is specified when the index name is followed by a '+' or '-', an unsigned numeric literal, and then delimited by the pair of separators (left and right parentheses) following the table-element data name. The resulting occurrence number is determined by increasing (when the '+' is used) or decreasing (when the '-' is used) the occurrence number represented by the value of the index by the value of the literal. When more than one index name is required, they are written in the order of successively less inclusive dimensions of the data organization.

In the following example, the sum of the current value of ACCOUNT-INDEX plus 3 is used to select one element of ACCOUNTS-TABLE.

```
MOVE ACCOUNT (ACCOUNT-INDEX + 3) TO ACCOUNT-OUT
```

When executing a statement that refers to an indexed table element, the value contained in the referenced index must correspond to a numeric value between one and the highest permissible element occurrence number. This restriction also applies to the value resultant from relative indexing.

Restrictions on subscripting and indexing are:

1. A data name must not itself be subscripted nor indexed when it is being used as an index or subscript.

2. Indexing is not permitted where subscripting is not permitted.

3. An index can be modified only by the SET, SEARCH, or PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values associated with index names. Such data items are called INDEX data items. The internal storage representation of an INDEX data item is binary.

4. Literal-1, literal-3, and literal-5 in the general format for indexing (which follows immediately) must be positive numeric integers. Literal-2, literal-4, and literal-6 must be unsigned integers.

The general format for indexing is:

$$\left\{\begin{array}{l}\text{data-name}\\ \text{condition-name}\end{array}\right\} \; \Bigg(\left\{\begin{array}{l}\text{index-name-1 } [\;\{\pm\}\text{ literal-2}]\\ \text{literal-1}\end{array}\right\}$$

$$\left[\left\{\begin{array}{l}\text{index-name-2 } [\;\{\pm\}\text{ literal-4}]\\ \text{literal-3}\end{array}\right\} \left[\left\{\begin{array}{l}\text{index-name-3 } [\;\{\pm\}\text{ literal-6}]\\ \text{literal-5}\end{array}\right\}\right]\right] \Bigg)$$

11.2.3 Condition-Names

A condition-name specifies 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. Each condition-name must be unique, or made unique through qualification and/or subscripting. The exception to this rule is when programs are directly or indirectly contained within other programs. In this situation, identical user-defined words can exist in both programs.

If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. The hierarchy of names associated with the conditional variable itself must also 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 of and restrictions on the combined use of qualification and subscripting of condition-names is exactly that of "identifier" except that "data-name-1" is replaced by "condition-name-1".

11.2.4 Identifiers

An identifier is a term used to reflect a data-name that, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or reference modifiers necessary to make it unique.

The general formats for identifiers are:

Format 1

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

Format 2

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

                                          [ { index-name-2 [ { + } literal-4] }
                                          [ { literal-3      { - }            }

                                          [ { index-name-3 [ { + } literal-6] } ] ] ) ]
                                          [ { literal-5      { - }            } ] ] ) ]
```

11.3 ORGANIZATION

The Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. A required division, it is subdivided into three sections: File, Working-Storage, and Linkage.

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 that 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 appears in the called program and describes data items that are to be referred to by the calling program and the called program. Its structure is the same as the Working-Storage Section describing the parameters passed by the calling program.

The following are the general formats of the sections in the Data Division. Level 77 items need not be restricted to the first entries in any of the sections.

```
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] ...]
```

11.3.1 File Section

In a COBOL program, the file description (FD) entry represents the highest level of organization in the File Section. The File Section header is followed by a FD entry consisting of a level indicator, a file name, and a series of independent clauses. The FD clauses specify the size of the logical and physical records, and the names of the data records that make up the file. The entry itself is terminated by a period.

File Description Entry

Function

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

General Format

```
FD file-name

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

    [RECORD CONTAINS [integer-2 TO] integer-3 [COMPRESSED] CHARACTERS]

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

[            { [FILENAME IS            { data-name-1 }]   }]
[            {                         { literal-1   }    }]
[            { [LIBRARY IS             { data-name-2 }]   }]
[            {                         { literal-2   }    }]
[            { [VOLUME IS              { data-name-3 }]   }]
[            {                         { literal-3   }    }]
[            { [SPACE IS               data-name-4]       }]
[ VALUE OF   { [POSITION IS            data-name-5]       }]
[            { [INDEX AREA IS          data-name-6]       }]
[            { [DATA AREA IS           data-name-7]       }]
[            { [RECOVERY-BLOCKS IS     { data-name-8 }]   }]
[            {                         { literal-4   }    }]
[            { [RECOVERY-STATUS IS     data-name-9]       }]
[            { [DATABASE-NAME IS       data-name-10]...]  }]

[DATA { RECORD IS    } data-name-11 [data-name-12]...]
      { RECORDS ARE  }

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

Syntax Rules

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

2. The clauses that follow the name of the file are optional in many cases, and their order of appearance is immaterial.

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

BLOCK CONTAINS Clause

Function

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

General Format

$$\underline{\text{BLOCK}}\ \text{CONTAINS [integer-1}\ \underline{\text{TO}}\text{] integer-2}\left\{\begin{array}{l}\underline{\text{RECORDS}}\\ \text{CHARACTERS}\end{array}\right\}$$

General Rules

1. The BLOCK CONTAINS clause is optional. If it is specified, and the BUFFER SIZE clause is also specified in the corresponding File Control entry (Section 9.2.2), the BLOCK CONTAINS clause is used to determine the number of records or characters to be allocated for the buffer. If the BUFFER SIZE clause is not also specified, the BLOCK CONTAINS clause is treated as a comment, except in the case of tape files.

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

3. The BLOCK CONTAINS clause is ignored for disk files, which are automatically blocked in 2K blocks. For tape files, the default condition allocates a blocksize equal to the record size.

4. Integer-1 is ignored by the compiler. However, if it is used, it must be less than Integer-2. Specifying integer-1 does not imply the existence of a variable length record; this is accomplished by the RECORD CONTAINS clause.

CODE-SET Clause

Function

The CODE-SET clause specifies the character code set used to represent data on the external media. The Wang VS uses the ASCII code set, and the CODE-SET clause is treated as a comment.

General Format

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

DATA RECORDS Clause

Function

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

General Format

$$\underline{\text{DATA}} \begin{Bmatrix} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{Bmatrix} \text{data-name-1 [data-name-2]...}$$

Syntax Rules

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

General Rules

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

2. Conceptually, all data records within a file share the same area, the presence of more than one type of data record within the file notwithstanding.

LABEL RECORDS Clause

Function

The LABEL RECORDS clause specifies whether or not labels are present.

General Format

$$\underline{\text{LABEL}} \left\{ \begin{matrix} \underline{\text{RECORD}}\ \text{IS} \\ \underline{\text{RECORDS}}\ \text{ARE} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \end{matrix} \right\}$$

Syntax Rules

1. If this clause is not coded, standard labels are provided.

General Rules

1. OMITTED specifies that no labels exist for the file.

2. STANDARD specifies that labels exist for the file. If device-type is TAPE, the labels provided conform to ANSI (ANSI X3.27-1969) specifications; both IBM and Wang labels are allowed. If device type is nontape, labels are provided as specified in VS Operating System Services.

3. The OMITTED clause is ignored for disk files.

RECORD CONTAINS Clause

Function

The RECORD CONTAINS clause specifies the size of fixed- or variable-length data records.

General Format

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

Syntax Rules

1. The maximum record lengths (maximum values for integer-2) are:

DISK FILES	NUMBER OF CHARACTERS
consecutive fixed	2048
consecutive variable	2024
indexed fixed	2040
indexed variable	2024
relative, fixed or variable	2040
TAPE FILES	32767 (with a minimum of 12)
PRINTER FILES	240
WORKSTATION DISPLAY	1924

General Rules

1. The RECORD CONTAINS clause with the integer-1 TO and COMPRESSED phrases omitted specifies that the file contains fixed-length records. Integer-2 is the record length of the file. Thus, integer-2 should equal the greatest length specified in the record description entries for the file.

2. The RECORD CONTAINS clause with the integer-1 TO phrase specifies that the file contains variable-length records. The length of the smallest data record, integer-1, and the length of the largest data record, integer-2, must be defined. Integer-1 must be smaller than integer-2.

3. 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 the record is determined by the sum of the number of characters in all elementary items.

4. If the RECORD CONTAINS clause is absent, the file contains fixed-length records. The record length is that of the longest record defined in a record description entry for the file.

5. The COMPRESSED option specifies compressed files. It is required when a file is being opened for output (created) and the user wants the file to be compressed. It is optional when reopening an old file, so the user need not know whether a file is compressed in order to use it. Compression is transparent to the user, i.e., the user's data area always contains noncompressed records, and the size always refers to noncompressed size. The following notes apply to the COMPRESSED option.

   a. If COMPRESSED is specified, the actual record as it is stored on the device medium is a variable-length record and is treated as such in I/O operations. This is not readily apparent to the user. It is the user's responsibility to ensure that the definition of a data file in a program is consistent with the attributes of the existing file. The results of not doing so are undefined.

   b. When COMPRESSED is specified, all character strings of three or more (up to 128) duplicate and consecutive characters are compressed into two bytes that contain the number of occurrences and the character itself. The byte containing the number of occurrences is referred to as a compression control byte.

   c. If COMPRESSED is specified and there is no repetition of characters within a string, compression control bytes are generated for each string of 128 nonrepetitive characters. These compression control bytes indicate that an uncompressed character string follows and contains the number of following nonrepetitive characters.

   d. Each set of bytes that indicate repetition is immediately followed by another set of bytes that indicate the compression characteristics of the succeeding characters.

   e. When the record is accessed, it is expanded to the size specified in the record description entry for the file.

   f. COMPRESSED is not allowed for relative files.

VALUE OF Clause

Function

The VALUE OF clause can provide file information required at runtime. When an OPEN is issued for a file, FILENAME, LIBRARY, and VOLUME names must be known by the system. If the file is on an unlabeled tape, only VOLUME and POSITION are required. This information can be made available by an operator response to a system prompt, through a procedure, or through the VALUE OF clause. Additionally, the VALUE OF clause can optionally be used to specifiy the packing density of data and index blocks. This packing option, meaningful for indexed files only, is available in output mode only and is implemented through the DATA AREA and INDEX AREA clauses.

When using DMS/TX files, VALUE OF clauses retrieve DMS/TX file information for existing files and allocate Recovery Blocks for program created files while the files are in the OPEN OUTPUT mode. The DMS/TX clauses are RECOVERY-BLOCKS, RECOVERY-STATUS, and DATABASE-NAME.

General Format

```
             ⎧⎪ ⎡ FILENAME IS          ⎧data-name-1⎫ ⎤ ⎪⎫
             ⎪⎪ ⎣                      ⎩literal-1  ⎭ ⎦ ⎪⎪
             ⎪⎪ ⎡ LIBRARY IS           ⎧data-name-2⎫ ⎤ ⎪⎪
             ⎪⎪ ⎣                      ⎩literal-2  ⎭ ⎦ ⎪⎪
             ⎪⎪ ⎡ VOLUME IS            ⎧data-name-3⎫ ⎤ ⎪⎪
             ⎪⎪ ⎣                      ⎩literal-3  ⎭ ⎦ ⎪⎪
             ⎪⎪  [SPACE IS              data-name-4]   ⎪⎪
    VALUE OF ⎨⎪  [POSITION IS           data-name-5]   ⎪⎬
             ⎪⎪  [INDEX AREA IS         data-name-6]   ⎪⎪
             ⎪⎪  [DATA AREA IS          data-name-7]   ⎪⎪
             ⎪⎪ ⎡ RECOVERY-BLOCKS IS   ⎧data-name-8⎫ ⎤ ⎪⎪
             ⎪⎪ ⎣                      ⎩literal-4  ⎭ ⎦ ⎪⎪
             ⎪⎪  [RECOVERY-STATUS IS    data-name-9]   ⎪⎪
             ⎩⎪  [DATABASE-NAME IS      data-name-10]  ⎪⎭
```

Syntax Rules

1. Data-name-1 through data-name-10 may be qualified.

2. Data-name-1 through data-name-10 cannot be subscripted or indexed, nor can they be items described with the USAGE IS INDEX clause.

3. Data-name-1 through data-name-10 must be defined in the Working-Storage Section.

4. The data items referenced by data-name-1, data-name-2, and data-name-3 must be alphanumeric. The data items referenced by data-name-4, data-name-5, data-name-6, and data-name-7 must be numeric. Data-name-8 and data-name-9 must be either alpha or alphanumeric with the declared length equal to one character. Data-name-10 must be either alpha or numeric with a declared length of not more than six characters long.

5. Literal-1, literal-2, and literal-3 must be nonnumeric literals. Literal-4 must be a single character literal.

General Rules

1. For any open mode, after the OPEN is issued, the file name, library name, and volume name are available in the data items referenced by data-name-1, data-name-2, and data-name-3, respectively. In the input mode, the the number of records in the file is available at OPEN time in the SPACE item, referenced by data-name-4.

2. For an output file (output or extend mode), VALUE OF parameters supply default information used to complete the file label.

3. For an input file (input, I-O, or shared mode), the file name is used to select the runtime file. Data-name-4 is set to the number of records in the file and can be used in allocating space for a new file. Data-name-2 is used to specify the name of the library in which the file is found. Data-name-3 is used to specify the name of the volume on which the library is found.

4. The INDEX AREA and DATA AREA phrases are meaningful for indexed files only. The values associated with data-name-6 and data-name-7 must be numeric and between the values 1-100. The value specifies the percentage packing density for data and index blocks that will occur for files being created in output mode.

   If these options are not coded, a default packing density of 100 percent is assumed.

5. The POSITION phrase is valid for tapes only. Data-name-5 specifies the ordinal postion number of a file on tape.

6. Literal-1, literal-2, literal-3, and the data items referenced by data-name-1, data-name-2, and data-name-3 can exceed eight characters. However, for literal-1, literal-2, data-name-1 and data-name-2, only the first eight characters are used; for literal-3 and data-name-3, only the first six characters are used. The value of data-name-5 must be between 1 and 65,545. The value of data-name-4 must be between 1 and 16,777,215.

7. If any one or a combination of RECOVERY-BLOCKS, RECOVERY-STATUS, or DATABASE-NAME appears in the VALUE OF clause, a larger UFB (with the DMS/TX extension block) is allocated for the associated file. With the larger UFB area, the Recovery Blocks are allocatable for the related file and the special DMS/TX information is retrievable.

8. When creating a new file (OPEN OUTPUT mode), the Recovery Blocks are allocated by specifying the literal-4 as "A" or storing "A" as the content of data-name-8 before the OPEN statement.

9. Any value other than "A" for literal-4 or any other value stored in data-name-8 will be ignored when the OPEN statement is executed. The value "A" is also ignored if the OPEN statement is not in the OUTPUT mode.

10. After the OPEN statement is executed, the requested information (RECOVERY-BLOCKS, RECOVERY-STATUS, DATABASE-NAME) is stored in data-name-8, data-name-9, and data-name-10 respectively. You can retrieve this information once the file is opened.

11. The possible values of the RECOVERY-BLOCKS and their meanings are as follows:

N	No Recovery Blocks
A	Recovery Blocks allocated (but file is unattached)
U	Recovery Blocks allocated and file is part of a DMS/TX database.

12. The possible values of the RECOVERY-STATUS and their meanings are as follows:

N	No Recovery
S	Softcrash Recovery
F	Full Recovery

13. The DATABASE-NAME clause returns the database name which the DMS/TX file is attached to as the value of data-name-10.

Example of VALUE OF Clause

```
DATA DIVISION.

FILE SECTION.

FD  ACCOUNTS-DATA

    BLOCK CONTAINS 25 RECORDS
    RECORD CONTAINS 80 COMPRESSED CHARACTERS
    LABEL RECORDS ARE STANDARD
    VALUE OF FILENAME IS "ACCTOUT"
             LIBRARY  IS "ACCTDATA"
             VOLUME   IS "VOL001"
             SPACE    IS RECNUM
             RECOVERY-BLOCKS IS "A"
             RECOVERY-STATUS IS RVS-1
             DATABASE-NAME IS DBN-1
    DATA RECORD       IS TRANSACTION-FILE.

01  TRANSACTION-FILE PIC X(80).
 .
 .
 .
WORKING-STORAGE SECTION.

77  RECNUM PIC 9(7) VALUE 100.
01  A               PIC S9999 COMP VALUE 0.
01  RVS-1           PIC X(1).
01  DBN-1           PIC X(6).
```

Example of Program-Provided File Parameters

```
ENVIRONMENT DIVISION.

FILE-CONTROL.

    SELECT CLIENT
        ASSIGN "CLIENTFI", "DISK", NODISPLAY.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD CLIENT
    VALUE OF FILENAME IS "CLIFILE"
             LIBRARY  IS "CLIDATA"
             VOLUME   IS "VOL444"

    LABEL RECORDS ARE STANDARD.
        .
        .
        .
    PROCEDURE DIVISION.
        OPEN INPUT CLIENT
              .
              .
              .
```

In this example, CLIFILE is the name of a data file needed by the program to obtain data. Normally, an OPEN statement would result in an operator prompt for the names of the data file, library, and volume. However, this example provides the necessary information with a VALUE OF clause. Note that the Environment Division ASSIGN clause utilizes the NODISPLAY option, thus suppressing the system from prompting the operator as long as the file information is made available. If the information is not known by the system at runtime, the NODISPLAY option will not prevent the system prompt.

NOTE

For a complete discussion of DMS/TX refer to Chapter 3.

The Sort-Merge File Description Entry

Function

The sort-merge file description furnishes information concerning the physical structure, identification, and record names for the file to be sorted or merged.

General Format

```
SD file-name

 [ BLOCK CONTAINS integer-1 { RECORDS    } ]
 [                          { CHARACTERS } ]

 [RECORD CONTAINS [integer-2 TO] integer-3 [COMPRESSED] CHARACTERS]

  [ VALUE OF                                              ]
  [            [ VOLUME IS      { data-name-1 } ]         ]
  [            [                { literal-1   } ]         ]
  [                                                       ]
  [            [ SPACE IS         data-name-2 ]           ]

  [ DATA       { RECORD IS    }  data-name-4 [ data-name-5] ... ] .
  [            { RECORDS ARE  }                                 ]
```

Syntax Rules

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

2. The clauses that follow the name of the file are optional and their order of appearance is immaterial.

3. One or more record description entries must follow the sort-merge file description entry. However, no input-output statements may be executed for this file.

4. Data-name-1, -2, -3, and -4, may be qualified.

General Rules

1. If the SORT or MERGE statement has USING and GIVING clauses, the file described by the sort-merge description entry is used to provide key information for the SORT utility.

2. If the SORT or MERGE statement has an INPUT or OUTPUT procedure the file described by the SORT-MERGE Description Entry is created and used as an intermediate file. It must be large enough to contain all sorted or merged records.

3. Refer to the VALUE OF clause description for rules governing VOLUME IS and SPACE IS.

11.3.2 Working-Storage Section

The Working-Storage Section is composed of the section header, followed by data description entries for noncontiguous data items and/or record description entries. Each Working-Storage Section record name and noncontiguous item name must be unique.

Noncontiguous Working-Storage

Items and constants in working-storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry that begins with the special level number 77. There is no restriction placed on the order of 01 and 77 level items.

The following data clauses are required in each data description entry.

1. Level-number
2. Data-name
3. The PICTURE, USAGE IS INDEX, or USAGE IS BINARY clause.

Other data description clauses are optional and can be used to complete the description of the item, if necessary.

Working-Storage Records

Data elements and constants in working-storage that bear a definite hierarchical relationship to one another must be grouped into records according to the rules for formation of record descriptions. All clauses that are used in record descriptions in the File Section can also be used in record descriptions in the Working-Storage Section. In addition, the VALUE clause can be used in the Working-Storage Section.

Initial Values

The initial value of any item in the Working-Storage Section, except an index data item, is specified by using the VALUE clause with the data item. The initial value of any index data item is unpredictable, and should therefore be set to a known value.

11.3.3 Data Description Entry

Function

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

General Format

Format 1

```
level-number { data-name-1 } [REDEFINES data-name-2]
             { FILLER      }

                                               { BINARY        }
                                               { COMPUTATIONAL }
[{ PICTURE } IS character-string]   [USAGE IS] { COMP          }
[{ PIC     }                    ]   [         ]{ DISPLAY-WS    }
                                               { DISPLAY       }
                                               { INDEX         }

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

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

[{ JUSTIFIED }  RIGHT]
[{ JUST      }       ]

[BLANK WHEN ZERO]

[ VALUE IS { literal                 } ]
[          { user-figurative-constant } ]

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

Format 2

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

Syntax Rules

1. For Format 1 the level number can be any number from 01 - 49 or 77. For Format 2, the level number must be 88.

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

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

4. The words THRU and THROUGH are equivalent.

General Rules

Format 1

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

Format 2

2. Format 2 is used for each condition name and requires a separate entry with level number 88. The format 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 that contains a level number except

   a. Another condition name

   b. An index data item.

BLANK WHEN ZERO Clause

Function

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

General Format

```
[BLANK WHEN ZERO]
```

Syntax Rules

1. The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited.

General Rules

1. When the BLANK WHEN ZERO clause is used, the item contains nothing but spaces if the value of the item is zero.

2. 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.

DATA-NAME or FILLER Clause

Function

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 explicitly.

General Format

$$\left\{\begin{array}{l}\text{data-name-1}\\ \underline{\text{FILLER}}\end{array}\right\}$$

Syntax Rules

1. In the File, Working-Storage and Linkage Sections, a data name or the key word FILLER must be the first word following the level number in each data description entry.

General Rules

1. The key word FILLER can be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly.

JUSTIFIED Clause

Function

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format

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

Syntax Rules

1. The JUSTIFIED clause can be specified only at the elementary item level.

2. JUST is an abbreviation for JUSTIFIED.

3. The JUSTIFIED clause cannot be specified for any data item described as numeric or for which editing is specified.

General Rules

1. When a receiving data item is described with the JUSTIFIED clause and is smaller than the sending data item, the leftmost characters are truncated. When the receiving data item is described with the JUSTIFIED clause and is larger than the sending data item, the data is aligned at the rightmost character position in the data item with spaces filling the leftmost character positions.

2. When the JUSTIFIED clause is omitted, the standard rules for aligning data within an elementary item apply. (Refer to Section 11.1.6, Standard Alignment Rules.)

LEVEL-NUMBER Clause

Function

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

General Format

```
level-number
```

Syntax Rules

1. A level number is required as the first element in each data description entry.

2. Data description entries subordinate to an FD entry must have level numbers with the values 01 - 49, or 88.

3. Data description entries in the Working-Storage Section and Linkage Section must have level numbers with the values 01 - 49, 77 or 88.

General Rules

1. The level number 01 identifies the first entry in each record description.

2. Special level numbers have been assigned to certain entries where there is no real concept of level.

   a. Level number 77 is assigned to identify noncontiguous working-storage or linkage data items.

   b. Level number 88 is assigned to entries that define condition names associated with a conditional variable and can be used only as described in Format 2 of the data description entry.

3. Multiple level 01 entries within the same FD paragraph represent implicit redefinitions of the same area, except that the order in which they are listed is not significant.

OCCURS Clause

Function

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indices.

General Format

$$
\begin{array}{l}
\underline{\text{OCCURS}}\ \text{integer TIMES} \\
\quad \left[\left\{ \begin{array}{l} \underline{\text{ASCENDING}} \\ \underline{\text{DESCENDING}} \end{array} \right\} \text{KEY IS data-name-1 [data-name-2] ...} \right] \text{...} \\
\quad \left[\underline{\text{INDEXED}}\ \text{BY index-name-1 [index-name-2] ...} \right]
\end{array}
$$

Syntax Rules

1. Data-name-1, data-name-2, ... may be qualified.

2. Data-name-1 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

3. Data-name-2, ..., must be the name of an entry subordinate to the group item which is the subject of this entry.

4. 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 hierachy.

5. The OCCURS clause cannot be specified in a data description entry that has an 01, 77 or an 88 level number.

6. If data-name-1 is not the subject of this entry, the following rules apply:

   a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of this entry.

   b. Items identified by the data-name in the KEY IS phrase must not contain an OCCURS clause.

c. There must not be any entry that contains an OCCURS clause between the items identified by the data-names in the KEY IS phrase and the subject of this entry.

7. Index-name-1, index-name-2, ..., must be unique words within the program.

General Rules

1. 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 that is the subject of this clause must be either subscripted or indexed whenever it is referred to in statements other than the SEARCH or USE FOR DEBUGGING statements. Further, if the subject of this clause 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.

2. 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.

3. The value of integer represents the exact number of occurences of the subject entry.

4. The KEY IS phrase is used with the OCCURS clause in conjunction with Format 2 of the SEARCH statement to perform a binary search on an ordered table, providing an efficient method of finding an element in a large table. The values contained in data-name-1, data-name-2, etc., specify parts of table records that are arranged in ascending or descending order. The ascending or descending order is determined according to the rules for comparison of operands. Refer to Section 12.4.1, <u>Comparison of Numeric Operands</u> and <u>Comparison of Nonnumeric Operands</u>. The data names are listed in their descending order of significance.

PICTURE Clause

Function

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

General Format

$$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}\ \text{IS character-string}$$

Syntax Rules

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

2. 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.

3. At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or '$' must be present in a PICTURE character-string.

4. The maximum number of characters allowed in the character-string is 30.

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

6. PIC is an abbreviation for PICTURE.

7. The asterisk, when used as the zero suppression symbol, and the BLANK WHEN ZERO clause cannot appear in the same entry.

General Rules

1. There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

2. To define an item as alphabetic:

   a. Its PICTURE character-string can only contain the symbols 'A', and 'B'.

   b. Its contents, when represented in standard data format (refer to Section 11.1), must be any combination of the 26 letters of the English alphabet and the space from the COBOL character set.

3. To define an item as numeric:

   a. 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. The PICTURE clause is omitted for numeric items when USAGE IS BINARY is coded. Refer to "USAGE Clause" later in this subsection.

   b. 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', and '9'; if signed, the item can also contain a '+', '-', or other representation of an operational sign. (Refer to "SIGN Clause", later in this subsection.)

4. To define an item as alphanumeric:

   a. Its PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', and '9', and the item is treated as if the character-string contained all 'X's. A PICTURE character-string that contains all 'A's or all '9's does not define an alphanumeric item.

   b. Its contents when represented in standard data format are allowable characters in the COBOL character set.

5. To define an item as alphanumeric edited:

   a. Its PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and '/'; and must contain one of the following combinations:

      1) At least one 'B' and at least one 'X'

      2) At least one '0' (zero) and at least one 'X'

      3) At least one '/' (stroke) and at least one 'X'

      4) At least one '0' (zero) and at least one 'A'

      5) At least one '/' (stroke) and at least one 'A'.

   b. The contents when represented in standard data format are allowable characters in the COBOL character set.

6. To define an item as numeric edited:

   a. Its PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', '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; the following rules also apply.

      1) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive.

      2) The character-string must contain at least one '0', 'B', '/', 'Z', '*', '+', ',', '.', '-', 'CR', 'DB', or the currency symbol.

   b. 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.

7. 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 that is enclosed in parentheses following the symbols 'A', ',', 'X', '9', 'P', 'Z', '*', 'B', '/', '0', '+', '-', or the currency symbol indicates the number of consecutive occurrences of the symbol. Note that the following symbols can appear only once in a given PICTURE: 'S', 'V', '.', 'CR', and 'DB'.

8. The functions of the symbols used to describe an elementary item are explained as follows:

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

   b. Each 'B' in the character-string represents a character position into which the space character is to be inserted.

   c. 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 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 the 'P's if 'P's are the leftmost PICTURE characters, and to the right if 'P's are the 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.

d. 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. The 'S' must be written as the leftmost character in the PICTURE. The 'S' is counted in determining the size of the elementary item, in terms of standard data format characters, unless the entry is subject to a SIGN clause from which the optional SEPARATE CHARACTER phrase is omitted. (Refer to "SIGN Clause", later in this subsection.)

e. The 'V' is used in a character-string to indicate the location of the assumed decimal point and can 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.

f. Each 'X' in the character-string is used to represent a character position that contains any allowable character from the COBOL character set.

g. Each 'Z' in a character-string can only be used to represent the leftmost leading numeric character positions that are replaced by a space character when the content of that character position is zero. Each 'Z' is counted in the size of the item.

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

i. Each '0' (zero) in the character-string represents a character position into which the numeral zero is to be inserted. The '0' is counted in the size of the item.

j. Each '/' (stroke) in the character-string represents a character position into which the stroke character is to be inserted. The '/' is counted in the size of the item.

k. Each ',' (comma) in the character-string represents a character position into which the character ',' is to 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.

l. When the character '.' (period) appears in the character-string it is an editing symbol that represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' is to 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 these characters appear in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

m. The symbols '+', '-', 'CR', and 'DB' are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol is to 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.

n. Each '*' (asterisk) in the character-string represents a leading numeric character position into which an asterisk is placed when the contents of that position is zero. Each '*' is counted in the size of the item.

o. 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 the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

Editing Rules

1. Editing in the PICTURE clause is either by insertion or by suppression and replacement. There are four types of insertion editing available:

   a. Simple insertion
   b. Special insertion
   c. Fixed insertion
   d. Floating insertion.

   There are two types of suppression and replacement editing:

   a. Zero suppression and replacement with spaces
   b. Zero suppression and replacement with asterisks.

2. The type of editing that can be performed upon an item is dependent upon the category to which the item belongs. The following table specifies the type of editing that can 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 '/'
Numeric Edited	All, subject to Editing Rule 3

3. Floating insertion editing and editing by zero suppression and replacement are mutually exclusive in a PICTURE clause. Only one type of replacement can be used with zero suppression in a PICTURE clause.

4. 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 is to be inserted.

5. 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.

6. Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and one editing sign control symbol can be used in a given PICTURE character-string. The symbols 'CR' or 'DB' represent two character positions in determining the size of the item; these symbols must represent the rightmost 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 symbols '+' or '-', when used, must be either the leftmost or the 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	Result Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

7. 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 can contain any of the fixed insertion symbols or have fixed insertion characters immediately to its right. 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 item.

   The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data can 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, a single floating insertion character is 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. 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 contains 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 nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. Zero Suppression Editing. The suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character 'Z' or the symbol '*' (asterisk). 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 is the space; if the asterisk is used, the replacement character is '*'.

   Zero suppression and replacement are indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions that are to be replaced when the associated character positions in the data contain zeros. Any of the simple insertion characters embedded in the string of symbols, or to the immediate right of this string, are part of the string.

   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 that 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 is spaces. If the value is zero and the suppression symbol is '*', the data item is all '*' except for the actual decimal point.

9. The symbols '+', '-', '*', 'Z', and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence Rules

The following 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 can 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'.

Nonfloating insertion symbols '+' and '-', floating insertion symbols 'Z', '*', '+', '-', and 'CS', and the 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.

Second Symbol	First Symbol	Nonfloating Insertion Symbols									Floating Insertion Symbols						Other Symbols					
		B	0	/	,	.	{+} {-}	{+} {-}	{CR} {DB}	CS	{Z} {*}	{Z} {*}	{+} {-}	{+} {-}	CS	CS	9	A	S	V	P	P
*	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	x
	{CR} {DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x
	CS						x															
**	{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
***	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

* Nonfloating Insertion Symbols
** Floating Insertion Symbols
*** Other Symbols

Figure 11-1. PICTURE Character Precedence Chart

REDEFINES Clause

Function

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

General Format

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

NOTE

Level-number and data-name-1 are shown in the above format to improve clarity. Level-number and data-name-1 are not part of the REDEFINES clause.

Syntax Rules

1. The REDEFINES clause, when specified, must immediately follow data-name-1.

2. The level numbers of data-name-1 and data-name-2 must be identical, but must not be 88.

3. This clause must not be used in level 01 entries in the File Section. (Refer to General Rule 2 of "DATA RECORDS Clause" in Section 11.3.1.)

4. The data description entry for data-name-2 cannot contain a REDEFINES clause. However, data-name-2 can be subordinate to an entry that contains a REDEFINES clause. The data description entry for data-name-2 cannot contain an OCCURS clause. However, data-name-2 can 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 cannot be subscripted or indexed.

5. No entry having a level-number numerically lower than the level-number of data-name-2 and data-name-1 can occur between the data description entries of data-name-2 and data-name-1.

General Rules

1. Redefinition starts at data-name-2 and ends when a 77 level or a level number less than or equal to that of data-name-2 is encountered.

2. When the level number of data-name-1 is other than 01, it must not specify a greater number of character positions than 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.

3. 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.

4. The entries giving the new description of the character positions must not contain any VALUE clauses, except in condition-name entries.

5. Multiple level 01 entries subordinate to any given level indicator represent implicit redefinitions of the same area. However, 01 entries subordinate to an FD may appear in any order, regardless of the sizes of the individual entries.

SIGN Clause

Function

The SIGN clause specifies the position and mode of representation of the operational sign when it is necessary to describe these properties explicitly.

General Format

$$[\underline{\text{SIGN}}\ \text{IS}]\ \left\{\begin{array}{l}\underline{\text{LEADING}}\\ \underline{\text{TRAILING}}\end{array}\right\}\ [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Syntax Rules

1. The SIGN clause can be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

2. The numeric data description entries to which the SIGN clause applies must be described as usage DISPLAY.

3. At most, one SIGN clause can apply to any given numeric data description entry.

General Rules

1. The optional SIGN clause, if present, specifies the position and 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 'S' indicates the presence of, but neither the representation nor necessarily the position of, the operational sign.

2. A numeric data description entry whose PICTURE contains the character 'S', but to which no SIGN clause applies, has an operational sign. Neither the representation nor, necessarily, the position of the operational sign is specified by the character 'S'. In this (default) case, the compiler uses a TRAILING SEPARATE CHARACTER sign. General rules do not apply to such signed numeric data items.

3. If the optional SEPARATE CHARACTER phrase is not present, then:

   a. The operational sign is presumed to be associated with the leading (or, respectively, trailing) digit position of the elementary numeric data item.

   b. The letter 'S' in a PICTURE character-string is not counted in determining the size of the item in terms of standard data format characters (refer to Section 11.1, Computer Independent Data Description).

4. If the optional SEPARATE CHARACTER phrase is present, then:

   a. The letter 'S' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

   b. The operational sign will be presumed to be the leading (or, respectively, trailing) character position of the elementary numeric data item; this character position is not a digit position.

   c. The operational signs for positive and negative are the standard data format characters '+' and '-', respectively.

5. Every numeric data description entry whose PICTURE contains the character 'S' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

SYNCHRONIZED Clause

Function

This clause is treated as a comment by the compiler.

General Format

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

Syntax Rules

1. This clause can only appear with an elementary item.

2. SYNC is an abbreviation for SYNCHRONIZED.

USAGE Clause

Function

The USAGE clause specifies the format of a data item in the computer storage.

General Format

```
              ⎧BINARY        ⎫
              ⎪COMPUTATIONAL ⎪
[USAGE IS]    ⎨COMP          ⎬
              ⎪DISPLAY-WS    ⎪
              ⎪DISPLAY       ⎪
              ⎩INDEX         ⎭
```

Syntax Rules

1. The PICTURE character-string of a COMPUTATIONAL item can contain only '9's, the operational sign character 'S', the implied decimal point character 'V', and one or more 'P's. (Refer to "PICTURE Clause", earlier in this subsection.)

2. COMP is an abbreviation for COMPUTATIONAL.

3. An INDEX data item can be referenced explicitly only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

4. The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules

1. Except for USAGE IS DISPLAY-WS, the USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

2. 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 can restrict the USAGE clause of the operands referred to. The USAGE clause can affect the radix or type of character representation of the item.

3. A COMPUTATIONAL item is capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (cannot be used in computations).

4. The USAGE IS DISPLAY clause indicates that the format of the data is ASCII.

5. 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.

6. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that must correspond to an occurrence number of a table element. The elementary item cannot be a conditional variable. The INDEX item is stored as if usage were BINARY. If a group item is described with the USAGE IS INDEX clause the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in the SET statement or in a relation condition.

7. An INDEX data item can be part of a group that is referred to in a MOVE or input/output statement, in which case no conversion takes place.

8. The external and internal format of an INDEX data item and a BINARY data item is specified as a half word (2-byte) BINARY item.

9. A BINARY item is capable of representing a value to be used in computations and is automatically defined by the compiler as two bytes. The permissible range for a BINARY item is +32,767 to -32,768. If a group item is described as BINARY, the elementary items in the group are BINARY. The group item itself is not BINARY (cannot be used in computations).

USAGE IS DISPLAY-WS

10. The USAGE IS DISPLAY-WS clause identifies display records for which automatic screen formatting is to occur. This is further explained in Section 4.3.2.

11. The USAGE IS DISPLAY-WS clause can only be applied to entries in the Working-Storage Section.

12. The level number must be an 01 entry.

13. A display picture order area (for screen control) consisting of four bytes is generated by the compiler. (Refer to "Order Area" in Section 4.3.4 and to Appendix D, Workstation Screen Order Area.) The order area can be fetched or modified via the ORDER-AREA OF record-name clause.

14. The automatic mapping operation does not occur until execution time when the DISPLAY AND READ command is invoked, therefore, data items in the display record cannot be directly accessed except via the OBJECT and SOURCE clauses. However, the Field Attribute Character for any data item can be accessed in a MOVE statement by referring to FAC OF display-item.

VALUE Clause

Function

The VALUE clause defines the value of constants, the initial value of working-storage items and the values associated with a condition name.

General Format

Format 1

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

Format 2

$$\left\{ \begin{array}{l} \underline{\text{VALUE}}\text{ IS} \\ \underline{\text{VALUES}}\text{ ARE} \end{array} \right\} \left\{ \text{literal-1} \left[\left\{ \begin{array}{l} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{literal-2} \right] \right\} \cdots$$

Syntax Rules

1. A signed numeric literal must have a signed numeric PICTURE character-string associated with it.

2. All numeric literals in a VALUE clause must have a value that is within the range of values indicated by the PICTURE clause, and must not have a value that would require truncation of nonzero digits. Nonnumeric literals must not exceed the size indicated by the PICTURE clause.

3. The words THROUGH and THRU are equivalent.

4. The VALUE clause cannot be used to describe items whose usage is INDEX.

General Rules

1. 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.

   a. 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. (Refer to Section 11.1.6, Standard Alignment Rules.)

   b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals in the VALUE clause must be nonnumeric literals. (There is an exception to this rule for USAGE IS DISPLAY-WS records; refer to "SOURCE or VALUE Clause" in Section 11.3.4.) 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 (refer to "PICTURE Clause", earlier in this subsection) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.

   c. Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that can be specified.

2. A figurative constant can be substituted for a literal.

3. 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.

4. Format 2 can be used only in conjunction with condition names (level 88). Whenever the THRU phrase is used, literal-1 must be less than literal-2, literal-3 less than literal-4, and so on.

5. Rules governing the use of the VALUE clause differ with the respective sections of the Data Division.

   a. In the File Section and the Linkage Section, the VALUE clause can be used only in condition-name entries.

   b. In the Working-Storage Section, the VALUE clause can be used to specify the initial value of any data item; in that 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. The VALUE clause must be used in condition-name entries.

6. 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 an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

7. The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

8. If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, 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.

9. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

10. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value that must correspond to an occurrence number of a table item. The VALUE clause cannot be specified for index data items.

11.3.4 Workstation Screen Description Entry

Function

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

General Format

```
01 record-name [USAGE IS] DISPLAY-WS
    level-number {data-name-1}
                 {FILLER     }
        [COLUMN NUMBER IS integer-1] [{ROW } NUMBER IS integer-2]
                                     [{LINE}                    ]
      [{PIC    } IS character-string]
      [{PICTURE}                    ]
        [          {NEGATIVE                                                 }]
        [RANGE IS  {FROM      {data-name-2} TO {data-name-3}                 }]
        [          {          {literal-1  }    {literal-2  }                 }]
        [          {POSITIVE                                                 }]
        [          {table-name                                               }]
        [OBJECT IS data-name-4] [VALUE IS literal-3      ]
                                [SOURCE IS data-name-5   ]
        [OCCURS integer-3 TIMES].
```

Syntax Rules

1. The record-name USAGE IS DISPLAY-WS clause must be the first clause in the record description entry.

2. The data-name-1 or FILLER clauses must precede all other clauses within their data name description entry.

3. Clauses that follow the data-name-1 or FILLER clause within the data description entry (e.g., COLUMN, ROW, PIC, VALUE) can be in any order.

4. The ROW clause must be specified for a group item.

5. The ROW clause and the OCCURS clause are the only clauses that can be specified for a group item. All clauses are legal for an elementary item.

6. The VALUE clause cannot be specified in display items in which the OCCURS clause is specified.

7. Elementary items for which the OCCURS clause is specified must also specify both ROW and COLUMN clauses.

8. Data names withing DISPLAY-WS cannot be qualified.

General Rules

1. Display items can be group items or elementary items. A group item refers to a row or a group of rows; its level number can be any number from 02 to 49. An elementary item refers to a field within a row; its level number can be any number from 02 to 49.

2. If group items are not used, elementary items can refer to fields of any specified row. (Refer to "ROW CLAUSE", later in this subsection.)

3. All elementary items specified within the same group item are associated with the same row. These elementary items must not overflow the row.

4. There is no restriction on the number of display record definition entries included in the Working-Storage Section.

5. Areas of the display screen that are not specified will display as blanks.

6. Blanks in modifiable fields and uninitialized modifiable fields display as pseudo-blanks (that is, half-solid characters). A modifiable field is any field for which the OBJECT clause is specified.

COLUMN Clause

Function

The COLUMN clause is used to define beginning column position of the field that is to be displayed.

A nondisplaying control character (generated by the compiler), referred to as a Field Attribute Character (FAC), occupies the column preceding the column specified by the COLUMN clause. The FAC controls the display attributes of the field. Default FACs are functions of the OBJECT clause.

General Format

```
[COLUMN NUMBER IS integer-1]
```

General Rules

1. Integer-1 can be any number in the range 2 through 80. (If 1 is specified, no field attribute character is generated.) The attributes of the field are: low intensity, protected, alphanumeric.

2. The COLUMN position of the end of the field to be displayed is one less than the sum of integer-1 and the number of characters specified for the field in the PICTURE clause.

3. If the ending column exceeds 80:

   a. The field continues with Column 1 of the next line.

   b. The portion of the line that overlaps to the next line is controlled by a nominal FAC that occupies Column 0 of each line (i.e., ahead of each line's first displayable column). Its display attributes are: low intensity, protected, alphanumeric.

4. At least one space must separate one field from another. If the COLUMN clause is not specified, the field's beginning column is automatically generated such that one space separates the field from the ending column of the preceding field. If there is no preceding field and the COLUMN clause is not specified, the field begins at Column 2 of Line 1.

5. The COLUMN clause must be specified for an elementary display item that is to OCCUR.

6. A COLUMN clause specified for a display item that OCCURS across a row (i.e., the OCCURS clause applies to a field in a row) indicates the starting column position of the first field in the series. The starting column of each subsequent field is two more than the ending column of the previous field. Thus, each field is separated from the next field by one blank column.

7. A COLUMN clause specified for a display item that OCCURS from row to row (i.e., the OCCURS clause applies to a row or group of rows containing the elementary display item) indicates the starting column position of the specified field in the first row, and in each subsequent row to which the OCCURS clause applies.

ROW Clause

Function

The ROW (or LINE) clause is used to specify the row (or line) in which display of data-name-1 is to begin.

General Format

$\left\{\begin{array}{l}\underline{\text{ROW}}\\ \underline{\text{LINE}}\end{array}\right\}$ NUMBER IS integer-1

General Rules

1. Integer-1 can be any number in the range 1 through 24.

2. If the ROW clause is omitted, the previous line is assumed (if no previous line was specified, Line 1 is assumed).

3. The ROW clause must be specified for all group display items.

4. The ROW clause must be specified for elementary display items that are to OCCUR.

PICTURE Clause

Refer to "PICTURE CLAUSE" in Subsection 11.3.3, Data Description Entry.

RANGE Clause

Function

The RANGE clause is used to specify the legal limits of user input. The limits specified with the RANGE clause are used by the DISPLAY AND READ statement to determine legal inputs to modifiable fields.

General Format

```
            ⎧NEGATIVE                                          ⎫
            ⎪                                                  ⎪
RANGE IS    ⎨FROM         {data-name-1}  TO  {data-name-2}     ⎬
            ⎪             {literal-1  }      {literal-2  }     ⎪
            ⎪POSITIVE                                          ⎪
            ⎩table-name                                        ⎭
```

General Rules

1. Table-name must reference a numeric or alphanumeric data item whose description in the Data Division contains an OCCURS clause.

2. Data-name-1 and data-name-2 must be elementary data items.

3. NEGATIVE is any value less than zero.

4. POSITIVE is any value greater than zero.

5. Literal-1 or the contents of data-name-1 must be less than or equal to literal-2 or the contents of data-name-2. The ASCII collating sequence is used to determine which alphanumeric character precedes another.

6. For a DISPLAY-WS item whose picture is alphanumeric or numeric edited, data is validated as follows:

   a. If the RANGE items are numeric, the data entered on the screen is convereted to numeric before being validated.

   b. If the RANGE items are alphanumeric, the validation is made according to the ASCII collating sequence.

7. The picture for table-name must be of the same length and type as that specified for the display data item.

8. Data-name-1 and data-name-2 may be qualified.

SOURCE or VALUE Clause

Function

The SOURCE clause specifies the data name from which data is obtained for initial display. The VALUE clause defines the value of constants and the initial contents of screen fields.

General Format

$$\left\{\begin{array}{l}\underline{\text{VALUE}}\ \text{IS literal-1}\\ \underline{\text{SOURCE}}\ \text{IS data-name-1}\end{array}\right\}$$

General Rules

1. SOURCE and VALUE clauses are not permitted for the same item.

2. The transfer of data from data-name-1 occurs at execution time in accordance with the rules of the MOVE statement.

3. Data-name-1 cannot be subscripted, but the data description entry for data-name-1 can contain an OCCURS clause. If the SOURCE clause and an OCCURS clause apply to the same display item, data-name-1 must:

   a. Have the same dimensions as the corresponding data name in the OCCURS clause.

   b. Have a compatible PICTURE clause.

4. The SOURCE clause cannot be specified for a group display item.

5. Data-name-1 cannot be the name of a DISPLAY-WS record or field.

6. Aside from the exception noted in General Rule 7, the rules governing the VALUE clause for workstation screen description entries are the same as those given in Subsection 11.3.3 for the VALUE clause in data description entries.

7. In a DISPLAY-WS record only, the VALUE clause can assign a numeric literal to an item with a nonnumeric picture.

8. Data-name-1 may be qualified.

OBJECT Clause

Function

The OBJECT clause specifies the data item to which data is to be moved after validation (refer to "DISPLAY AND READ Statement" in Section 12.5). This clause also determines display attributes for fields.

General Format

```
[OBJECT IS data-name-1]
```

General Rules

1. The transfer of data to data-name-1 occurs in accordance with the rules of the MOVE statement.

   a. If data-name-1 is numeric, the move is MOVE WITH CONVERSION with the following exceptions.

      1) The value in the object field is not aligned according to an implied decimal point or scaling in the source field. If the operator enters an actual decimal point from the workstation, the data in the object field is aligned according to it. Otherwise, the decimal point is assumed to be positioned to the right of the last digit.

      2) In moves from numeric edited screen fields, currency signs, CR, DB, asterisks, slashes, spaces (represented by the 'B' character symbol), and commas (or periods if DECIMAL-POINT IS comma) are removed.

   b. If data-name-1 is alphanumeric the move is MOVE.

2. Display attributes for fields are determined as follows.

   a. If the OBJECT clause is not specified, the FAC is generated as low intensity, protected, alphanumeric.

   b. If the OBJECT clause is specified, the FAC is a function of the PICTURE clause of the elementary item. For numeric items, FAC is high intensity, modifiable, numeric only. For alphanumeric items, the FAC is high intensity, modifiable, alphanumeric,upper case.

3. Data-name-1 cannot be subscripted, but the data description entry for data-name-1 can contain an OCCURS clause. If the OBJECT clause and an OCCURS clause apply to the same display item, data-name-1 must:

   a. Have the same dimensions as the corresponding data item in the OCCURS clause.

   b. Have a compatible PICTURE clause.

4. The OBJECT clause cannot be specified for a group display item.

5. Data-name-1 cannot be the name of a DISPLAY-WS record or field.

6. Data-name-1 may be qualified.

OCCURS Clause

Function

The OCCURS clause defines a table of display items. This clause can be used to repeat a field, a row, or a group of rows in the display.

General Format

```
[OCCURS integer-1 TIMES]
```

General Rules

1. The OCCURS clause is legal for both group and elementary display items. When specified for a group item, this clause indicates that a row of field(s) or a group of rows is to be repeated. When specified for an elementary item, it indicates that a field is to be repeated within a row.

2. An elementary item that OCCURS within a row cannot extend beyond Column 80 on any repetition.

3. A group item that OCCURS cannot repeat a row or group of rows enough times to cause the display picture to extend past Row 24.

4. When an elementary item OCCURS, the starting column position of the first field in the row must be specified by the COLUMN clause. The starting position of each subsequent field is two more than the ending position of the previous field. Thus, each field is separated from the next by one blank column.

5. When a group item OCCURS, placement of fields within the row (or group of rows) is determined by the COLUMN clauses of the elementary display items. The fields in the repeated row or group of rows are started in the same columns as they were in the preceding row (or relevant row within the preceding group).

6. Up to three levels of OCCURS are permitted. This provides for a 3-dimensional table.

11.3.5 Linkage Section

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 (refer to "CALL Statement" in Section 12.5). The CALL statement in the calling program must contain 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.

Identifiers defined in the Linkage Section of the called program can 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.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section (refer to Section 11.3.2). Thus, the Linkage Section begins 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.

Noncontiguous Linkage Storage

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementary items. Each of these data items is defined in a separate data description entry that begins with the special level-number 77.

The following data clauses are required in each data description entry:

1. Level-number

2. Data-name

3. The PICTURE clause, or, if the PICTURE clause is omitted, the USAGE IS INDEX or the USAGE IS BINARY clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

Linkage Records

Data elements in the Linkage Section that bear a definite hierarchic relationship to one another must be grouped into records according to the rules for formation of record descriptions. Any clause that is used in an input or output record description can be used in a Linkage Section.

Initial Values

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

11.4 Example of Data Division

```
DATA DIVISION.

FILE SECTION.

FD  CLIENT-FILE
    RECORD CONTAINS 60 CHARACTERS
    LABEL RECORDS ARE STANDARD.

01  CLIENT-REC.
    05  SS-NUMBERS  PIC 9(9).
    05  NAME        PIC X(20).
    05  STATE-NAME  PIC X(2).
    05  DUE-DATE.
        10  MM      PIC 99.
        10  DD      PIC 99.
        10  YY      PIC 99.
    05  BALANCE     PIC S9(5)V99  COMP.
    05  PAYMENT     PIC S9(5)V99  COMP.
    05  PRINCIPLE   PIC S9(5)V99  COMP.
    05  INTEREST    PIC S9(3)V999 COMP.
    05  FILLER      PIC X(7).

WORKING-STORAGE SECTION.
77  LINE-CTR    PIC 999 VALUE ZERO.
77  DISCOUNT    PIC S9(3)V999 VALUE ZERO.

01  NEW-DATE.
    05  NMM     PIC 99 VALUE ZERO.
    05  NDD     PIC 99 VALUE ZERO.
    05  NYY     PIC 99 VALUE ZERO.

01  CONSTANTS.
    05  CONSTANT-A  PIC X(8) VALUE "CONSTANT".
    05  SEVEN       PIC S9 VALUE +7.

01  EOF-SWITCH      PIC S9 VALUE +0.
    88  EOF                VALUE +1.
    88  NOT-EOF            VALUE +0.

LINKAGE SECTION.

01  LINKINFO.
    05  DATE-INFO       PIC 9(6).
    05  DELINQUENT-NAME PIC X(20).

PROCEDURE DIVISION USING LINKINFO.
```

CHAPTER 12
PROCEDURE DIVISION

12.1 GENERAL DESCRIPTION

The Procedure Division must be included in every COBOL source program. This division can contain declaratives and nondeclarative procedures.

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. The Declaratives Section provides a method of including procedures that are not executed as part of the logical order of execution, but are executed when a condition occurs that cannot normally be tested by the programmer. The Declaratives Section is used with the USE statement. (Refer to "USE Statement" in Section 12.5.)

The key words DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a period. No other text can appear on the same line.

A procedure is composed of a paragraph, a group of successive paragraphs, 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 in the source program in which it occurs. A procedure name consists of a paragraph name or a section name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear. The size of the Procedure Division cannot exceed 512 K.

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.

12.2 ORGANIZATION

12.2.1 Procedure Division Header

The Procedure Division is identified by and must begin with the following header:

<u>PROCEDURE</u> <u>DIVISION</u> [<u>USING</u> 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 data descriptions given in the called program.

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 identifier-1 in the CALL statement of the calling program refer to a single set of data, data that is equally available to both the called and calling programs. Their descriptions must define an equal number of character positions; 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 identifier-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 identifier can appear more than once in the same USING phrase of a CALL statement.

12.2.2 Procedure Division Body

The body of the Procedure Division must conform to one of the following formats:

Format 1:

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

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

Format 2:

```
{paragraph-name.   [sentence]...}...
```

12.2.3 Statements and Sentences

There are three types of statements: conditional statements, compiler directing statements (declaratives), and imperative statements.

Correspondingly, there are three types of sentences: conditional sentences, compiler directing sentences (declarative), and imperative sentences.

Conditional Statements and Sentences

Definition of 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.

A conditional statement is one of the following:

1. An IF or a SEARCH statement.
2. A READ statement that specifies the AT END, INVALID KEY, or TIMEOUT phrase.
3. A WRITE statement that specifies the INVALID KEY or TIMEOUT phrase.
4. A START, REWRITE, or DELETE statement that specifies the INVALID KEY phrase.
5. An arithmetic statement (ADD, COMPUTE, DIVIDE, MULTIPLY, or SUBTRACT) that specifies the SIZE ERROR phrase.
6. A MOVE WITH CONVERSION, or ROLLBACK statement that specifies the ON ERROR phrase.
7. A HOLD statement that specifies the TIMEOUT or ON ERROR phrase.
8. A DISPLAY AND READ statement that specifies the PFKEY or NOMOD phrase.

Definition of Conditional Sentence -- A conditional sentence is a conditional statement (optionally preceded by an imperative statement or another conditional statement) terminated by a period followed by a space.

Compiler Directing (Declarative) Statements and Sentences

Definition of Compiler Directing Statement -- A compiler directing statement consists of a compiler directing verb and its operands. The compiler directing verbs are COPY, ENTER, and USE. A compiler directing statement causes the compiler to take a specific action during compilation. Refer to the USE statement in Section 12-5 for the declarative-sentence format.

Definition of Compiler Directing Sentence -- A compiler directing sentence is a single compiler directing statement terminated by a period followed by a space.

Imperative Statements and Sentences

Definition of Imperative Statement -- An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement, nor a compiler directing statement. An imperative statement can consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

ACCEPT	EXIT	READ (3)
ADD (1)	FREE (4)	REWRITE (2)
ALTER	GO	ROLLBACK (4)
CALL	HOLD (6)	SET
CLOSE	INSPECT	START (2)
COMPUTE (1)	MOVE	STOP
DELETE (2)	MOVE WITH CONVERSION (4)	SUBTRACT (1)
DISPLAY	MULTIPLY (1)	WRITE (5)
DISPLAY AND READ (7)	OPEN	
DIVIDE (1)	PERFORM	

---

(1) Without the optional SIZE ERROR phrase.
(2) Without the optional INVALID KEY phrase.
(3) Without the optional AT END, INVALID KEY, or TIMEOUT phrase.
(4) Without the optional ON ERROR phrase.
(5) Without the optional INVALID KEY or TIMEOUT phrase.
(6) Without the optional TIMEOUT or ON ERROR phrase.
(7) Without the optional PFKEY or NOMOD phrase.

When "imperative-statement" appears in the General Format of statements, "imperative-statement" refers to that sequence of consecutive imperative statements that must be ended by a period or an ELSE phrase associated with a previous IF statement.

Definition of Imperative Sentence -- An imperative sentence is an imperative statement terminated by a period followed by a space.

Categories of Statements

Category	Verbs
Arithmetic	ADD COMPUTE DIVIDE INSPECT (TALLYING) MULTIPLY SUBTRACT
Compiler-Directing	COPY ENTER USE
Conditional	ADD (SIZE ERROR) CALL COMPUTE (SIZE ERROR) DELETE (INVALID KEY) DISPLAY AND READ (PFKEY or NOMOD) DIVIDE (SIZE ERROR) HOLD (ON ERROR) IF MOVE WITH CONVERSION (ON ERROR) MULTIPLY (SIZE ERROR) READ (END or INVALID KEY) REWRITE (INVALID KEY) ROLLBACK (ON ERROR) SEARCH START (INVALID KEY) SUBTRACT (SIZE ERROR) WRITE (INVALID KEY)
Data Movement	ACCEPT (DATE, DAY, or TIME) INSPECT (REPLACING) MOVE MOVE WITH CONVERSION
Ending	STOP
Input/Output	ACCEPT (identifier) CLOSE DELETE DISPLAY DISPLAY AND READ FREE HOLD OPEN READ REWRITE ROLLBACK START STOP (literal) WRITE

Inter-Program Communicating	CALL
Procedure Branching	ALTER CALL EXIT GO TO PERFORM
Table Handling	SEARCH SET

12.3 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 can be preceded by a unary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in Table 12-1.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

12.3.1 Arithmetic Operators

There are five binary arithmetic operators and two unary arithmetic operators that can be used in arithmetic expressions. Arithmetic operators 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
**	Exponentiation

Unary Arithmetic Operators	Meaning
+	The effect of multiplication by numeric literal +1
-	The effect of multiplication by numeric literal -1

12.3.2 Formation and Evaluation Rules

1. Parentheses can 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--Exponentiation
   3rd--Multiplication and division
   4th--Addition and subtraction

2. 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 specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which operators, variables, and parentheses can be combined in an arithmetic expression are summarized in Table 12-1 where the following indicators are used:

   a. The letter 'P' indicates a permissible pair of symbols.
   b. The character 'I' indicates an invalid pair.
   c. "Variable" indicates an identifier or literal.

Table 12-1. Combination of Symbols in Arithmetic Expressions

FIRST SYMBOL	SECOND SYMBOL				
	Variable	* / ** - +	Unary + or -	(	)
Variable	I	P	I	I	P
* / ** + -	P	I	P	P	I
Unary + or -	P	I	I	P	I
(	P	I	P	P	I
)	I	P	I	I	P

4. An arithmetic expression can only begin with the symbol '(', '+', '-', or a variable and can 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.

5. Arithmetic expressions allow the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items except for the right operand of an exponentiation. The composite of operands must not contain more than 18 digits.

6. The following rules apply to the evaluation of exponentiation in arithmetic expresions:

   a. The exponent must be either an integer numeric literal or a numeric data item whose picture has no digits to the right of the decimal point.

   b. If the value of an expression to be raised to a power is zero, the exponent must have a value greater than zero. Otherwise, a runtime error will occur.

   c. If the value of the exponent is negative, a runtime error will occur.

7. Certain uses of arithmetic expressions utilize an intermediate result field that may affect the precision of the final result. Refer to Appendix G, Intermediate Results.

12.3.3 Arithmetic Statements

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

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

2. The maximum size of each operand is 18 decimal digits. The composite of operands specified in a statement must not contain more than 18 decimal digits. The composite of operands is a hypothetical data item resulting from the superimposition of operands aligned on their decimal points.

Common Phrases and General Rules for Statement Formats

Phrases appearing frequently in the statement descriptions that follow are the ROUNDED phrase and the SIZE ERROR phrase.

In the discussion that follows, a resultant-identifier is the identifier that is associated with a result of an arithmetic operation.

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 whenever the most significant digit of the excess is greater than or equal to five.

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.

SIZE ERROR Phrase

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by 0 always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action is as follows:

1. If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR phrase is specified and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

3. If the SIZE ERROR phrase is specified and a size error occurs for any of the operations of an ADD CORRESPONDING or SUBTRACT CORRESPONDING statement, the imperative statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

Overlapping Operands

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

Incompatible Data

Except for class conditions (refer to Section 12.4.1), when the contents of a data item are referenced in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, the result of such a reference is undefined.

12.4 CONDITIONS

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, PERFORM and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions. Each can be enclosed within any number of paired parentheses, in which case its category is not changed.

12.4.1 Simple Conditions

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

Relation Conditions

A relation condition causes a comparison of two operands, each of which can be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic expression. A relation condition has a 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:

$$\left\{\begin{array}{l}\text{identifier-1}\\ \text{literal-1}\\ \text{arithmetic-expression-1}\end{array}\right\} \left\{\begin{array}{l}\text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{GREATER}}\text{ THAN}\\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{LESS}}\text{ THAN}\\ \text{IS [}\underline{\text{NOT}}\text{] }\underline{\text{EQUAL}}\text{ TO}\\ \text{IS [}\underline{\text{NOT}}\text{] } >\\ \text{IS [}\underline{\text{NOT}}\text{] } <\\ \text{IS [}\underline{\text{NOT}}\text{] } =\end{array}\right\} \left\{\begin{array}{l}\text{identifier-2}\\ \text{literal-2}\\ \text{arithmetic-expression-2}\end{array}\right\}$$

NOTE

The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '$\underline{>}$' (greater than or equal to).

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is called the subject of the condition; the second operand (identifier-2, literal-2, or arithmetic-expression-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 meanings of the relational operators are 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 '$\geq$' (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 literal or arithmetic expression 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 the ASCII collating sequence of characters. An edited item is considered nonnumeric for comparisons. Comparison of two literals is not allowed. If one of the operands is specified as numeric, it must be an integer data item or an integer literal. Comparisons between numeric and nonnumeric operands are made as follows:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though: (1) it were moved to an elementary alphanumeric data item of the same size (in terms of standard data format characters) as the numeric data item; and (2) the contents of this alphanumeric data item were then compared to the nonnumeric operand.

2. If the nonnumeric operand is a group item, the numeric operand is treated as though: (1) it were moved to a group item of the same size (in terms of standard data format characters) as the numeric data item; and (2) the contents of this group item were then compared to the nonnumeric operand.

3. 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. There are two cases to consider: operands of equal size and operands of unequal size.

1. 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.

2. 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 Involving Index Names and/or Index Data Items -- Relation tests can be made between:

1. The values associated with two index names. The result is the same as if the corresponding occurrence numbers were compared.

2. The value of an index name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index name is compared to the data item or literal.

3. An index data item and another index data item or the value of an index name. The actual values are compared without conversion.

4. The comparison of an index data item with any data item or literal not specified in Rule 3 is illegal. In IF statements involving an index data item, the index data item may not be the operand of an arithmetic expression. For example,

   ```
   IF INDX-ITEM = 3 * COMP-ITEM
   ```

   is a legal statement, but

   ```
   IF INDX-ITEM * 3 = 3
   ```

   is not legal.

Comparisons Involving Binary Data Items -- Relation tests can be made between binary data items and any other integer data item.

Class Conditions

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', or space. The general format for the class condition is as follows:

$$\boxed{\text{identifier IS } [\underline{\text{NOT}}] \begin{Bmatrix} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{Bmatrix}}$$

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 described with the SIGN IS SEPARATE clause are the standard data format characters, '+' and '-'. Refer to "SEPSGN" in Appendix B for a discussion of what constitutes valid sign(s) for data items not described with the SIGN IS SEPARATE clause.

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 Conditions (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:

$$\boxed{\text{condition-name}}$$

If the condition name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions (refer to "Relation Conditions" earlier in this subsection).

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 Conditions

A switch-status condition determines the ON or OFF status of a switch. The switch 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:

```
condition-name
```

The result of the test is true if the switch is set to the specified position corresponding to the condition name.

Sign Conditions

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

$$\text{arithmetic-expression IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{POSITIVE}} \\ \underline{\text{NEGATIVE}} \\ \underline{\text{ZERO}} \end{array} \right\}$$

When used, NOT and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; e.g., NOT ZERO is a truth test for a nonzero (positive or negative) value. An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

Modified Data Tag Conditions

When a field of the workstation screen is modified by the user, Bit 1 of its FAC is set to 1. This bit is referred to as the modified data tag (MDT). It can be tested to determine if modification has been made to the associated field. The general format for the MDT condition is as follows:

$$\underline{\text{FAC}}\ \underline{\text{OF}}\ \text{display-item}\ \underline{\text{ALTERED}}$$

The result of the test is true if Bit 1 of the specified FAC is set to 1.

Figurative Constant Conditions

A user-figurative-constant may be defined to test a bit or any combination of bits in any 1-byte item. The general format for figurative constant conditions is as follows:

$$\text{figurative-constant} \quad \left\{\begin{array}{l}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\} \left\{\begin{array}{l}\text{identifier}\\ \underline{\text{FAC}}\ \underline{\text{OF}}\ \text{display-item}\end{array}\right\} \quad \text{IS } [\underline{\text{NOT}}] \quad \left\{\begin{array}{l}\underline{\text{ON}}\\ \underline{\text{OFF}}\end{array}\right\}$$

The ON condition is true if the bits are set to 1 in the FAC or identifier are those bits corresponding to the bits set to 1 in the user-figurative-constant.

12.4.2 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.
Logical Operator	**Meaning**
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:

$$\boxed{\underline{\text{NOT}}\ \text{simple-condition}}$$

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:

$$\boxed{\text{condition} \left\{ \begin{Bmatrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{Bmatrix} \text{condition} \right\} \ldots}$$

Where "condition" can be:

1. A simple condition
2. A negated simple condition
3. A combined condition
4. A negated combined condition, i.e., the NOT logical operator followed by a combined condition enclosed within parentheses
5. Combinations of the previous four, specified according to the rules summarized in Table 12-2.

Although parentheses need never be used when either AND or OR (but not both) is used exclusively in a combined condition, parentheses can be used to effect a final truth value when a mixture of AND, OR and NOT is used. (Refer to Table 12-2 and Section 12.4.3, Condition Evaluation Rules.)

Table 12-2 indicates the ways in which conditions and logical operators can 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.

Table 12-2. Combinations of Conditions, Logical Operators, and Parentheses

Given the following element	Location in conditional expression		In a left-to-right sequence of elements:	
	First	Last	Element, when not first, may be immediately preceded by only:	Element, when not last, may be immediately followed by only:
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition,(
(	Yes	No	OR, NOT, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

The element pair "OR NOT" is permissible while the pair "NOT OR" is not permissible; "NOT (" is permissible while "NOT NOT" is not permissible.

<u>Abbreviated Combined Relation Conditions</u>

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition, and no parentheses are used within such a consecutive sequence, any relation condition except the first may be abbreviated by either of the following:

1. The omission of the subject of the relation condition

2. The omission of the subject and relational operator of the relation condition.

The format for an abbreviated combined relation condition is:

$$\text{relation-condition} \left\{ \begin{Bmatrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{Bmatrix} [\underline{\text{NOT}}]\ [\text{relational-operator}]\ \text{object} \right\} \ldots$$

Within a sequence of relation conditions, both forms of abbreviation can be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last preceding stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules of Table 12-2. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation applied to the use of the word NOT in an abbreviated combined relation condition is as follows:

1. If the word immediately following NOT is GREATER, >, LESS, <, EQUAL, or =, then the NOT participates as part of the relational operator.

2. Otherwise, the NOT is interpreted as a logical operator, and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

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

12.4.3 Condition Evaluation Rules

Parentheses can 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 evaluation is implied until the final truth value is determined.

1. Values are established for arithmetic expressions. (Refer to Section 12.3.2., Formation and Evaluation Rules)

2. Truth values for simple conditions are established.

3. Truth values for negated simple conditions are established.

4. Truth values for combined conditions are established in the following order.

   1st -- AND logical operators
   2nd -- OR logical operators

5. Truth values for negated combined conditions are established.

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

12.5 PROCEDURE DIVISION STATEMENTS

ACCEPT Statement

Function

The ACCEPT statement causes low volume data to be made available to the specified data item.

General Format

Format 1

```
    ACCEPT identifier-1 [identifier-2]...
    ------
```

Format 2

```
                                  { DATE }
                                  { ---- }
    ACCEPT identifier  FROM       { DAY  }
    ------             ----       { ---  }
                                  { TIME }
                                  { ---- }
```

General Rules

Format 1

1. The ACCEPT statement causes the transfer of alphanumeric data by issuing a request that can be fulfilled through a procedure or by the operator's response to a system prompt. If the data is not available from a procedure specification, the workstation operator will be prompted for it. The character-string from the procedure ENTER statement or from the workstation replaces the contents of the data item referenced by identifier-1, identifier-2, ... No editing, validation, or conversion of the input data takes place.

2. Any workstation file defined by the program is not affected by the identifier data provided by an operator response at the workstation, unless that identifier is contained in a screen record entry of that workstation file.

3. If the size of the receiving data item exceeds 68 characters, the transferred data is stored aligned to the left in the receiving data item.

4. For USAGE IS BINARY or COMPUTATIONAL items, the length of the data entry field displayed on the screen will be equivalent to the internal length of the data item. Data transferred from the screen to storage will not be converted to binary or packed decimal format.

5. The compiler accepts nonnumeric data for a field whose PICTURE is numeric.

6. Up to 16 separate data items can be accepted.

7. In the Procedure Language entry, ACCEPT is used as the parameter reference name, and the identifier (right truncated to eight characters) is used as the keyword for the operator prompt.

NOTE

If data conversion or validation of numeric input is desired, the programmer should:

1. Declare a dummy alphanumeric variable to be used in the Accept statement.

2. Declare a target variable of the required type.

3. Use MOVE WITH CONVERSION, after ACCEPT, to assign the desired value to the target variable.

For example:

```
01    DUMMY    PIC XX.
01    NUMB     PIC 99 COMP.

ACCEPT DUMMY
MOVE WITH CONVERSION DUMMY TO NUMB
     ON ERROR... .
```

Format 2

8. The ACCEPT statement causes the information requested to be transferred to the data item specified by the identifier according to the rules of the MOVE statement. DATE, DAY and TIME are conceptual data items and, therefore, must not be described in the COBOL program.

9. 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, 1983 would be expressed as 830701. DATE, when accessed by a COBOL program, behaves as if it had been described in the COBOL program as an unsigned, elementary, numeric integer six digits in length.

10. 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, 1983 would be expressed as 83182. DAY, when accessed by a COBOL program, behaves as if it had been described in a COBOL program as an unsigned, elementary, numeric integer five digits in length.

11. 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 as 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 eight digits in length. The minimum value of TIME is 00000000; the maximum value of TIME is 23595999.

ADD Statement

Function

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

General Format

```
Format 1
     ADD { identifier-1 } [ identifier-2 ] ...TO identifier-m [ROUNDED]
         { literal-1    } [ literal-2    ]
        [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]
```

Syntax Rules

1. In Formats 1 and 2, each identifier must refer to an elementary numeric 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, each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits (refer to Subsection 12.3.3, Arithmetic Statements).

   a. In Format 1 the composite of operands is determined by using all of the operands in a given statement.

   b. In Format 2 the composite of operands is determined by using all of the operands in a given statement, excluding the identifier that follows the word GIVING.

   c. In Format 3 the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. Refer to "ROUNDED Phrase", "SIZE ERROR Phrase", and "Overlapping Operands" in Subsection 12.3.3, Arithmetic Statements, and the CORRESPONDING Phrase in Subsection 11.2.1.

2. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-m, and the result stored immediately into identifier-m.

3. If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m, the resultant-identifiers.

4. When ADD is used, enough places are carried so that no significant digits are lost during execution.

5. If Format 3 is used, data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

Examples of ADD Statement

```
ADD INVENTORY-TOTAL RECEIVED-TOTAL GIVING NEW-INVENTORY-TOTAL
```

The values of INVENTORY-TOTAL and RECEIVED-TOTAL are added and the sum placed in NEW-INVENTORY-TOTAL.

```
ADD VOL1 VOL2 VOL3 GIVING VOL4
```

The sum of VOL1, VOL2 and VOL3 is placed in VOL4.

```
ADD LDATA TO RDATA ON SIZE ERROR GO TO ERR-MESSAGE
```

LDATA is added to RDATA and the sum is placed in RDATA. If the length of the sum exceeds the described length of RDATA, a SIZE ERROR condition exists and control is passed to ERR-MESSAGE.

```
ADD CORR FILE1 TO FILE2.
```

Elementary items from FILE1 are added to and stored in corresponding elementary data items in FILE2.

ALTER Statement

Function

The ALTER statement modifies a predetermined sequence of operations.

General Format

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2 ...
```

Syntax Rules

1. Each procedure-name-1 is the name of a paragraph that contains a single sentence consisting of a GO TO statement without the DEPENDING phrase.

2. Each procedure-name-2 is the name of a paragraph or section in the Procedure Division.

General Rules

1. Execution of the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, so that subsequent executions of the modified GO TO statements cause transfer of control to procedure-name-2.

CALL Statement

Function

The CALL statement causes control to be transferred from one object program to another, within the run unit.

General Format

```
CALL literal-1 [USING identifier-1 [identifier-2]...]
```

Syntax Rules

1. Literal-1 must be a nonnumeric literal.

2. 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. The number of operands in each USING phrase must be identical.

3. 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.

4. Literal-1 can be a maximum of eight characters. The first character must be alphabetic and the rest can be alphanumeric. Literal-1 must not be a reserved word.

5. Literal-1 specifies the entry-point name associated with the called program. This name must agree with the program-ID of the called program (refer to Section 9.2.1, PROGRAM-ID Paragraph).

General Rules

1. The program whose name is specified by the value of literal-1 is the called program; the program in which the CALL statement appears is the calling program.

2. The execution of a CALL statement causes control to pass to the called program.

3. A called program is in its initial state the first time it is called within a run unit.

   On all other entries into the called program, the state of the program remains unchanged from its state when last exited. This includes all data fields, and the status and positioning of all files.

4. Called programs can contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

5. The identifiers specified by the USING phrase of the CALL statement indicate those data items available to a calling program that can be used in the called program. The order of appearance of the identifiers in the USING phrase of the CALL statement and the USING phrase in the Procedure Division header is critical. Identifiers appearing in the same relative position in the respective USING phrases refer to a single set of data that is available to both the called and the 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.

6. Upon the return from a CALLED program, the RETURN-CODE is set.

7. Refer to Appendix H, Passing Parameters to COBOL Subroutines, for further rules governing the use of the CALL statement.

Example of CALL Statement

```
IF ERROR-COUNT IS GREATER THAN 10
    THEN CALL "ERRORRTN" USING ERROR-COUNT
    ELSE GO TO STEP-ONE.
DISPLAY ERROR-COUNT.
```

The name of the subroutine (subprogram) is ERRORRTN. ERRORRTN is CALLed only if the value of ERROR-COUNT is greater than 10, in which case all of the code included in the program ERRORRTN is executed. Control then returns to the calling program, at the first statement following the CALL (DISPLAY ERROR-COUNT).

CLOSE Statement -- for Consecutive Files

Function

The CLOSE statement terminates the processing of reels/units and files with optional rewind and/or lock or removal where applicable.

General Format

```
                         [{REEL}  [WITH NO REWIND]]
                         [{UNIT}  [FOR REMOVAL   ]]
CLOSE file-name-1        [                         ]
                         [ WITH {NO REWIND}        ]
                         [      {LOCK     }        ]
        [                [{REEL}  [WITH NO REWIND]  ]]
        [                [{UNIT}  [FOR REMOVAL   ]  ]] ...
        [ file-name-2    [                          ]]
        [                [ WITH {NO REWIND}         ]]
        [                [      {LOCK     }         ]]
```

Syntax Rules

1. The REEL or UNIT phrase can only be used for tape files.

2. The files referenced in the CLOSE statement need not all have the same organization or access.

General Rules

1. A CLOSE statement can only be executed for a file in an open mode.

2. When the LOCK phrase is specified, the file cannot be opened again during execution of this run unit.

3. The REEL/UNIT phrase, for magnetic tape devices, causes the next executed READ statement for that file to make available the next data record on the new reel/unit. If no additional reel units are available, the end-of-file condition exists. For output files, the next executed WRITE statement that references that file directs the next logical data record to the next reel/unit of the file.

4. The REWIND phrase causes the current reel device to be positioned at its physical beginning.

5. The terms REEL and UNIT are synonymous and completely interchangeable in the CLOSE statement.

6. The terms REEL, UNIT, FOR REMOVAL and WITH NO REWIND are ignored by the operating system for all devices except magnetic tape.

7. If a CLOSE statement without the REEL or UNIT phrase 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.

8. Following the successful execution of a CLOSE statement without the REEL or UNIT phrase, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

9. The WITH NO REWIND and FOR REMOVAL phrases have no effect at object time if they do not apply to the storage media on which the file resides.

10. When a STOP RUN statement is executed the operating system attempts to close any files in the open mode.

CLOSE Statement -- for Indexed and Relative Files

Function

The CLOSE statement terminates the processing of files with an option of locking the files from further processing during the current program execution.

General Format

$$\underline{\text{CLOSE}} \left\{ \text{file-name-1 [WITH } \underline{\text{LOCK}}\text{]} \right\} \ldots$$

Syntax Rules

1. Files referenced in the CLOSE statement need not have the same organization or access.

General Rules

1. A CLOSE statement can only be executed for a file in the open mode.

2. If the WITH LOCK phrase is specified, the file is closed and cannot be opened again during the current program execution.

3. The operating system attempts to close any open files when a STOP RUN statement is executed.

4. 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.

5. Following the successful execution of a CLOSE statement, the record area associated with file-name-1 is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

COMPUTE Statement

Function

The COMPUTE statement assigns to one or more data items the value of an arithmetic expression.

General Format

```
COMPUTE  identifier-1 [ROUNDED] = arithmetic-expression

   [ON SIZE ERROR  imperative-statement]
```

Syntax Rules

1. Identifiers that appear only to the left of '=' must refer to either an elementary numeric item or an elementary numeric edited item.

General Rules

1. Refer to "ROUNDED Phrase", "SIZE ERROR Phrase", and "Overlapping Operands" in Subsection 12.3.3, Arithmetic Statements.

2. An arithmetic expression consisting of a single identifier or literal provides a method of setting the values of identifier-1 equal to the value of the single identifier or literal.

3. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY and DIVIDE. (Refer to each statement in this section and Subsection 12.3.3, Arithmetic Statements.)

4. For techniques used in handling arithmetic expressions, refer to Appendix G, Intermediate Results.

Example of COMPUTE Statement

```
COMPUTE RESULT ROUNDED = (A * (B - C)) + D
```

C is subtracted from B to give n. A is then multiplied by n and this product is added to D. The sum rounded is placed in RESULT.

COPY Statement

Function

The COPY statement incorporates text into any division of a COBOL source program.

COBOL libraries contain library texts that are available to the compiler for copying at compile time. The effect of the interpretation of the COPY statement is to insert text into the source program, where it is treated by the compiler as part of the source program.

COBOL library text is placed on the COBOL library by a text editor or other program as a function independent of the COBOL program.

General Format

$$\underline{\text{COPY}} \left\{\begin{array}{l}\text{file-name}\\ \text{literal-1}\end{array}\right\} \left[\left\{\begin{array}{l}\underline{\text{IN}}\\ \underline{\text{OF}}\end{array}\right\} \left\{\begin{array}{l}\text{library-name}\\ \text{literal-2}\end{array}\right\} \left[\left\{\begin{array}{l}\underline{\text{OF}}\\ \underline{\text{IN}}\\ \underline{\text{ON}}\end{array}\right\}\left\{\begin{array}{l}\text{volume-name}\\ \text{literal-3}\end{array}\right\}\right]\right] .$$

Syntax Rules

1. The COPY statement must be preceded by a space and terminated by the separator period.

2. File-name cannot be a reserved word. Library-name and volume-name cannot be reserved words, except that library-name may be LIBRARY.

3. A COPY statement can occur in the source program anywhere a character-string or a separator may occur except that a COPY statement must not occur within a COPY statement.

4. COPY statements are not restricted to the Procedure Division. They are permitted in all parts of the source program.

5. If more than one COBOL library is available during compilation, file-name must be qualified by either library-name or volume-name.

6. Within one COBOL library, each file name must be unique.

7. Literal-1, -2, and -3 must be nonnumeric literals.

General Rules

1. 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.

2. The effect of processing a COPY statement is that the library text associated with file-name is copied into the source program, logically replacing the entire COPY statement, beginning with the reserved word COPY and inclusive of the punctuation character period.

3. The library text is copied unchanged.

4. Comment lines appearing in the library text are copied into the source program unchanged.

5. Debugging lines are permitted within library text. If a COPY statement is specified on a debugging line, then the text that is the result of the processing of the COPY statement appears as though it were specified on debugging lines with the following exception: comment lines in library text appear as comment lines in the resultant source program.

6. The text produced as a result of the complete processing of a COPY statement must not contain a COPY statement.

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

8. Library text must conform to the rules for COBOL reference format.

9. If library-name is not specified, the default input library name is used. If no default input library has been provided, the default library of LIBRARY is provided. If the file is not in the specified or default library, the system issues an operator prompt requesting the correct file name, library name, and volume name of the file to be copied.

10. The operator is prompted for the volume if this information is not supplied with the default input library name. There is no program control over the value of volume, as there is with value of file-name and library-name.

Example of COPY Statement

```
        CALC-INTEREST.
        COPY INTEREST-ROUTINE OF GS.
        IF INTEREST NOT GREATER THAN ZERO
            THEN GO TO NEXT-ENTRY
            ELSE NEXT SENTENCE.
```

This program requests that the section of code contained in INTEREST-ROUTINE of the GS library be inserted into the program at compile time. If INTEREST-ROUTINE contains the following code,

```
        COMPUTE TOTAL-AMOUNT = PAYMENTS + EARNINGS.
        COMPUTE ANNUAL-TOTAL = 12 * TOTAL-AMOUNT.
        COMPUTE INTEREST = ANNUAL-TOTAL * .053.
```

then the code that is actually compiled is:

```
        CALC-INTEREST.
            COPY INTEREST-ROUTINE OF GS.
            COMPUTE TOTAL-AMOUNT = PAYMENTS + EARNINGS.
        COMPUTE ANNUAL-TOTAL = 12 * TOTAL-AMOUNT.
        COMPUTE INTEREST = ANNUAL-TOTAL * .053.
        IF INTEREST NOT GREATER THAN ZERO
            THEN GO TO NEXT-ENTRY
            ELSE NEXT SENTENCE.
```

DELETE Statement -- for Indexed Files

Function

The DELETE statement logically removes a record from a mass storage file.

General Format

```
DELETE file-name RECORD [INVALID KEY imperative-statement]
```

Syntax Rules

1. The INVALID KEY phrase must not be specified for a DELETE statement that references a file in SEQUENTIAL or DYNAMIC access mode.
2. The INVALID KEY phrase must be specified for a DELETE statement that references a file that is in RANDOM access mode and for which an applicable USE procedure is not specified.

General Rules

1. The file named by file-name must be an indexed file.

2. The associated file must be open in the I-O or shared mode at the time of the execution of this statement.

3. 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 WITH HOLD statement. The operating system logically removes from the file the record that was accessed by that READ statement.

4. For a file in RANDOM access mode, the operating system logically removes from the file the record identified by the contents of the primary record key associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

5. 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.

6. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

7. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

8. The current record pointer is not affected by the execution of a DELETE statement.

DELETE Statement -- for Relative Files

Function

The DELETE statement logically removes a record from a mass storage file.

General Format

```
DELETE file-name RECORD [INVALID KEY imperative-statement]
```

Syntax Rules

1. The INVALID KEY phrase must not be specified for a DELETE statement that references a file in SEQUENTIAL access mode.

2. The INVALID KEY phrase must be specified for a DELETE statement that references a file that is in RANDOM or DYNAMIC access mode and for which an applicable USE procedure is not specified.

General Rules

1. The file named by file-name must be a relative file.

2. The associated file must be open in the I-O mode at the time of the execution of this statement.

3. 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 operating system logically removes from the file the record that was accessed by that READ statement.

4. For a file in RANDOM or DYNAMIC access mode, the operating system logically removes from the file the record identified by the contents of the RELATIVE KEY data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

5. 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.

6. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

7. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

8. The current record pointer is not affected by the execution of a DELETE statement.

DISPLAY Statement

Function

The DISPLAY statement causes low volume data to be transferred to the workstation.

General Format

$$\underline{\text{DISPLAY}} \begin{Bmatrix} \text{identifier-1} \\ \text{literal-1} \end{Bmatrix} \begin{bmatrix} \text{identifier-2} \\ \text{literal-2} \end{bmatrix} \ldots$$

Syntax Rules

1. Each literal can be any figurative constant.

2. The usage of the identifier should be DISPLAY and must not be INDEX.

3. If the literal is numeric, it must be an unsigned integer.

General Rules

1. The DISPLAY statement causes the contents of each operand to be transferred to the workstation in the order listed. Program execution is then suspended until the operator acknowledges the message by pressing the ENTER key. If the workstation is OPEN as a file, the contents of the screen are saved before the message is displayed and are restored when the operator presses ENTER.

2. A maximum of 79 characters are displayed on each workstation row up to a maximum of 18 rows.

3. When a DISPLAY statement contains more than one operand, the size of the sending item is the sum of the sizes associated with the operands, and the values of the operands are transferred in the sequence in which the operands are encountered.

4. If a figurative constant is specified as one of the operands only a single occurrence of the figurative constant is displayed.

DISPLAY AND READ Statement

Function

The DISPLAY AND READ statement automatically controls the movement of data to be displayed on and read from the workstation screen. This statement, with its associated phrases, moves values from the specified fields to the display location. The operator can input or modify any displayed value. DISPLAY AND READ can verify the operator response and move the displayed value to a specified data field. For further explanation of this statement, refer to Subsection 4.3.3, Procedure Division Requirements for DISPLAY AND READ.

General Format

```
DISPLAY AND READ [ALTERED] record-name ON file-name
 ┌                                                                      ┐
 │ [ONLY]  {PFKEY }  {identifier-1}  [identifier-2]  ...                │
 │         {PFKEYS}  {integer-1   }  [integer-2   ]                     │
 │┌                                                                    ┐│
 ││ ON     {PFKEY }  {identifier-3}  [identifier-4]  imperative-statement-1││
 ││        {PFKEYS}  {integer-3   }  [integer-4   ]                     ││
 │└                                                                    ┘│
 └ [NO-MOD imperative-statement-2]                                      ┘
```

Syntax Rules

1. Record-name must reference a screen record with usage of DISPLAY-WS.

2. File-name must reference a workstation file (refer to Section 4.3, Coding Requirements for DISPLAY AND READ) that is assigned to device DISPLAY.

3. The valid range of values for any of the integers is 1 through 32, inclusive.

4. All PF keys specified in the ON phrase must also be specified in the ONLY phrase.

5. The NO-MOD phrase is valid only if ALTERED is specified.

General Rules

1. Up to 32 integer and identifier phrases can be used with the ON and ONLY phrases.

2. If ALTERED is specified, only those fields that have been modified by the user are read.

3. The DISPLAY AND READ statement causes the following sequence of events to occur:

   a. Data is moved from the SOURCE clause data items (if specified) to the appropriate locations in the USAGE IS DISPLAY-WS record.

   b. The record defined in the USAGE IS DISPLAY-WS clause is displayed on the screen.

   c. User responses are read.

   d. Data is validated according to the specifications of the PICTURE and RANGE clauses. Alphanumeric and numeric edited data will be converted to numeric before comparison with numeric RANGE items. If user responses violate record description requirements, they are flagged as errors by their FACs being set to blink. Already-blinking fields, however, may have their FACs reset to a nonblinking mode.

   e. User responses that meet the record description requirements are transferred to the appropriate OBJECT fields. When data is moved from numeric edited screen fields to numeric OBJECT fields, currency signs, CR, DB, asterisks, and commas (or periods if DECIMAL-POINT IS COMMA) are removed.

   f. Protected fields are neither validated, since invalid data could not be corrected, nor deedited. Therefore, if the FAC of a numeric-edited DISPLAY-WS field is changed to protected on a DISPLAY AND READ, a subsequent DISPLAY AND READ may move invalid data to the OBJECT field. To avoid this, the subsequent DISPLAY AND READ should use the ALTERED phrase, which allows only modified fields to be read.

4. Scaling characters or implied decimals in the PICTUREs of workstation record entries are ignored in the screen display and in the transfer of data from the SOURCE field to the screen or from the screen to the OBJECT field. These characters do not display on the screen, nor are they replaced by blank spaces. For purposes of data movement, a display field with such a PICTURE is treated as an integer field, unless decimal points are explicitly entered from the workstation by the user.

5. The ONLY phrase is used to specify which PF keys are valid operator responses when DISPLAY AND READ is issued. The word ONLY is optional in the ONLY phrase. If it is coded, the sole valid operator response is one of the specified PF keys. If it is not coded, the valid operator response is one of the specified PF keys or the ENTER key. If the operator response is invalid, the audio alarm is sounded, no data transfer occurs, and the screen record is displayed again.

6. If the ONLY phrase is not coded, the sole valid operator response is the ENTER key.

7. When a DISPLAY AND READ has been issued, a valid operator response can have one of two functions: to signal that data has been entered, and to indicate a special condition request, such as program termination, without entering data. The ON PFKEYS phrase identifies the PF keys used for special condition requests. When a key numbered in ON PFKEY is pressed, the imperative statement is executed, and no data is entered.

8. If a valid PF key or valid ENTER key is selected, but a data input error is detected, no transfer of data occurs. The screen record is displayed again with the field(s) in error blinking, and the audio alarm is sounded. The operator must enter another response. This cycle continues until a correct response is entered or the program execution is terminated.

9. If a valid PF key or valid ENTER key is selected and there are no input data errors, data transfer occurs.

10. The imperative statement of the NO-MOD phrase is only executed if there is no modification of any of the displayed fields. The NO-MOD phrase is ignored if an ON PFKEY option is selected.

DIVIDE Statement

Function

The DIVIDE statement divides one numeric data item into another and sets the values of data items equal to the quotient and remainder.

General Format

```
Format 1
    DIVIDE  {identifier-1}  INTO identifier-2 [ROUNDED]
            {literal-1   }
        [ON SIZE ERROR imperative-statement]
Format 2
    DIVIDE  {identifier-1}  INTO  {identifier-2}  GIVING identifier-3 [ROUNDED]
            {literal-1   }        {literal-2   }
        [ON SIZE ERROR imperative-statement]
Format 3
    DIVIDE  {identifier-1}  BY    {identifier-2}  GIVING identifier-3 [ROUNDED]
            {literal-1   }        {literal-2   }
        [ON SIZE ERROR imperative-statement]
Format 4
    DIVIDE  {identifier-1}  INTO  {identifier-2}  GIVING identifier-3 [ROUNDED]
            {literal-1   }        {literal-2   }
        REMAINDER identifier-4 [ON SIZE ERROR imperative-statement]
Format 5
    DIVIDE  {identifier-1}  BY    {identifier-2}  GIVING identifier-3 [ROUNDED]
            {literal-1   }        {literal-2   }
        REMAINDER identifier-4 [ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. Each identifier must refer to an elementary numeric item, except that any identifier associated with the GIVING or REMAINDER phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits. The composite of operands is the hypothetical data item resulting from the superimposition of all receiving data items (except the REMAINDER data item) of a given statement aligned on their decimal points.

General Rules

1. Refer to "ROUNDED Phrase", "SIZE ERROR Phrase", and "Overlapping Operands" in Subsection 12.3.3, Arithmetic Statements.

2. When Format 1 is used, 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.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3.

4. When Format 3 is used, 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.

5. Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. Identifier-4 must not be defined as USAGE IS BINARY. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If identifier-3 is defined as a numeric edited item, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field that contains the quotient of the DIVIDE statement truncated rather than rounded. Refer to Appendix G, Intermediate Results.

6. In Formats 4 and 5, the accuracy of the REMAINDER data item (identifier-4) is defined by the calculation described in General Rule 5. Appropriate decimal alignment and truncation (not rounding) are performed as needed for the content of the data item referenced by identifier-4.

7. When the ON SIZE ERROR phrase is used in Formats 4 and 5, the following rules pertain:

   a. If the size error occurs on the quotient, no remainder calculation is meaningful. Thus, the contents of the data items referenced by both identifier-3 and identifier-4 remain unchanged.

   b. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 remain unchanged. However, as with other instances of multiple results of arithmetic statements, the user will have to analyze the situation to determine what has actually occurred.

Examples of DIVIDE Statement

```
        DIVIDE 3 INTO VALUE-ITEMS.
```

If VALUE-ITEMS has the value 24, then the result 8 is placed in the data item VALUE-ITEMS.

```
        DIVIDE 10 INTO VALUE-ITEMS GIVING AVG-VALUE.
```

If VALUE-ITEMS has the value 24, then the result (2.4) is placed in AVG-VALUE. AVG-VALUE has a PICTURE of S99V9.

```
        DIVIDE VALUE-ITEMS BY NUMBER-ITEMS GIVING AVG-VALUE
            REMAINDER REM-ITEMS.
```

If VALUE-ITEMS IS 34 AND NUMBER-ITEMS is 5, then the result (6) is placed in AVG-VALUE and the remainder 4 is placed in REM-ITEMS.

In this example, AVG-VALUE has a PICTURE of S99.

ENTER Statement

Function

The ENTER statement is treated as a comment by this compiler.

General Format

```
ENTER language-name [routine-name].
```

EXIT Statement

Function

The EXIT statement provides a common end point for a series of procedures.

General Format

```
EXIT.
```

Syntax Rules

1. The EXIT statement must appear in a sentence by itself.

2. The EXIT sentence must be the only sentence in the paragraph.

General Rules

1. An EXIT statement 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.

EXIT PROGRAM Statement

Function

The EXIT PROGRAM statement marks the logical end of a called program.

General Format

```
EXIT PROGRAM.
```

General Rules

1. Execution of an EXIT PROGRAM statement in a called program causes control to be returned to the calling program. An EXIT PROGRAM statement executed in a program that is not called behaves as an EXIT statement.

FREE Statement

Function

The FREE statement frees resources that have been previously held. The FREE ALL statement is used in DMS/TX processing to release all shared resources. For a complete discussion of DMS/TX protocol, refer to Chapter 3.

General Format

```
FREE ALL [ON ERROR imperative statement]
```

General Rules

1. FREE ALL releases from hold status all resources held by the program.
2. FREE ALL is the only FREE statement format used with DMS/TX files.
3. If there is an ON ERROR clause, any unsuccessful execution of these statements (with non-zero return code) causes the ERROR imperative statement to be executed.
4. If there is no ON ERROR clause, the user's program is cancelled upon unsuccessful execution of these statements.
5. If the ON ERROR clause exists, the special register RETURN-CODE contains the return code of the statement being executed.

GO TO Statement

Function

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

General Format

```
Format 1
    GO TO [procedure-name-1]
Format 2
    GO TO procedure-name-1 [procedure-name-2]... procedure-name-n
       DEPENDING ON identifier
```

Syntax Rules

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

2. 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.

3. If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules

1. When a GO TO statement, represented by Format 1 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.

2. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1, procedure-name-2,... , depending on the value of the identifier being 1, 2,... , n. If the value of the identifier is anything other than the positive integers 1, 2,... , n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

HOLD Statement

Function

The HOLD statement is used in the Procedure Division to hold files and/or a range of records within a file, or to claim the exclusive right to claim these resources.

General Format

```
HOLD [LIST] ⎧ RECORDS OF file-name-1                                                                        ⎫
            ⎪ ⎧                                                                                             ⎫ ⎪
            ⎪ ⎪   FOR ⎧RETRIEVAL⎫                                                                           ⎪ ⎪
            ⎨ ⎨       ⎩UPDATE   ⎭                                                                           ⎬ ⎬ ...
            ⎪ ⎪ ⎧FOR ⎧RETRIEVAL⎫WITH KEYS⎧[INITIAL ⎧data-name-1⎫CHARACTERS OF ] ⎧data-name-2⎫⎫ ... ⎫ ...   ⎪ ⎪
            ⎪ ⎪ ⎩    ⎩UPDATE   ⎭         ⎩         ⎩literal-1  ⎭                ⎩literal-2  ⎭⎭     ⎭       ⎪ ⎪
            ⎩ ⎩                                                                                             ⎭ ⎭

              [TIMEOUT OF ⎧data-name-3⎫ [SECOND ]]
              [           ⎩integer    ⎭ [SECONDS]]
                    [HOLDER-ID IN data-name-4]
                    ⎧imperative-statement⎫]
                    ⎩NEXT SENTENCE       ⎭]
```

General Rules

1. HOLD LIST builds a list of multiple files, and/or a range of records of an indexed file, to be held by a program. The LIST option specifies that the HOLD statement is part of a list of HOLD statements. Each HOLD statement in the list must have the LIST option except for the last executed HOLD statement in the list, which must not.

2. The last HOLD statement (without the LIST option) generates an attempt to hold the entire list of requested resources.

3. The TIMEOUT phrase cannot be used with the LIST option because the attempt to hold the resources is not made until the list is complete.

4. File-name-1 must be an indexed file.

5. Once specified, the hold class remains in effect until overridden by an explicit HOLD-CLASS phrase or a RECORDS OF phrase. The RECORDS OF phrase causes the current hold class to become UPDATE.

6. Data-name-1, if specified, must be a numeric data item and must not contain a value greater than the length of the shorter of data-name-2 or literal-2, and the primary key of file-name-1.

7. Data-name-2 may be any data item of usage DISPLAY. Literal-2 must be nonnumeric.

8. If the INITIAL phrase is not specified, the length of data-name-2 or literal-2 must not exceed the length of the primary key of file-name-1.

9. The TIMEOUT phrase is allowed with a HOLD statement that specifies more than one file name or record group.

10. If the record group is not specified, all records of the file are held. If the record group is specified, the records held are those with the following keys.

    a. If INITIAL is not specified, the value of data-name-2 or literal-2

    b. If INITIAL is specified, the first n characters of data-name-2 or literal-2, where n is the value of data-name-1 or literal-1.

11. The HOLD statement sets the FILE STATUS variable.

12. If the TIMEOUT phrase is specified, and the HOLD cannot be completed in data-name-4 or integer-1 seconds, then imperative-statement-1 is executed and control passes to the next sentence. If the number of seconds specified is zero, the timeout exit will immediately be taken if the HOLD cannot be completed. If the HOLDER-ID phrase is specified in the TIMEOUT phrase the logon initials of the user holding the resources are moved to data-name-5.

13. Files or groups of records are held for update unless RETRIEVAL is specified. Multiple groups may be specified for a file and both hold classes may be used within a file.

IF Statement

Function

The IF statement causes a condition (refer to Section 12.4, Conditions) to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

General Format

Format 1

```
IF condition  THEN  {statement-1   }
                    {NEXT SENTENCE } .
```

Format 2

```
IF condition  THEN  {statement-1   }
                    {NEXT SENTENCE }
              ELSE  {statement-2   }
                    {NEXT SENTENCE }
```

Format 3

```
                      {=              }
IF FAC OF data-name-1 {EQUAL TO       } data-name-2
                      {NOT EQUAL TO   }
                      {NOT =          }
     THEN  {statement-1   }
           {NEXT SENTENCE }
    [ELSE  {statement-2   }]
    [      {NEXT SENTENCE }]
```

Format 4

```
IF FAC OF data-name ALTERED THEN  {statement-1   }
                                  {NEXT SENTENCE }
                           [ELSE  {statement-2   }]
                           [      {NEXT SENTENCE }]
```

Format 5

```
IF figurative-constant {IN} {identifier             } IS [NOT]  {ON }
                       {OF} {FAC OF display-item    }           {OFF}
     THEN  {statement-1   }
           {NEXT SENTENCE }
    [ELSE  {statement-2   }]
    [      {NEXT SENTENCE }]
```

Syntax Rules

1. Statement-1 and statement-2 can be imperative or conditional statements, or sequences of such statements.

2. The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the terminal period of the sentence.

3. The FAC OF phrase can apply to a single display item or to a display item that OCCURS more than once.

4. Data-name-2 must be defined in the Data Division as a 1-byte item.

5. The figurative constant must be defined in the FIGURATIVE-CONSTANTS paragraph as a 1-byte item.

6. The identifier must reference a 1-byte item.

7. Display-item must be an item in a Working-Storage record whose USAGE IS DISPLAY-WS or in the record of a file for which the device type "DISPLAY" has been declared in the SELECT clause.

General Rules

1. When an IF statement is executed, the following transfers of control occur:

   a. If the condition is true, statement-1 is executed if specified. If statement-1 contains a procedure branching statement or conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-1 does not contain a procedure branching statement or conditional statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   b. 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.

   c. 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 statement or a conditional statement, control is explicitly transferred in accordance with the rules of that statement. If statement-2 does not contain a procedure branching statement or a 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.

   d. 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.

2. Statement-1 and/or statement-2 can contain an IF statement. In this case, the IF statement is said to be nested.

   IF statements within IF statements can 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.

3. When an IF FAC OF...ALTERED statement is executed, the compiler performs a test to determine if Bit 1 of the FAC is on (set to 1) or off. Bit 1 of a FAC is called its modified data tag (MDT). The MDT is set to 1 when and only when the field with which the FAC is associated is modified by the workstation operator. Following the MDT test, control is transferred as described in General Rules 1 and 2.

4. An IF figurative-constant statement allows the user to define a mask, via a figurative constant, that can be used to test a bit or combination of bits in a FAC or any other 1-byte item. The figurative constant is ON if for each bit set to 1 in the figurative constant, and for only those bits, the corresponding bit is set to 1 in the FAC or the item referenced by the identifier; otherwise the figurative constant is OFF. Following the test, control is transferred as described in General Rules 1 and 2.

INSPECT Statement

Function

The INSPECT statement provides the ability to TALLY (Format 1), REPLACE (Format 2), or TALLY and REPLACE (Format 3) occurrences of single characters in a data item.

General Format

Format 1

```
INSPECT identifier-1 TALLYING

  {                     { { {ALL    } {identifier-3} }                               }
  { identifier-2 FOR    { { {LEADING} {literal-1   } }                               }
  {                     { { CHARACTERS               }                               }
  {                     {                                                            }
  {                     { [ {BEFORE} INITIAL {identifier-4} ] } ...                  } ...
  {                     { [ {AFTER }         {literal-2   } ] }                      }
```

Format 2

```
INSPECT identifier-1 REPLACING

  { CHARACTERS BY {identifier-6} [ {BEFORE}   INITIAL   {identifier-7} ]                          }
  {               {literal-4   } [ {AFTER }             {literal-5   } ]                          }
  {                                                                                               }
  { { {ALL    } { {identifier-5} BY {identifier-6}                                                }
  { { {LEADING} { {literal-3   }    {literal-4   }                                                }
  { { {FIRST  } {                                                                                 }
  {             { [ {BEFORE} INITIAL {identifier-7} ] } ... } ...                                 }
  {             { [ {AFTER }         {literal-5   } ] }                                           }
```

Format 3

```
INSPECT identifier-1 TALLYING
 {                 {{{ALL     } {identifier-3}}
 {identifier-2 FOR {{{LEADING } {literal-1   }}
 {                 {{ CHARACTERS              }
 {                 {
 {  [{BEFORE} INITIAL {identifier-4}]}         } ...
 {  [{AFTER } INITIAL {literal-2   }]} ...     }
  REPLACING
 {CHARACTERS BY {identifier-6} [{BEFORE}   INITIAL   {identifier-7}]
 {              {literal-4   } [{AFTER }             {literal-5   }]
 {  {{ALL    } {{identifier-5}  BY {identifier-6}
 {  {{LEADING} {{literal-3   }     {literal-4   }
 {  {{FIRST  } {
 {
 {       [{BEFORE} INITIAL {identifier-7}]}      }     }
 {       [{AFTER }         {literal-5   }]} ...  } ... }
```

Syntax Rules

All Formats

1. Identifier-1 must reference either a group item or any category of elementary item, described (implicitly or explicitly) as usage is DISPLAY.

2. Identifier-3,... , identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described (implicitly or explicitly) as usage is DISPLAY.

3. Each literal must be nonnumeric and can be any figurative constant.

Formats 1 and 3 Only

4. Identifier-2 must reference an elementary numeric data item.

5. If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit 1-character data item.

Formats 2 and 3 Only

6. The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

7. When the CHARACTERS phrase is used, literal-4, literal-5, or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

8. When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

General Rules

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 are 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 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 (refer to 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.

3. In General Rules 4 through 11 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 operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1 and/or literal-3 is compared to an equal number of contiguous 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, character for character.

   b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

   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 rightmost character position that participated in 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 the first 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 1-character operand participates in the cycle described in General Rules 5a through 5d, 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 or 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 that involve that portion of the contents of the identifier-1 data item from its leftmost character position up to, but not including, the first occurrence of literal-2, literal-5. The position of this first occurrence is determined before the first cycle of the comparison operation 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 can participate only in the comparison cycles that involve those positions from the position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the identifier-1 data item to the rightmost character position of the identifier-1 data item. 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 are not initialized by the execution of the INSPECT statement, but must be initialized to 0 before each execution.

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 are incremented by one 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 identifier-2 data item are incremented by one for each contiguous occurrence of literal-1 matched within the contents of the identifier-1 data item, 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 are incremented by one 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 required words ALL, LEADING, and FIRST are adjectives.

10. 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 the adjective 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 the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the identifier-1 data item is replaced by literal-4, 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.

   d. When the adjective 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

11. A Format 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written: one statement being a Format 1 statement with TALLYING phrases identical to those specified in the Format 3 statement, 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 2 statement.

Examples of the INSPECT Statement

The following are six examples of the INSPECT statement:

```
INSPECT WORD TALLYING COUNTER-1 FOR LEADING "L" BEFORE INITIAL  "A"
    COUNTER-2 FOR LEADING "A" BEFORE INITIAL "L".

    Where WORD = LARGE,   COUNTER-1 = 1; COUNTER-2 = 0.
    Where WORD = ANALYST, COUNTER-1 = 0; COUNTER-2 = 1.

INSPECT WORD TALLYING COUNTER FOR ALL "L", REPLACING LEADING "A" BY
    "E"             AFTER INITIAL "L".

    Where WORD before = CALLAR, COUNTER = 2, WORD after = CALLAR.
    Where WORD before = SALAMI, COUNTER = 1, WORD after = SALEMI.
    Where WORD before = LATTER, COUNTER = 1, WORD after = LETTER.

INSPECT WORD REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

    Where WORD before = ARXAX,  WORD after = GRXAX.
    Where WORD before = HANDAX, WORD after = HGNDGX.

INSPECT WORD TALLYING COUNTER FOR CHARACTERS AFTER INITIAL "J"
    REPLACING           ALL "A" BY "B".

    Where WORD before = ADJECTIVE, COUNTER = 6, WORD after =
        BDJECTIVE.
    Where WORD before = JACK,      COUNTER = 3, WORD after = JBCK.
    Where WORD before = JUJMAB,    COUNTER = 5, WORD after = JUJMBB.

INSPECT WORD REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q"
    AFTER INITIAL "R".

    Where WORD before = RXXBQWY, WORD after = RYYZQQY.
    Where WORD before = YZACDWBR, WORD after = YZACDWZR.
    Where WORD before = RAWRXEB, WORD after = RAQRYEZ.

INSPECT WORD REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

    Where WORD before = 12 XZABCD,
          WORD after  = BBBBBABCD
```

MERGE Statement

Function

The MERGE statement combines two or more identically sequenced files on a set of user-specified keys. During the process, the statement makes the records available, in merge order, to an output procedure or to an output file.

General Format

```
MERGE file-name-1  ON { ASCENDING  } KEY data-name-1 [ data-name-2] ...
                      { DESCENDING }

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

   [COLLATING SEQUENCE IS alphabet-name]

   USING file-name-2 [file-name-3] ...

  { OUTPUT PROCEDURE IS section-name-1  [ { THROUGH } section-name-2 ] }
  {                                       { THRU    }                  }
  { GIVING file-name-4                                                 }
```

Syntax Rules

1. File-name-1 must be described in a sort-merge file description (SD) entry in the Data Division.

2. Section-name-1 represents the name of an output procedure.

3. File-name-2, file-name-3, and file-name-4 must be described in a file description entry, not in a sort-merge file description entry, of the Data Division. The actual size of the logical record(s) described for these file names must be equal to the actual size of the logical record(s) described for file-name-1.

   If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records in such a manner as to cause an equal number of character positions to be allocated for the corresponding records.

4. The words THRU and THROUGH are equivalent.

5. Data-name-1, data-name-2, data-name-3 and data-name-4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be described in records associated with file-name-1.

   b. KEY data-names may be qualified.

   c. If file-name-1 has more than one record description, the data items identified by KEY data-names need be described on only one of the record descriptions.

   d. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause or is subordinate to an entry which contains an OCCURS clause.

6. File-names must not be repeated within the MERGE statement.

General Rules

1. The MERGE statement will merge all records contained in file-name-2 and file-name-3. The files referenced in the MERGE statement must not be opened at the time the MERGE statement is executed. These files are automatically opened and closed by the merge operation. All implicit operations, such as the execution of any associated USE procedures, are automatically performed. The terminating function for all files is performed as if a CLOSE statement, without optional phrases, is executed for each file.

2. The data-names following the word KEY are listed from left to right in the MERGE statement. They are also listed in order of decreasing signifigance without regard to how they are divided in the KEY phrases. Data-name-1 is the major key, data-name-2 is the next most significant key, etc. The following rules also apply:

   a. When the ASCENDING phrase is specified, the merge sequence is from the lowest value of the contents of the data items identified by the KEY data-names, to the highest value. the rules for comparison of operands in a related condition apply.

   b. When the DESCENDING phrase is specified, the merge sequence is from the highest value of the contents of the items identified by the KEY data-names, to the lowest value. The rules for comparison of operands in a relation condition apply.

3. The COLLATING SEQUENCE IS phrase is treated as a comment.

4. The following rules apply to the MERGE output procedures:

   a. They must consist of one or more sections.

   b. Each procedure should contain at least one RETURN statement. (This makes the merged records available for processing.)

c. The procedure may consist of any procedures needed to select, modify, or copy the records being returned in merged order from file-name-1. The restrictions on any procedural statements within an output procedure are as follows:

   - Control can be transferred to points outside the procedure, but it is the programmer's responsibility to ensure a return to the procedure.

   - COBOL statements that cause an implied transfer of control to declaratives are allowed.

   - The procedures must not contain any SORT or MERGE statements.

   - The remainder of the Procedure Division may transfer control to points inside the output procedure by ALTER, GO TO, and PERFORM statements. If a MERGE statement is not being executed, such transfers must not cause the execution of a RETURN statement.

5. If an output procedure is specified, control passes to it during execution of the MERGE statement. The compiler inserts a return mechanism at the end of the last section of the output procedure. When control passes the last statement in the procedure, the return mechanism provides for termination of the merge. It then passes control to the next executable statement after the MERGE statement.

   Before entering the output procedure, the merge procedure reaches a point at which it can select the next record in merged order when requested. The RETURN statments in the output procedure are the requests for the next record.

6. If the GIVING phrase is specified, all merged records in file-name-1 are automatically written to file-name-4 (the implied output for this statement).

7. If, according to the rules for the comparison of operands in a relation condition, the contents of all the key data items of one data record are equal to the corresponding key data items of one or more other data records, the records are returned in the order of the associated input files as specified in the MERGE statement.

8. The results of the merge operation are predictable only when the records in the files referenced by file-name-2, file-name-3, ..., are ordered as described in the ASCENDING or DESCENDING KEY clause of the statement.

MOVE Statement

Function

The MOVE statement transfers data in accordance with the rules of editing to one or more data areas.

General Format

Format 1

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{literal} \end{matrix} \right\} \underline{\text{TO}} \text{ identifier-2 [identifier-3]...}$$

Format 2

```
MOVE WITH CONVERSION identifier-1 TO identifier-2
     [ON ERROR imperative-statement]
```

Format 3

```
MOVE figurative-constant TO FAC OF data-name
```

Format 4

```
MOVE FAC OF data-name-1 TO data-name-2
```

Format 5

```
MOVE data-name TO ORDER-AREA OF record-name
```

Format 6

```
MOVE ORDER-AREA OF record-name TO data-name
```

Format 7

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \text{identifier-1 } \underline{\text{TO}} \text{ identifier-2}$$

Syntax Rules

1. Identifier-1 and the literal represent the sending area; identifier-2, identifier-3, ... represent the receiving area.

2. An index data item cannot appear as an operand of a MOVE statement.

3. If a binary data item appears as an operand of a MOVE statement, all operands must be integer operands.

4. Format 2, MOVE WITH CONVERSION, cannot be used as an imperative statement.

5. In Format 2, identifier-2 must reference a numeric item whose usage is binary, computational, or display.

6. The FAC OF phrase can apply to a single display item or to a display item that OCCURS more than once.

7. The figurative constant must be defined in the FIGURATIVE-CONSTANTS paragraph as a 1-byte item.

8. Record-name must reference a Working-Storage record whose USAGE IS DISPLAY-WS or a record of a file for which the device type "DISPLAY" has been declared in the SELECT clause.

9. Data-name-1 must be defined in the Data Division as a 1-byte item.

10. Data-name-2 must be defined in the Data Division as a 4-byte group item.

11. CORR is an abbreviation for CORRESPONDING.

12. When the CORRESPONDING phrase is used, both identifiers must be group items.

General Rules

Format 1

1. The data designated by the literal 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)
   ```

   where "temp" is an intermediate item provided by the compiler.

2. 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, or 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 and user-figurative constants belong to the category alphanumeric.

   The following rules apply to an elementary move between categories:

   a. The figurative constant SPACE, user-figurative constants, a numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. 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.

   c. A noninteger numeric literal or a noninteger numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to General Rule 3.

3. Any necessary conversion of data from one form of internal representation to another, along with any editing specified for the receiving data item, takes place during legal elementary moves as follows:

   a. 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 (refer to Section 11.1.6). 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 is not moved; if the operational sign occupied a separate character position, that character is not moved, and the size of the sending item is considered to be one less than its actual size (in terms of standard data format characters).

   b. 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.

1) When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

2) 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.

3) 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.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the Standard Alignment Rules (refer to Section 11.1.6). 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.

4. 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 is filled without consideration for the individual elementary or group items contained within the sending or receiving area.

5. Data in the following chart summarizes the legality of the various types of MOVE statements. The general rule reference indicates the rule that prohibits the move or the behavior of a legal move.

Table 12-3. Permissible Moves Between Data Categories

CATEGORY OF SENDING DATA ITEM	CATEGORY OF RECEIVING DATA ITEM		
	ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	NUMERIC INTEGERO NUMERIC NONINTEGER NUMERIC EDITED
ALPHABETIC	Yes/3c	Yes/3a	No/2a
ALPHANUMERIC	Yes/3c	Yes/3a	Yes/3b
ALPHANUMERIC EDITED	Yes/3c	Yes/3a	No/2a
NUMERIC INTEGER	No/2b	Yes/3a	Yes/3b
NUMERIC NONINTEGER	No/2b	No/2c	Yes/3b
NUMERIC EDITED	No/2b	Yes/3a	No/2a

Format 2

6. The rules for formation of the data in identifier-1 are:

   a. The data can be either alphanumeric or numeric edited. It must not be over 16 characters in length.

   b. There can be any number of leading and trailing blanks.

   c. The digits of a number can either be immediately preceded or followed by a sign. The sign can be either the character '+' or '-'. If neither is specified, the sign of '+' is assumed.

   d. The number cannot have embedded blanks.

   e. The number can have one decimal point anywhere in the number. If none is present, it is assumed to be positioned to the right of the last digit. The value in identifier-2 is aligned by this explicit or implicit decimal point in the source.

   f. Other than a single optional decimal point, there cannot be any embedded characters in the number.

7. MOVE WITH CONVERSION converts character-strings containing a character representation of a number into a number that can be used for computation. The sending field can contain blanks, a sign and a decimal point. If there is an ON ERROR phrase, an attempt to move items that are not in conformance with the contents of identifier-1, as specified in General Rule 6, causes the ERROR imperative-statement to be executed. If the data to be moved is valid, but there is either right or left truncation of the digit positions in moving it into the receiving field, the value is truncated and moved to identifier-2, and the ON ERROR imperative-statement is executed.

8. 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.

Formats 3 to 6

9. For a single display item, the FAC OF phrase can be processed simply by coding FAC OF display-item. For a display item which OCCURS, the FAC OF display-item is processed with subscripts equal to the dimensions of the display-item in the OCCURS clause.

10. Refer to Appendix C, Field Attribute Characters, and Appendix D, Workstation Screen Order Area, for the hexadecimal characters used for FACS and for order area control, respectively.

NOTE

The programmer is reminded that FAC and ORDER AREA values are hexadecimal characters. Care must be taken that they are not confused with other data types in MOVE statements.

Format 7

1. When the CORRESPONDING Phrase is used, selected items within identifier-1 are moved to selected items within identifier-2, according to the rules given in the CORRESPONDING Phrase. The results are the same as if the programmer referenced each pair of corresponding items in separate MOVE statements.

MULTIPLY Statement

Function

The MULTIPLY statement causes numeric data items to be multiplied and sets the values of data items equal to the results.

General Format

Format 1

```
MULTIPLY {identifier-1}  BY identifier-2 [ROUNDED]
         {literal-1   }
      [ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY {identifier-1}  BY {identifier-2}  GIVING identifier-3 [ROUNDED]
         {literal-1   }     {literal-2   }
      [ON SIZE ERROR imperative-statement]
```

Syntax Rules

1. 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.

2. Each literal must be a numeric literal.

General Rules

1. When Format 1 is used, 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 the product.

2. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3.

3. Refer to "ROUNDED Phrase", "SIZE ERROR Phrase", and "Overlapping Operands" in Section 11.3.3, Arithmetic Statements.

Example of MULTIPLY Statement

```
MULTIPLY QTY-IN-STOCK BY TOT-INVEST ROUNDED
    ON SIZE ERROR GO TO EXIT-RTN.
```

If TOT-INVEST is defined with a PICTURE of S9(4) and the values associated with QTY-IN-STOCK and TOT-INVEST are 426 and 2.7 respectively, the result 1150.2 is rounded to 1150 and placed in TOT-INVEST.

OPEN Statement -- for Consecutive Files

Function

The OPEN statement initiates the processing of files.

General Format

```
      ⎧INPUT   file-name-1   [file-name-6] ...⎫
      ⎪OUTPUT  file-name-2   [file-name-7] ...⎪
OPEN  ⎨I-O     file-name-3   [file-name-8] ...⎬ ...
      ⎪SHARED  file-name-4   [file-name-9] ...⎪
      ⎩EXTEND  file-name-5   [file-name-10]...⎭
```

Syntax Rules

1. The SHARED phrase can be used only for mass storage files. The I-O phrase can be used only for mass storage files and workstation files.

2. The files referenced in the OPEN Statement need not all have the same organization or access.

3. The EXTEND phrase must not be specified for multiple file reels.

4. A workstation file can only be opened in the I-O or shared mode.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. 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.

3. An OPEN statement must be successfully executed prior to the execution of any of the permissible input/output statements. In Table 12-4, Permissible Statements -- Consecutive Files, 'X' at an intersection indicates that the specified statement can be used with the access mode given on the left and the open mode given at the top of the column.

Table 12-4. Permissible Statements -- Consecutive Files

File Access Mode	Statement	Open Mode				
		INPUT	OUTPUT	INPUT-OUTPUT	SHARED	EXTEND
SEQUENTIAL	READ	X		X		
	WRITE		X			X
	REWRITE			X		
	START*			X		
RANDOM	READ	X		X		
	WRITE					
	REWRITE			X		
DYNAMIC	READ	X		X		
	WRITE		X		X	X
	REWRITE			X		

* Workstation files only

4. A file can be opened with the INPUT, OUTPUT, EXTEND, I-O and SHARED phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent execution of an OPEN statement for that same file must be preceded by the execution of a CLOSE statement for that file without the REEL, UNIT, or LOCK phrase.

5. Execution of the OPEN statement does not obtain or release the first data record.

6. If label records are present and the INPUT or OUTPUT phrase is specified, execution of the OPEN statement causes the beginning labels to be checked or written in accordance with the conventions for label processing. Refer to Chapter Six, Tape File Processing, and to "LABEL RECORDS Clause" in Section 11.3.1.

7. The file description entry for existing files must be equivalent to that used when the file was created.

8. If the storage medium for the file permits rewinding, and the EXTEND phrase is not specified, execution of the OPEN statement causes the file to be positioned at its beginning.

9. For files being opened with the INPUT, or I-O phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file results in an AT END condition.

10. 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 until the mass storage file exists.

11. When the I-O phrase is specified and label records are present, the execution of the OPEN statement includes the following steps:

    a. The labels are checked in accordance with the conventions for Input/Output label checking.

    b. The new labels are written in accordance with the conventions for Input/Output label writing.

    Refer to Chapter 6, Tape File Processing, and to "LABEL RECORDS Clause" in Section 11.3.1.

12. 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.

13. Extend mode cannot be used unless the file already exists. After successful execution of an OPEN EXTEND, the current record pointer is positioned immediately following the last logical record of that file. Subsequent WRITE statements referencing the file add records to the file as though the file had been opened with the OUTPUT phrase.

14. The SHARED phrase permits the file to be written to by several users concurrently. A consecutive file opened in shared mode must be specified, in the RECORD CONTAINS clause of the File Description entry, as containing variable-length records.

15. The only valid function request for a consecutive file opened in shared mode is WRITE, which appends a record to the end of the file.

16. If the specified file does not exist when opened in shared mode, it is created, and its label is set to indicate that the file is a "logging file" (i.e., open for shared access). If the first character of the file name is an ASCII "at" sign ('@'), an additional flag is set in the file label indicating that each record is to be written immediately to disk when received (thus further securing the file against lost updates).

17. A shared mode OPEN on an existing file is allowed only if the file is a designated "logging file" (i.e., the file was created in shared mode).

OPEN Statement -- for Indexed Files

Function

The OPEN statement initiates the processing of files.

General Format

$$\underline{\text{OPEN}}\left\{\begin{array}{lll}\underline{\text{INPUT}} & \text{file-name-1} & [\text{file-name-5}]\ldots \\ \underline{\text{OUTPUT}} & \text{file-name-2} & [\text{file-name-6}]\ldots \\ \underline{\text{I-O}} & \text{file-name-3} & [\text{file-name-7}]\ldots \\ \underline{\text{SHARED}} & \text{file-name-4} & [\text{file-name-8}]\ldots\end{array}\right\}\ldots$$

Syntax Rules

1. The files referenced in the OPEN statement need not all have the same organization or access.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. 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.

3. An OPEN statement must be successfully executed prior to the execution of any of the permissible input/output statements. In Table 12-5, Permissible Statements -- Indexed Files, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, can be used with the indexed file organization and the open mode given at the top of the column.

4. A file can be opened with the INPUT, OUTPUT, I-O and SHARED phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for the same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

5. Execution of the OPEN statement does not obtain or release the first data record.

6. The file description entry for input, I-O, or shared files must be equivalent to that used when this file was created.

Table 12-5. Permissible Statements -- Indexed Files

File Access Mode	Statement	Open Mode			
		INPUT	OUTPUT	INPUT-OUTPUT	SHARED
SEQUENTIAL	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	
RANDOM	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
DYNAMIC	READ	X		X	X
	WRITE		X	X	X
	REWRITE			X	X
	START	X		X	X
	DELETE			X	X

7. For files being opened with the INPUT, I-O, or SHARED phrases, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the record key is used to determine the first record to be accessed. 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 results in an AT END condition.

8. The I-O and SHARED phrases permit the opening of a file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the file is being initially created.

9. 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.

10. The shared mode OPEN permits two or more users to modify the same file concurrently. There is no limit to the number of users who can share the same file. A shared mode OPEN is valid only for indexed files whose access is DYNAMIC.

OPEN Statement -- for Relative Files

Function

The OPEN statement initiates the processing of files. It also performs checking and/or writing of labels and other input-output operations.

General Format

$$\underline{\text{OPEN}} \left\{ \begin{array}{ll} \underline{\text{INPUT}} & \{\text{file-name-1}\}\ldots \\ \underline{\text{OUTPUT}} & \{\text{file-name-2}\}\ldots \\ \underline{\text{I-O}} & \{\text{file-name-3}\}\ldots \\ \underline{\text{EXTEND}} & \{\text{file-name-4}\}\ldots \end{array} \right\} \ldots$$

Syntax Rules

1. The files referenced in the OPEN statement need not all have the same organization or access.

General Rules

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. 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.

3. An OPEN statement must be successfully executed prior to the execution of any of the permissible input/output statements. In Table 12-6, Permissible Statements -- Relative Files, 'X' at an intersection indicates that the specified statement, used in the access mode given for that row, can be used with the relative file organization and the open mode given at the top of the column.

4. A file can be opened with the EXTEND, 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 the same file must be preceded by the execution of a CLOSE statement, without the LOCK phrase, for that file.

5. Execution of the OPEN statement does not obtain or release the first data record.

Table 12-6. Permissible Statements -- Relative Files

File Access Mode	Statement	Open Mode			
		INPUT	OUTPUT	INPUT-OUTPUT	EXTEND
SEQUENTIAL	READ	X		X	
	WRITE		X		X
	REWRITE			X	
	START	X		X	
	DELETE			X	
RANDOM	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START				
	DELETE			X	
DYNAMIC	READ	X		X	
	WRITE		X	X	
	REWRITE			X	
	START	X		X	
	DELETE			X	

6. Upon execution, The OPEN statement causes the operating system to check label records, if specified.

7. The file description entry for input or I-O files must be equivalent to that used when the file was created.

8. For files being opened with the INPUT or I-O phrases, 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 results in an AT END condition.

9. The I-O phrase permits the opening of a file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the file is being initially created.

10. If the I-O phrase is specified, execution of the OPEN statement causes the operating system to check the existing labels and then write new ones.

11. 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.

12. OPEN EXTEND is only allowed for relative files with the ACCESS IS SEQUENTIAL phrase specified. The only operation allowed for such a file is a WRITE.

13. Extend mode cannot be used unless the file already exists. After successful execution of an OPEN EXTEND, the current record pointer is positioned immediately following the last logical record of that file. Subsequent WRITE statements referencing the file add records to the file as though the file had been opened with the OUTPUT phrase.

PERFORM Statement

Function

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.

General Format

Format 1

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{procedure-name-2}\right]$$

Format 2

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{procedure-name-2}\right]\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\}\ \underline{\text{TIMES}}$$

Format 3

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\ \left[\left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{procedure-name-2}\right]\ \underline{\text{UNTIL}}\ \text{condition-1}$$

Format 4

$$\underline{\text{PERFORM}}\ \text{procedure-name-1}\left[\left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{procedure-name-2}\right]$$

$$\underline{\text{VARYING}}\ \left\{\begin{array}{l}\text{identifier-2}\\ \text{index-name-1}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\text{identifier-3}\\ \text{index-name-2}\\ \text{literal-1}\end{array}\right\}$$

$$\underline{\text{BY}}\ \left\{\begin{array}{l}\text{identifier-4}\\ \text{literal-2}\end{array}\right\}\ \ \underline{\text{UNTIL}}\ \text{condition-1}$$

Syntax Rules

1. Each identifier represents an elementary numeric item described in the Data Division. Identifier-1 must represent an integer.

2. Each literal represents a numeric literal.

3. The words THRU and THROUGH are equivalent.

4. If an index name is specified in the VARYING phrase, then:

   a. The identifier in the associated FROM and BY phrases must be an integer item.

   b. The literal in the associated FROM phrase must be a positive integer.

   c. The literal in the associated BY phrase must be a nonzero integer.

5. If an index name is specified in the FROM phrase, then:

   a. The identifier in the associated VARYING phrase must be an integer data item.

   b. The identifier in the associated BY phrase must be an integer data item.

   c. The literal in the associated BY phrase must be an integer.

6. The literal in the BY phrase must not be a nonzero integer.

7. Condition-1 can be any conditional expression.

8. Where procedure-name-1 and procedure-name-2 are both specified and either is the name of a procedure in the Declaratives Section of the program then both must be procedure names in the same Declaratives Section.

General Rules

1. The data item referenced by identifier-4 must not have a zero value.

2. If an index name is specified in the VARYING phrase, and an identifier is specified in the associated FROM phrase, then the data item referenced by the identifier must have a positive value.

3. When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1 (except as indicated in General Rules 6b, 6c, and 6d). This transfer of control occurs only once for each execution of a PERFORM statement. For those cases where 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:

   a. 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.

   b. 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.

   c. If procedure-name-2 is specified and it is a paragraph-name then the return is after the last statement of the paragraph.

   d. 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.

4. 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 can 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 can be the name of a paragraph consisting of the EXIT statement, to which all of these paths must lead.

5. 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.

6. The PERFORM statements operate as follows with General Rule 5 applying to all formats:

   a. Format 1 is the basic PERFORM statement. A procedure referenced by this type of PERFORM statement is executed once. Then control passes to the next executable statement following the PERFORM statement.

   b. Format 2 is the PERFORM...TIMES. The procedures are performed the number of times specified by integer-1 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 the number indicated by the initial value of identifier-1.

   c. 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.

d. 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 and FROM (current value) phrases also refers to index-names. When index-name appears in a VARYING phrase, it is initialized and subsequently augmented according to the rules of the SET statement. If index-name appears in the FROM phrase, the identifier, when it appears in an associated VARYING phrase, is initialized according to the rules of the SET statement; subsequent augmentation is as described in the next paragraph.

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. Figure 12-1 is a flowchart for the VARYING phrase of the PERFORM statement.

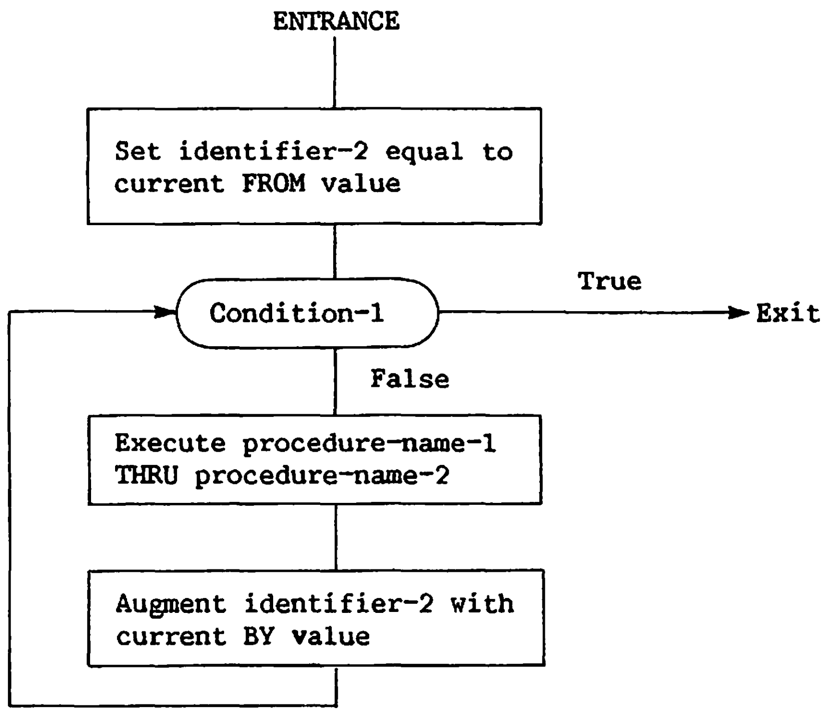
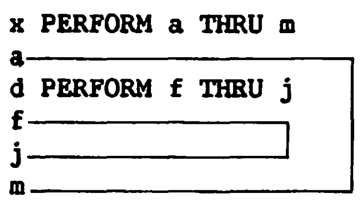

ENTRANCE
Set identifier-2 equal to current FROM value
Condition-1
True
Exit
False
Execute procedure-name-1 THRU procedure-name-2
Augment identifier-2 with current BY value

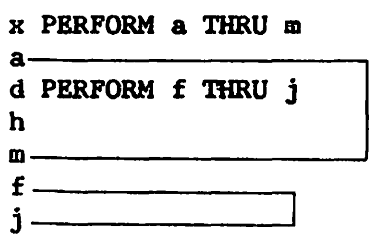
Figure 12-1. Flowchart for the VARYING Phrase of a PERFORM Statement

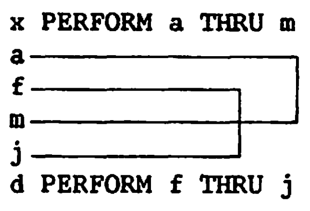
7. The range of a PERFORM statement consists logically of all those statements that are executed as a result of executing the PERFORM statement through execution of the implicit transfer of control the the end of the PERFORM statement. The range includes all statements that are executed as the result of a transfer of control by CALL, EXIT, GO TO, and PERFORM statements in the range of the PERFORM statement, as well as all statements in declarative procedures that are executed as a result of the execution of statements in the range of the PERFORM statement. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

8. Statements executed as the result of a transfer of control caused by executing an EXIT PROGRAM statement within the range of a PERFORM statement are not considered to be part of that range.

9. If the range of 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. The following illustrations show the valid sequences:







Example of PERFORM Statements To Initialize a 3 by 6 Table

```
       PROCEDURE DIVISION.
       INITIALIZE-THE-TABLE.
           PERFORM INIT1 VARYING INDX1 FROM 1 BY 1 UNTIL INDX1 > 3.
           DISPLAY "THE TABLE IS INITIALIZED.".
           STOP RUN.

       INIT1.
           PERFORM INIT-TABLE VARYING INDX2 FROM 1 BY 1 UNTIL INDX2 > 6.

       INIT-TABLE.
      *
      *    Initialize each table element here.
      *
```

READ Statement -- for Consecutive Files

Function

For SEQUENTIAL access, the READ statement makes available the next logical record from a file. For RANDOM access, the READ statement makes available a specified record from a mass storage file.

General Format

Format 1

```
READ file-name [NEXT] RECORD [WITH HOLD] [INTO identifier]
     [AT END imperative-statement]
```

Format 2

```
                                 ┌WITH HOLD ┐
READ file-name [NEXT] RECORD     │MODIFIABLE│  [INTO identifier]
                                 └ALTERED   ┘

   ┌┌INVALID KEY┐                      ┐
   ││           │ imperative-statement │
   └└AT END     ┘                      ┘
```

Syntax Rules

1. Format 1 must be used for all files in SEQUENTIAL access mode.

2. The NEXT phrase must be specified for files in DYNAMIC access mode, when records are to be retrieved sequentially.

3. Format 2 is used for files in RANDOM access mode or for files in DYNAMIC access mode when records are to be retrieved randomly.

4. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

5. The AT END phrase can be specified in a Format 2 READ statement only when ALTERED has been specified.

6. The storage area associated with identifier and the record area associated with file-name must not be the same.

General Rules

1. The associated files must be open in the input or I-O mode at the time this statement is executed.

2. The following rules pertain to files read sequentially (Format 1).

a. The record pointed to by the current record pointer is made available provided that the current record pointer was positioned by the OPEN statement and the record is still accessible through the path indicated by the current record pointer.

b. 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.

c. 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.

d. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, an AT END condition occurs, and the execution of the READ statement is unsuccessful.

e. If the RELATIVE KEY phrase is specified in the file control entry, 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.

3. For files to be read randomly (Format 2) the execution of a 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. The RELATIVE KEY phrase must be included in the FILE CONTROL entry for sequential files that are to be read randomly.

4. The execution of the READ statement causes the value of the FILE STATUS data item, if described in the ASSIGN clause of the file control entry, to be updated.

5. When the logical records of a file are described with more than one record description in the File Description (FD) entry, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

6. 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 without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

7. 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.

8. When the AT END condition is recognized the following actions are taken in the specified order:

   a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

   b. 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.

   c. 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 that caused the condition is unsuccessful.

9. 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.

10. 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. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful Format 2 READ statement for that file.

11. 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 as described in General Rule 2.

12. Except in the case of workstation files, the WITH HOLD phrase must be specified in a READ statement that precedes a REWRITE for a file in DYNAMIC or SEQUENTIAL access mode.

13. The WITH HOLD phrase is only valid for files opened in I-O mode.

Workstation Screen I/O

14. The execution of the READ statement for a workstation file causes the relative record number (the data item named in the RELATIVE KEY Clause) to be moved to the first byte of the order area, and an attempt is made to read the workstation data to the user's "mapping area".

    A record of the workstation file contains the 4-byte order area (refer to Section 4.3.4) followed by a variable number (up to 1920) of bytes containing the data transmitted either to or from the screen. The latter is the mapping area. (Refer to Appendix D.)

15. After I/O completion, the following information is available:

    a. The record is available in the mapping area.

    b. The cursor position data item and order area Bytes 2 and 3 receive the current cursor position.

    c. The second byte of the FILE STATUS key receives the AID character (refer to Section E.6, AID Characters).

16. If MODIFIABLE is specified, the file must be a workstation file. Specifying MODIFIABLE causes the following actions to occur:

    a. The contents of modifiable locations of the workstation screen are moved to the mapping area.

    b. Pseudo-blank characters within the range of the READ operation are changed to blanks before transmission; therefore, they are also represented as blanks in the mapping area.

    c. Blinking characters within the range of the READ operation are changed to high-intensity nonblinking characters (by changing the associated Field Attribute Character both on the screen and in the mapping area).

    d. The cursor position data item and Bytes 2 and 3 in the order area receive the current cursor position.

    e. The second byte of the FILE STATUS key receives the AID character (refer to Section E.6, AID Characters).

17. If the ALTERED modifier is specified, the file must be a workstation file. When ALTERED is specified, only those fields that have been modified by the user are moved. If AT END imperative-statement is specified with ALTERED and no field has been changed, the imperative statement is executed.

NOTE

The recommended options for the READ I/O Statement are MODIFIABLE and INVALID KEY.

READ Statement -- for Indexed Files

Function

For SEQUENTIAL access, the READ statement makes available the next logical record from a file. For RANDOM access, the READ statement makes available a specified record from a mass storage file.

General Format

```
Format 1
    READ file-name [NEXT] RECORD  [WITH HOLD]  [INTO identifier]
      [TIMEOUT OF {data-name-1}  [SECOND ]
      [           {integer    }  [SECONDS]
      [     [HOLDER-ID IN data-name-2]
      [      {imperative-statement}]
      [      {NEXT SENTENCE       }]
      [AT END imperative-statement]
Format 2
    READ file-name RECORD  [WITH HOLD]  [INTO identifier]
        [KEY IS data-name-3]
      [TIMEOUT OF {data-name-1}  [SECOND ]
      [           {integer    }  [SECONDS]
      [     [HOLDER-ID IN data-name-2]
      [      {imperative-statement}]
      [      {NEXT SENTENCE       }]
        [INVALID KEY imperative-statement]
```

Syntax Rules

1. The storage area associated with INTO identifier and the storage area that is the record area associated with file-name must not be the same storage area.

2. Format 1 must be used for all files in SEQUENTIAL access mode.

3. The NEXT phrase must be specified for files in DYNAMIC access mode when records are to be retrieved sequentially.

4. Format 2 is used for files in RANDOM access mode or for files in DYNAMIC access mode when records are to be retrieved randomly.

5. The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

6. Data-name-1 must refer to an integer item no greater than 255.

7. Data-name-2 must be defined in the Working-Storage Section or Linkage Section and have a PICTURE of X(3).

8. Data-name-3 must be the name of a data item specified as an alternate record key associated with file-name. Data-name-3 may be qualified.

9. The TIMEOUT phrase can be specified only if the HOLD phrase is specified.

General Rules

1. The associated file must be open in the input, I-O, or shared mode at the time this statement is executed.

2. The record to be made available by the Format 1 READ statement is determined as follows:

   a. 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 or a change in an alternate record key, the current record pointer is updated to point to the next existing record within the established key of reference, and that record is then made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference, and then that record is made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

4. When the logical records of a file are described with more than one record description in the File Description (FD) entry, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. The contents of any data items that lie beyond the range of the current data record are undefined at the completion of the execution of the READ statement.

5. For a file open in shared mode or I-O mode for which the access mode is DYNAMIC, a READ statement containing the WITH HOLD phrase must precede any DELETE or REWRITE operation.

6. For a file open in shared mode, the WITH HOLD phrase is used to hold a record for update. Following successful execution of a READ WITH HOLD, the record read can be accessed by other users sharing the same file, but it cannot be modified by anyone else until it is released by the READ issuer. If the issuer is holding the record read or the file containing it, the issuer continues to hold it. If another record (in the same or any other file) is being held, that record is released. If the specified record is being held by another user, the READ issuer is forced to wait until it is released by that user. If the READ issuer is holding a file other than the one on which the READ WITH HOLD is issued, an error message is displayed and the READ is not performed. A record held for update can be released in three ways:

   a. By executing any DELETE, REWRITE or WRITE statement

   b. By executing a succeeding READ WITH HOLD on the same or any other file

   c. By executing a CLOSE for the file containing the held record, including an implied CLOSE on run termination.

7. 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 without the CORRESPONDING phrase. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

8. 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.

9. 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.

10. If, at the time of the execution of a Format 1 READ statement, no next logical record is accessible through the established key of reference, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

11. When the AT END condition is recognized the following actions are taken in the specified order:

    a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

b. 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.

c. 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 that caused the condition is unsuccessful.

12. 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.

13. 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. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

    b. A successful START statement for that file.

    c. A successful Format 2 READ statement for that file.

14. For a file for which DYNAMIC access mode is specified, a Format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in General Rule 2.

15. For an indexed file being SEQUENTIALLY accessed, records having the same duplicate value in an alternate record key that is the key of reference are made available in ascending primary key order.

16. 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.

17. If the KEY phrase is not specified in a Format 2 READ statement, the primary record key is established as the key of reference for this retrieval. If the DYNAMIC access mode is specified, this key of reference is also used for retrievals by any subsequent executions of Format 1 READ statements for the file until a different key of reference is established.

18. Execution of a Format 2 READ statement causes the value of the key to be compared with the value contained in the corresponding data item of the stored records in the file, until the record having an equal value is found. The current record pointer is positioned to this record which is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

19. If the TIMEOUT phrase is specified and the READ cannot be completed in data-name-2 or integer seconds, then imperative-statement-1 is executed. If the number of seconds specified is 0, the timeout exit will immediately be taken if the READ cannot be completed.

20. If the HOLDER-ID phrase is specified in the TIMEOUT phrase, the logon initials of the user currently holding the resources is moved to data-name-3.

READ Statement -- for Relative Files

Function

The READ statement makes available the next logical record of a file when access is sequential. For random access, the READ statement makes available a specific record from a mass storage unit.

General Format

Format 1

READ file-name [NEXT] RECORD [WITH HOLD] [INTO identifier]
    [AT END imperative-statement]

Format 2

READ file-name RECORD [WITH HOLD] [INTO identifier]
    [INVALID KEY imperative-statement]

Syntax Rules

1. The storage area associated with identifier and the record area associated with file-name must not be the same storage area.

2. Format 1 must be used for all files in sequential access mode.

3. The NEXT phrase must be specified for files in dynamic access mode when records are to be retrieved sequentially.

4. Format 2 is used for files in either random access mode or for files in dynamic access mode when records are to be retrieved randomly.

5. Either the INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

6. Relative files do not require the WITH HOLD phrase with a READ statement prior to a WRITE, REWRITE, or DELETE operation. Therefore, the WITH HOLD phrase is treated as a comment by the compiler.

General Rules

1. The associated files must be opened in either INPUT or I-O mode at the time the READ statement is executed.

2. Records are made available by the Format 1 READ as follows:

   a. The record pointed to by the current record pointer is made available if the pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the pointer. If the record is no longer accessible (i.e., it was deleted), the current record pointer is updated to point to the next existing record in the file. That record is then made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the pointer is updated to point to the next existing record in the file. That record is then made available.

3. The execution of the READ statement causes the value of the FILE STATUS data item (if any) associated with file-name, to be updated.

4. The concept of the READ statement remains unchanged regardless of the method used to overlap access time with processing time. This means that a record is available to the object program prior to the execution of any statement following the READ statement.

5. Logical records share the same storage area if they are described with more than one record description. 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.

6. A record being read is moved from the record area to the area specified by identifier if the INTO phrase is specified. This move follows the rules for a MOVE statement without the CORRESPONDING phrase. The implied move does not occur if the execution of the READ statement is unsuccessful. Any subscripting or indexing associated with identifier is evaluated after the record is read and immediately before it is moved.

7. A record being read is accessable from both the input record area and data area associated with identifier when the INTO phrase is used.

8. The execution of a READ statement is unsuccessful if, at the time of execution of a Format 1 statement, the position of the current record pointer for that file is undefined.

9. If no next logical record exists in a file at the time of execution of a Format 1 READ statement, the AT END condition occurs and the read is considered unsuccessful.

10. When the AT END condition occurs, the following actions are taken in the specified order:

    a. The AT END condition is indicated by the placement of a value in the FILE STATUS data item, if one was specified for the file.

    b. Control is transferred to the AT END imperative-statement if the AT END phrase is specified in the statement causing the condition. Any USE procedure specified for the file is not executed.

    c. A USE procedure must be specified if an AT END phrase is not. The USE procedure can be either implicitly or explicitly specified.

    Execution of an input-output statement that causes an AT END condition to occur is considered unsuccessful.

11. The contents of an associated record area and the position of the current record pointer are undefined following the unsuccessful execution of a READ statement.

12. When an 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. A successful CLOSE statement followed by the execution of a successful OPEN statement.

    b. A successful START statement.

    c. A successful Format 2 READ statement.

13. A Format 1 READ statement with the NEXT phrase causes the next logical record to be retrieved from a file with dynamic access mode specified (refer to General Rule 2).

14. 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.

15. 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. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

RELEASE Statement

Function

The RELEASE statement transfers records to the initial phase of a SORT operation.

General Format

```
RELEASE record-name [FROM identifier]
```

Syntax Rules

1. Record-name must be the name of a logical record in a sort-merge file description entry and it may be qualified.

2. A RELEASE statement may only be used within the range of an input procedure that is associated with a SORT statement for a file whose SD file description entry contains record-name.

3. Record-name and identifier must not refer to the same storage area.

General Rules

1. The execution of a RELEASE statement causes the record named by record-name to be released to the initial phase of a sort operation.

2. When the FROM phrase is used, the contents of the identifier data area are moved to record-name. The contents of record-name are then released to the sort file. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. After the move, the information in the record area is no longer available, but the information in the data area associated with the identifier is still available.

3. When control passes from the input procedure, the file consists of all those records which were placed in it by the execution of RELEASE statements.

RETURN Statement

Function

The RETURN statement obtains either sorted records from the final phase of a SORT operation, or merged records during a MERGE operation.

General Format

```
RETURN file-name RECORD [INTO identifier] AT END imperative-statement
```

Syntax Rules

1. File-name must be described by an SD file description entry in the Data Division.

2. A RETURN statement may only be used within the range of an output procedure associated with a SORT or MERGE statement for file-name.

3. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage areas associated with identifier, and the record area associated with file-name, must not be the same storage area.

General Rules

1. 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 RETURN statement.

2. The execution of the RETURN statement causes the next record to be made available for processing. The order of records is that specified by the keys listed in the SORT or MERGE statement. The record is made available in the record areas associated with the SORT or MERGE file.

3. If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. The implied MOVE does not occur if there is an AT END condition. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to the data item.

4. If the INTO phrase is specified, the data is available in both the input record area and the data area associated with identifier.

5. If no next logical record exists for the file at the time of the execution of a RETURN statement, the AT END condition occurs. The contents of the record areas associated with the file when this happens is undefined. After the execution of the imperative-statement in the AT END phrase, no RETURN statement may be executed as part of the current output procedure.

REWRITE Statement -- for Consecutive Files

Function

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format

```
Format 1
    REWRITE record-name [FROM identifier]
Format 2
    REWRITE record-name [FROM identifier] AFTER
     ⎧ ALARM                                                            ⎫
     ⎪ SETTING CURSOR COLUMN ⎧identifier-1⎫ ⎧ROW ⎫ ⎧identifier-2⎫      ⎪
     ⎪                       ⎩integer-1   ⎭ ⎩LINE⎭ ⎩integer-2   ⎭      ⎪
     ⎨ ROLL DOWN                                                        ⎬
     ⎪ ROLL UP                                                          ⎪
     ⎪ ERASE PROTECT                                                    ⎪
     ⎩ ERASE MODIFY                                                     ⎭
```

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. Record-name is the name of a logical record in the file section of the Data Division and may be qualified.

General Rules

Formats 1 and 2

1. For a mass storage file, the last input/output statement executed for the associated I-O mode file prior to the execution of the REWRITE statement must have been a successfully executed READ WITH HOLD statement. The record accessed by the READ statement is logically replaced by REWRITE. If the file is a workstation file, there is no requirement of a previous READ or a HOLD.

2. 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.

3. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

4. The file associated with record-name must be a mass storage file open in the I-O mode, or a workstation file open in I-O mode, at the time of execution of this statement.

5. The file associated with record-name must not be a compressed file. It must not contain variable-length records. Records must be defined as fixed-length.

6. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. If the file is so named, the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

7. The execution of a rewrite statement with the FROM phrase is equivalent to the execution of the statement,

   <u>MOVE</u> identifier <u>TO</u> 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.

8. The current record pointer is not affected by the execution of a REWRITE statement.

Format 2 Only

9. Format 2 is used for records of the workstation file.

10. Integer-1 can be any number from 1 through 80.

11. The function selected by the AFTER phrase specifies the Write Control Character (WCC) to be used on the REWRITE operation.

Function	WCC
ALARM	X"C0"
CURSOR	X"A0"
DOWN	X"90"
UP	X"88"
PROTECT	X"82"
MODIFY	X"84"

12. Execution of a Format 2 READ statement causes the relative record number (the data item named in the RELATIVE KEY phrase) to be moved to the first byte of the order area, and an attempt is made to write to the workstation screen in accordance with actions directed by the Write Control Character.

13. The display item referenced by record-name may be only one line for the following functions:

    ROLL DOWN
    ROLL UP

14. Integer-2 can be any number from 1 through 24.

15. If ROW (or LINE) is not specified, the line is identified by the relative key for the record being rewritten.

REWRITE Statement -- for Indexed Files

Function

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format

```
REWRITE record-name [FROM identifier]
    [INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

General Rules

1. The file associated with record-name must be open in the I-O or shared mode at the time of execution of this statement.

2. For files in the RANDOM access mode, the last input/output statement executed for the associated file prior to REWRITE must have been a successfully executed READ statement. The record accessed by the READ statement is logically replaced by REWRITE.

3. For files in the DYNAMIC or 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 WITH HOLD statement. The record accessed by the READ WITH HOLD statement is logically replaced by REWRITE.

4. For files open in shared 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 WITH HOLD, with no intervening READ WITH HOLD on any other shared file.

5. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause. If the file is so named, the logical record is available to the program as a record of other files appearing in the same SAME RECORD AREA clause as the associated I-O file, as well as to the file associated with record-name.

6. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of the statement,

   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.

7. The current record pointer is not affected by the execution of a REWRITE statement.

8. The execution of the REWRITE statement causes the value of the FILE STATUS data item in the file control entry to be updated.

9. For a file in the SEQUENTIAL access mode, the record to be replaced is specified by the value contained in the record key. When the REWRITE statement is executed the value contained in the record key data item of the record to be replaced must be equal to the value of the record key of the last record read from this file.

10. For a file in the RANDOM or DYNAMIC access mode, the record to be replaced is specified by the record key data item.

11. If record-name references a variable-length COMPRESSED record, the rewritten record can have a different length.

12. The INVALID KEY condition exists when one of the following occurs.

    a. The access mode is SEQUENTIAL and the value contained in the record key data item of the record to be replaced is not equal to the value of the record key of the last record read from this file.

    b. The value contained in the record key data item does not equal that of any record stored in the file.

    c. The value contained in an alternate record key data item for which a DUPLICATES phrase has not been specified is equal to that of a record already stored in the file.

    The updating operation does not take place and the data in the record area is unaffected.

13. The alternate access paths (indices) through which the rewritten record is to be available is determined by the alternate keys associated with the specified record-name. When rewritten, the record is deleted from any current paths not so determined; and is added to any new paths so determined. When added to a new path, the record is logically placed among any other records with the same alternate key value, ordered by primary key value.

REWRITE Statement -- for Relative Files

Function

The REWRITE statement logically replaces a record existing in a mass storage file.

General Format

```
REWRITE record-name [FROM identifier]

     [INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not refer to the same storage area.

2. Record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

3. The INVALID KEY phrase must not be specified for a REWRITE statement which references a file in sequential access mode.

4. 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.

General Rules

1. The file associated with record-name must be opened in the I-O mode at the time of execution of this statement.

2. For files in the sequential access mode, the last input-output statement executed prior to the execution of a REWRITE statement must be a successfully executed READ statement. The operating system logically replaces the record that was accessed by the READ statement.

3. For variable length records, the length of the record rewritten need not be the same length as the original record.

4. 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.

5. The current record pointer is not affected by the execution of the REWRITE statement.

6. The execution of the REWRITE statement causes the value of the FILE STATUS data item (if any) associated with the file, to be updated.

7. The operating system logically replaces the record specified by the contents of the RELATIVE KEY data item associated with the file if the file is accessed in either random or dynamic mode. If the file does not contain the record specified by the key, the INVALID KEY condition exists. The updating operation does not take place and the data in the record area is unaffected.

ROLLBACK Statement

Function

The ROLLBACK statement undoes the entire transaction in the database user's Before Image Journal. ROLLBACK is part of the DMS/TX protocol. For a complete discussion of DMS/TX refer to Chapter 3.

General Format

```
ROLLBACK [ ON ERROR imperative-statement ]
```

General Rules

1. The ROLLBACK statement undoes the entire transaction.

2. If there is an ON ERROR clause, any unsuccessful execution of the ROLLBACK statement (with a non-zero return code) causes the ERROR imperative statement to be executed.

3. If there is no ON ERROR clause, the user's program is cancelled upon unsuccessful execution of the ROLLBACK statement.

4. The special register RETURN-CODE contains the return code for the ROLLBACK statement. ROLLBACK return code values are as follows:

0	Unqualified success
4	Caller not in DMS/TX mode
8	DMS/TX not supported on this system
20	I/O error encountered while accessing BIJ
24	I/O error encountered while accessing Data File
32	Unable to set File Crash Status.

SEARCH Statement

Function

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index to indicate that table element.

General Format

Format 1

```
SEARCH identifier-1  VARYING  [{identifier-2 }]
                              [{index-name-1 }]

    [AT END imperative-statement-1]

    WHEN condition-1 { imperative-statement-2 }
                     { NEXT SENTENCE          }

   [WHEN condition-2 { imperative-statement-3 }] ...
   [                 { NEXT SENTENCE          }]
```

Format 2

```
SEARCH ALL identifier-1 [AT END imperative-statement-1]
        {                    {IS EQUAL TO} {identifier-3             } }
        { data-name-1        {IS =       } {literal-1                } }
WHEN    {                                  {arithmetic-expression-1  } }
        {                                                              }
        { condition-name-1                                             }
 [      {                    {IS EQUAL TO} {identifier-4             } }]
 [      { data-name-2        {IS =       } {literal-2                } }]
 [ AND  {                                  {arithmetic-expression-2  } }] ...
 [      {                                                              }]
 [      { condition-name-2                                             }]
{ imperative-statement-2 }
{ NEXT SENTENCE          }
```

Syntax Rules

1. In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description in the Data Division must contain an OCCURS clause and an INDEXED BY phrase. The description of identifier-1 in Format 2 must also contain the KEY IS phrase in its OCCURS clause.

2. Identifier-2 must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

3. In Format 1, condition-1 may be any condition as described in Section 12.4.

4. In Format 2, all referenced condition-names must be defined as having only a single value. The data-name associated with a condition-name must appear in the KEY clause of identifier-1.

   Each data-name-1 and data-name-2 must be indexed by the first index-name associated with identifier-1, along with other indices or literals as required, and must be referenced in the KEY clause of identifier-1.

   Identifier-3, identifier-4 or any identifiers specified in arithmetic-expression-1 or -2, must not be referenced in the KEY clause of identifier-1. Nor can they be indexed by the first index-name associated with identifier-1.

5. In Format 2, when a data-name in the KEY clause of identifier-1 is referenced, or when a condition-name associated with a data-name in the KEY clause of identifier-1 is referenced, all preceeding data-names in the KEY clause of identifier-1 or their associated condition-names must also be referenced.

6. Data-name-1 and data-name-2 may be qualified.

General Rules

Format 1

1. If Format 1 of SEARCH is used, a serial type of search operation takes place, starting with the current index setting.

   a. If, at the start of execution of the SEARCH statement, the index name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. The number of occurrences of identifier-1, the last of which is the highest permissible, is discussed under the OCCURS clause (refer to "OCCURS Clause" in Subsection 11.3.3.) Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next executable sentence.

b. If, at the start of execution of the SEARCH statment, the index name associated with identifier-1 contains a value corresponding to an occurrence number that is not greater than the highest permissable occurrence number for that identifier, the statement operates by evaluating the conditions in the order that they are written. The operation makes use of the index settings, wherever specified, to determine the occurrence of those items to be tested. The highest permissible occurrence number is the last occurence of the identifier. Refer to the OCCURS clause for a detailed discussion.

   If none of the conditions are satisfied, the index-name for identifier-1 is incremented to obtain the reference to the next occurrence. The process is then repeated using the new index-name settings. If the new value of the index-name settings for identifier-1 corresponds to a table element outside the permissible range of occurrence values, the search terminates as indicated in General Rule 1a.

   If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative statement associated with that condition is executed. The index-name remains set at the occurrence that caused the condition to be satisfied.

2. If imperative-statement-1, imperative-statement-2, or imperative-statement-3 does not terminate with a GO TO statement, control passes to the next executable sentence after execution.

3. If the VARYING phrase is not used, the index name that is used for the search operation is the first (or only) index name that appears in the INDEXED BY phrase of identifier-1. Any other index names for identifier-1 remain unchanged.

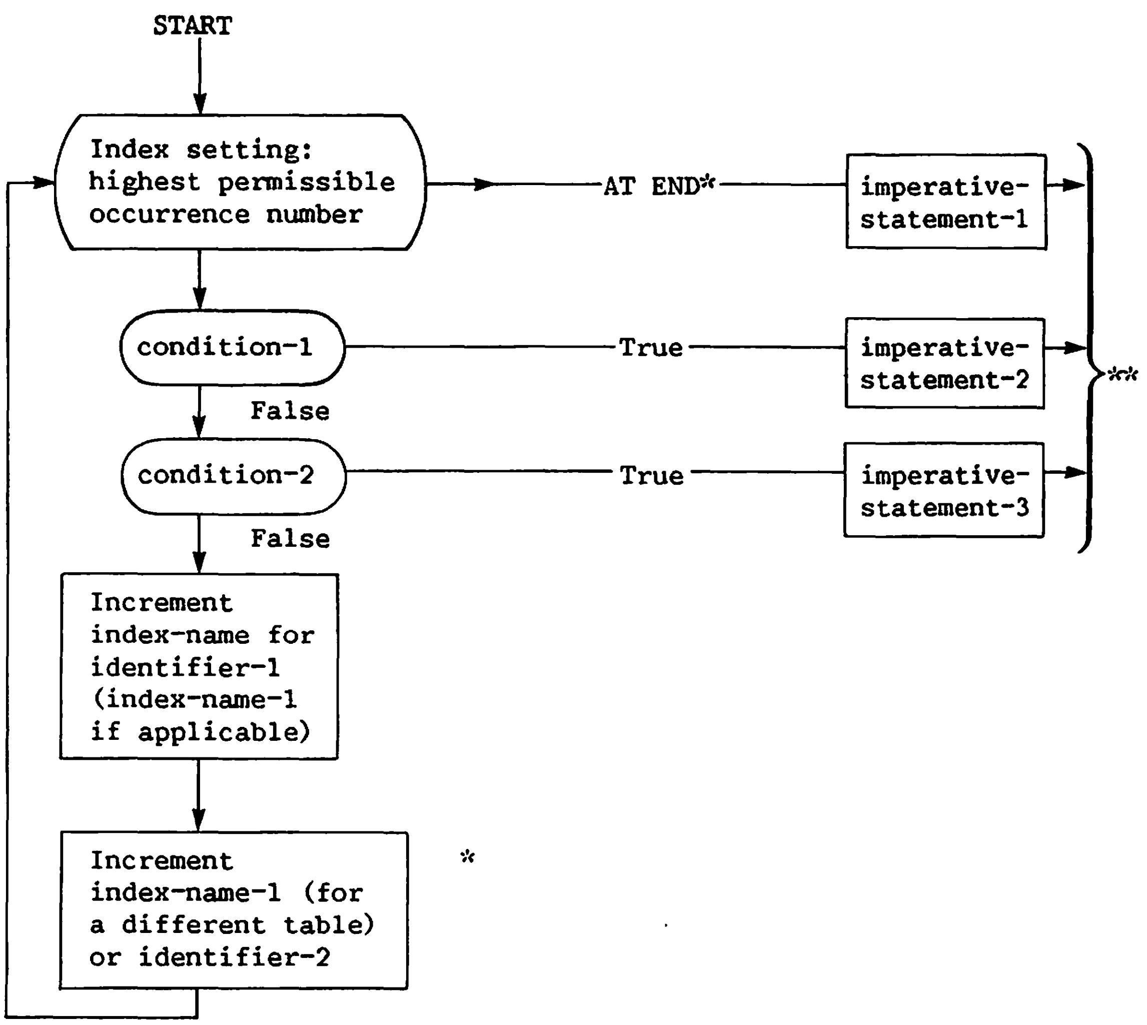
4. If the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of identifier-1, that index name is used for this search. If index-name-1 does not appear in the INDEXED BY phrase of identifier-1 or if the VARYING identifier-2 phrase is specified, the first (or only) index name given in the INDEXED BY phrase of identifier-1 is used for the search. In addition, the following operations will occur:

   a. If the VARYING index-name-1 phrase is specified, and if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the index associated with identifier-1.

b. If the VARYING identifier-2 phrase is specified, and if identifier-2 is an index data item, the data item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index associated with identifier-1. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value one at the same time as the index associated with identifier-1.

5. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a 2- or 3-dimensional table), an index name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of a SEARCH statement. To search an entire 2- or 3-dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index names must be adjusted to appropriate settings.

Figure 12-2 is a flowchart of the Format 1 SEARCH operation containing two WHEN phrases.


START
Index setting: highest permissible occurrence number
AT END*
imperative-statement-1
condition-1
True
imperative-statement-2
False
condition-2
True
imperative-statement-3
False
**
Increment index-name for identifier-1 (index-name-1 if applicable)
Increment index-name-1 (for a different table) or identifier-2
*

* These operations are options included only when specified in the SEARCH statement.

** Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

Figure 12-2. Flowchart for the SEARCH Statement

Format 2

1. The results of the SEARCH ALL operation are predictable only when the following conditions are met:

   a. The data in the table is ordered in the same manner as described in the ASCENDING/DESCENDING KEY clause associated with the description of identifier-1.

   b. The contents of the key(s) referenced in the WHEN clause is sufficient to identify a unique table element.

2. Format 2 provides an efficient method of finding an element in a large table by performing a binary SEARCH on the ordered table. The initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation in a manner which implements the binary search. For example, to find an element existing in a table of 32,000 elements, a maximum if 15 elements need be examined. At no time is the index-name for identifier-1 set to a value that exceeds the value corresponding to the last element of the table, or that is less than the value corresponding to the first element of the table. The length of the table is discussed in the OCCURS clause.

   If any of the conditions specified in the WHEN clause cannot be satisfied for any setting of the index within the permitted range, control is passed to imperative-statement-1 of the AT END phrase, when specified, or to the next executable sentence. In either case, the final setting of the index is <u>not</u> predictable. If all conditions can be satisfied, the index indicates an occurence that allows the conditions to be satisfied and control passes to imperative-statement-2.

3. If imperative-statement-1, imperative-statement-2, or imperative-statement-3 does not terminate with a GO TO statement, control passes to the next executable sentence after execution.

4. The index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY phrase of identifier-1. Any other index-names for identifier-1 remain unchanged.

5. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a 2- or 3-dimensional table), an index name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of a SEARCH statement. To search an entire 2- or 3 dimensional table it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index names must be adjusted to appropriate settings.

Example of SEARCH Statement

Figure 12-3 is a sample program using the SEARCH ALL statement and OCCURS clause.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  SEARCHALL.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 FILE SECTION.
000600*TEST SEARCH ALL STATEMENT
000700 WORKING-STORAGE SECTION.
000800 01  TABLE-1.
000900     05  EMPLOYEES  OCCURS 10 TIMES
001000                    ASCENDING  KEY IS EMP-ID
001100                    DESCENDING KEY IS EMP-YEARS,
001200                                      EMP-SHARES,
001300                                      EMP-ELIGIBLE
001400                    INDEXED BY AX, AX1.
001500         10  EMP-NAME        PIC X(20).
001600         10  EMP-ID          PIC 9(4).
001700         10  EMP-YEARS       PIC 99.
001800         10  EMP-SHARES      PIC 9(5).
001900         10  EMP-STATUS.
002000             15  EMP-ELIGIBLE    PIC X.
002100                 88  TRUE        VALUE  "T".
002200                 88  FALSE       VALUE  "F".
002300 01  TEST-RESULT.
002400     05  NAME            PIC X(20).
002500     05  YEARS           PIC 99.
002600 01  TEST-2-RESULT.
002700     05  NAME            PIC X(20).
002800     05  SHARES          PIC 9(5).
002900 01  TEST-3-RESULT.
003000     05  NAME            PIC X(20).
003100 PROCEDURE DIVISION.
003200 STARTUP.
003300     MOVE 30 TO RETURN-CODE.
003400     PERFORM INIT-EMP-NAME THRU INIT-EMP-ELIGIBLE.
003500*
003600*    THIS TEST FINDS THE NAME AND NUMBER OF YEARS OF SERVICE OF
003700*    EMPLOYEE WITH ID NUMBER 1076.
003800*
003900 SEARCH-1.
004000     SEARCH ALL EMPLOYEES
004100         AT END ADD 1 TO RETURN-CODE
004200         WHEN EMP-ID (AX) = 1076
004300             MOVE EMP-NAME (AX) TO NAME OF TEST-1-RESULT
004400             MOVE EMP-YEARS (AX) TO YEARS OF TEST-1-RESULT
004500             DISPLAY "NAME AND YEARS OF EMPLOYEE 1076 ARE ",
004600                     TEST-1-RESULT.
```

Figure 12-3. SEARCH ALL Example

```

*
*    THIS TEST RETURNS THE NAME AND NUMBER OF SHARES OF THE
*    EMPLOYEE WITH ID 1023 AND 4 YEARS EXPERIENCE.
*
 SEARCH-2.
     SEARCH ALL EMPLOYEES
         AT END ADD 1 TO RETURN-CODE
         WHEN EMP-ID (AX) = 1023
             AND EMP-YEARS (AX) = 4
                 MOVE EMP-NAME (AX) TO NAME OF TEST-2-RESULT
                 MOVE EMP-SHARES (AX) TO SHARES OF TEST-2-RESULT
                 DISPLAY "NAME AND SHARES OF EMPLOYEE 1023 ARE ",
                         TEST-2-RESULT.

*
*    THIS TEST RETURNS THE NAME OF THE EMPLOYEE WITH THE
*    CHARACTERISTICS INDICATED.
*
 SEARCH-3.
     SEARCH ALL EMPLOYEES
         AT END ADD 1 TO RETURN-CODE
         WHEN EMP-ID (AX) = 1060
             AND EMP-YEARS (AX) = 2
             AND EMP-SHARES (AX) = 350
             AND FALSE (AX)
             MOVE EMP-NAME (AX) TO NAME OF TEST-3-RESULT
             DISPLAY "NAME OF EMPLOYEE IS ", TEST-3-RESULT.

 END-PROGRAM.
     STOP RUN.

*
*    THIS PARAGRAPH INITIALIZES THE EMP-NAME FIELD IN EACH
*    TABLE ENTRY.  NOTE THAT THE NAME FIELD IS NOT ORDERED.
*
 INIT-EMP-NAME.
     MOVE "MADDEN E R          " TO EMP-NAME (1).
     MOVE "DUGGEN S            " TO EMP-NAME (2).
     MOVE "STOESSEL P W        " TO EMP-NAME (3).
     MOVE "OZKAYNAK L W        " TO EMP-NAME (4).
     MOVE "TOR N               " TO EMP-NAME (5).
     MOVE "KILPATRICK R        " TO EMP-NAME (6).
     MOVE "CARUSO A H          " TO EMP-NAME (7).
     MOVE "DUNCAN W            " TO EMP-NAME (8).
     MOVE "MAMIYA B C          " TO EMP-NAME (9).
     MOVE "FRIEDMAN M          " TO EMP-NAME (10).

```

Figure 12-3. SEARCH ALL Example (continued)

```
009500*    THIS PARAGRAPH INITIALIZES THE EMP-ID FIELD IN EACH TABLE
009600*    ENTRY WITH VALUES IN ASCENDING ORDER.
009700*
009800 INTI-EMP-ID.
009900     MOVE 0900 TO EMP-ID (1).
010000     MOVE 0955 TO EMP-ID (2).
010100     MOVE 1023 TO EMP-ID (3).
010200     MOVE 1026 TO EMP-ID (4).
010300     MOVE 1038 TO EMP-ID (5).
010400     MOVE 1060 TO EMP-ID (6).
010500     MOVE 1061 TO EMP-ID (7).
010600     MOVE 1062 TO EMP-ID (8).
010700     MOVE 1076 TO EMP-ID (9).
010800     MOVE 1105 TO EMP-ID (10).
010900
011000*    THIS PARAGRAPH INITIALIZES THE EMP-YEARS IN EACH TABLE
011100*    ENTRY WITH VALUES IN DESCENDING ORDER.
011200*
011300 INIT-EMP-YEARS.
011400     MOVE 6 TO EMP-YEARS (1).
011500     MOVE 5 TO EMP-YEARS (2).
011600     MOVE 4 TO EMP-YEARS (3).
011700     MOVE 4 TO EMP-YEARS (4).
011800     MOVE 3 TO EMP-YEARS (5).
011900     MOVE 2 TO EMP-YEARS (6).
012000     MOVE 2 TO EMP-YEARS (7).
012100     MOVE 2 TO EMP-YEARS (8).
012200     MOVE 1 TO EMP-YEARS (9).
012300     MOVE 1 TO EMP-YEARS (10).
012400
012500*
012600*    THIS PARAGRAPH INITIALIZES THE EMP-SHARES FIELD IN
012700*    EACH TABLE ENTRY WITH VALUES IN DESCENDING ORDER.
012800*
012900 INIT-EMP-SHARES.
013000     MOVE 02000 TO EMP-SHARES (1).
013100     MOVE 01500 TO EMP-SHARES (2).
013200     MOVE 01200 TO EMP-SHARES (3).
013300     MOVE 01100 TO EMP-SHARES (4).
013400     MOVE 01100 TO EMP-SHARES (5).
013500     MOVE 00350 TO EMP-SHARES (6).
013600     MOVE 00250 TO EMP-SHARES (7).
013700     MOVE 00250 TO EMP-SHARES (8).
013800     MOVE 00000 TO EMP-SHARES (9).
013900     MOVE 00000 TO EMP-SHARES (10).
014000
```

Figure 12-3. SEARCH ALL Example (continued)

```
014100*
014200*    THIS PARAGRAPH INITIALIZES THE EMP-ELIGIBLE FIELD
014300*    TO TRUE OR FALSE IN DESCENDING ORDER.  EMPLOYEES WITH
014400*    1000 OR MORE SHARES HAVE THE ELIGIBLE FIELD SET TO TRUE.
014500*
014600 INIT-EMP-ELIGIBLE.
014700     MOVE "T" TO EMP-ELIGIBLE (1).
014800     MOVE "T" TO EMP-ELIGIBLE (2).
014900     MOVE "T" TO EMP-ELIGIBLE (3).
015000     MOVE "T" TO EMP-ELIGIBLE (4).
015100     MOVE "T" TO EMP-ELIGIBLE (5).
015200     MOVE "F" TO EMP-ELIGIBLE (6).
015300     MOVE "F" TO EMP-ELIGIBLE (7).
015400     MOVE "F" TO EMP-ELIGIBLE (8).
015500     MOVE "F" TO EMP-ELIGIBLE (9).
015600     MOVE "F" TO EMP-ELIGIBLE (10).
015700
```

Figure 12-3. SEARCH ALL Example (continued)

SET Statement

Function

The SET statement establishes reference points for table handling operations by setting index names associated with table elements.

General Format

Format 1

$$\underline{\text{SET}} \left\{ \begin{array}{ll} \text{identifier-1} & [\text{identifier-2}\ \ ]\ldots \\ \text{index-name-1} & [\text{index-name-2}]\ldots \end{array} \right\} \ \underline{\text{TO}} \ \left\{ \begin{array}{l} \text{identifier-3} \\ \text{index-name-3} \\ \text{integer-1} \end{array} \right\}$$

Format 2

$$\underline{\text{SET}} \ \text{index-name-4 [index-name-5]}\ldots \left\{ \begin{array}{l} \underline{\text{UP}}\ \underline{\text{BY}} \\ \underline{\text{DOWN}}\ \underline{\text{BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-4} \\ \text{integer-2} \end{array} \right\}$$

Format 3

$$\underline{\text{SET}} \ \text{figurative-constant} \ \left\{ \begin{array}{l} \underline{\text{IN}}\ \underline{\text{FAC}}\ \underline{\text{OF}} \\ \underline{\text{OF}} \end{array} \right\} \left\{ \begin{array}{l} \text{display-item} \\ \text{identifier-5} \end{array} \right\} \ \left\{ \begin{array}{l} \underline{\text{ON}} \\ \underline{\text{OFF}} \end{array} \right\}$$

Syntax Rules

1. 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.

2. Identifier-1 and identifier-3 must name either index data items, or elementary items described as an integer.

3. Identifier-4 must be described as an elementary numeric integer.

4. Integer-1 and integer-2 can be signed. Integer-1 must be positive.

5. The figurative constant must be defined as 1-byte in the FIGURATIVE-CONSTANTS paragraph.

6. Identifier-5 must reference a 1-byte item.

7. Display-item must be an item in a Working-Storage record whose USAGE IS DISPLAY-WS or in a record of a file for which the device type "DISPLAY" has been declared in the SELECT clause.

General Rules

1. Index names are considered related to a given table and are defined by being specified in the INDEXED BY phrase.

2. 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 SEARCH or PERFORM statement may be undefined.

3. In Format 1, the following action occurs:

   a. Index-name-1 is set to a value that causes 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.

   b. If identifier-1 is an index data item, it can 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. Identifier-1 can also be set equal to the contents of identifier-3 where identifier-3 is an integer; in this case, identifier-3 is treated as a USAGE BINARY item and moved to identifier-1.

   c. If identifier-1 is an integer data item, it can be set to an occurrence number that corresponds to the value of index-name-3. It can also be set equal to identifier-3 where identifier-3 is an index data item; in this case, identifier-3 is treated as a USAGE BINARY item and moved to identifier-1.

   d. The process is repeated for index-name-2, ... identifier-2, ... 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,... is evaluated immediately before the value of the respective data item is changed.

4. In Format 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by the number of occurrences represented by the value of integer-2 or identifier-4; thereafter, the process is repeated for index-name-5, ... Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

5. Data in the following chart represents the validity of various operand combinations in Formats 1 and 2 of the SET statement. The numbers indicate the applicable general rules.

Table 12-7. Valid Operands for the SET Statement

Sending Item	Receiving Item		
	Integer Data Item	Index-name	Index Data Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	No/3c	Valid/3a	Valid/3b *
Index-Name	Valid/3c	Valid/3a	Valid/3b *
Index Data Item	Valid/3c*	Valid/3a*	Valid/3b *
* No conversion takes place.			

6. Format 3 permits the user to set selected bits in any 1-byte item. Bits in the specified item corresponding to bits set to 1 in the figurative constant are set to 1 if the ON option is coded or to 0 if the OFF option is coded.

SORT Statement

Function

The SORT statement creates a sort file either by executing an input procedure or by transferring records from another file. The statement then sorts the records on a set of specified keys and makes each record available, in the sorted order, to an output file or procedure.

General Format

$$\underline{\text{SORT}}\ \text{file-name-1 ON}\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\text{KEY data-name-1 [data-name-2] ...}$$

$$\left[\text{ON}\left\{\begin{array}{l}\underline{\text{ASCENDING}}\\ \underline{\text{DESCENDING}}\end{array}\right\}\text{KEY data-name-3 [data-name-4] ...}\right]\text{ ...}$$

$$\text{[WITH }\underline{\text{DUPLICATES}}\text{ IN ORDER]}$$

$$\text{[COLLATING }\underline{\text{SEQUENCE}}\text{ IS alphabet-name]}$$

$$\left\{\begin{array}{l}\underline{\text{INPUT}}\ \underline{\text{PROCEDURE}}\text{ IS section-name-1 }\left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\}\text{section-name-2}\right]\\ \underline{\text{USING}}\text{ file-name-2 [file-name-3] ...}\end{array}\right\}$$

$$\left\{\begin{array}{l}\underline{\text{OUTPUT}}\ \underline{\text{PROCEDURE}}\text{ IS section-name-3 }\left[\left\{\begin{array}{l}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{array}\right\}\text{section-name-4}\right]\\ \underline{\text{GIVING}}\text{ file-name-4}\end{array}\right\}$$

Syntax Rules

1. File-name-1 must be described in a sort-merge (SD) file description entry in the Data Division.

2. Section-name-1 represents the name of an input procedure. Section-name-3 represents the name of an output procedure.

3. File-name-2, -3, and -4 must be described in a file description (FD) entry, not an SD entry. The actual size of the logical record(s) for these file-names must be equal to the actual size described for file-name-1. If the data descriptions of the elementary items that make up these records are not identical, it is the programmer's responsibility to describe the corresponding records to cause equal amounts of character positions to be allocated for the corresponding records.

4. Data-name-1, -2, -3, and -4 are KEY data-names and are subject to the following rules:

   a. The data items identified by KEY data-names must be describe in records associated with file-name-1.

b. KEY data-names may be qualified.

c. The data items identified by KEY data-names must not be variable length items.

d. If file-name-1 has more than one record description, the data items identified by KEY data-names need be described in only one of the record descriptions.

e. None of the data items identified by KEY data-names can be described by an entry which either contains an OCCURS clause, or is subordinate to an entry which contains one.

5. The words THROUGH and THRU are equivalent.

General Rules

1. The Procedure Division may contain more than one SORT statement. Sort statements can not appear in the declaratives portion or in the input and output procedures associated with a SORT or MERGE statement.

2. The data-names following the word KEY are listed from left to right in the order of decreasing significance without regard to how they are divided into KEY phrases. Data-name-1 is the major key, data-name-2 is the next most significant key, etc. The sort sequence is also governed by the following rules:

   a. When the ASCENDING phrase is specified, the sorted sequence is from the lowest value of the contents of the data items identified by the KEY data-names to the highest value. This rule is in accordance with the rules for comparison of operands in a relational condition.

   b. When the DESCENDING phrase is specified, the sorted sequence is from the highest value of the contents of the data items identified by the KEY data-names to the lowest value. This rule is also governed by the rules for comparison of operands in a relational condition.

3. If the DUPLICATES phrase is specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding KEY data items associated with one or more other data records, the order of return of these records is as follows:

   a. the order of the associated input files as specified in the SORT statement. Within a given input file, the order is that in which the records are accessed from that file.

   b. The order in which these records are released by an input procedure, if such a procedure is specified.

4. If the DUPLICATES phrase is <u>not</u> specified and the contents of all the key data items associated with one data record are equal to the contents of the corresponding key data items associated with one or more other data records, the order of return of these records is undefined.

5. The COLLATING SEQUENCE IS phrase is currently treated as a comment by the compiler.

6. The input procedure must consist of one or more sections. In order to transfer records to the file referenced by file-name-1, the input procedure must contain at least one RELEASE statement.

   Transfer of control to points outside an input procedure is permitted, but it is the responsibility of the programmer to ensure a return to the input procedure.

   The input procedure can include any procedures needed to select, create, or modify records. The input procedure must not contain SORT or MERGE statements.

   Transfer of control to points inside the input procedure by ALTER, GO TO, and PERFORM statements are allowed. During the execution of a SORT statement, transfers of control to an input procedure is legal if the records have not yet been sorted. If a SORT statement is <u>not</u> being executed, such transfers must not cause the execution of a RETURN or RELEASE statement, or cause control to reach the end of the input procedure.

7. If an input procedure is specified, control is passed to that procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure, and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

8. The output procedure must consist of one or more contiguous source program sections that do not form a part of any input procedure. In order to make sorted records available for processing, the output procedure must include the execution of at least one RETURN statement.

   Control must not be passed to the output procedure except when a related SORT statement is being executed. The output procedure may consist of any procedures needed to select, modify, or copy the records being returned from the sort file. Records are returned one at a time. The restrictions on the procedural statements within the output procedure are as follows:

   a. The output procedure must not contain any SORT or MERGE statements.

b. Transfer of control to points outside an output procedure is permitted, but it is the responsibility of the programmer to ensure a return to the output procedure.

c. Transfer of control to points inside the output procedure by ALTER, GO TO, and PERFORM statements are allowed. During the execution of a SORT statement, transfers of control to an output procedure is legal if the records have been sorted. If a SORT statement is <u>not</u> being executed, such transfers must not cause the execution of a RETURN or RELEASE statement, or cause control to reach the end of the output procedure.

9. If an output procedure is specified, control is passed to it after file-name-1 has been sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section of the procedure. This mechanism terminates the sort and passes control to the next executable statement. Before entering the output procedure the sort procedure reaches a point where it can select the next record in sorted order when requested. The RETURN statements in the output procedure are the requests for the next record.

10. When the USING clause is specified, all records in file-name-2 and -3 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 and -3 must not be open. The SORT statement automatically initiates the processing of, makes available the logical records for, and terminates the processing of file-name-2 and -3. These implicit functions are performed in such a way that any associated USE procedures are executed.

    The terminating function for all files is performed as if a CLOSE statement, without optional phrases, is executed for each. The SORT statement also automatically performs the implicit functions of moving records from the file area of file-name-2 and -3 to the file area for file-name-1, as well as releasing the records to the initial phase of the sort operation.

    When the USING clause is specified, the programmer may have also coded a VALUE OF clause for the FD. In this case, the SPACE IS phrase has no effect for disk resident files, but does for tape resident files.

11. When the GIVING phrase is specified, all sorted records in file-name-1 are automatically written to file-name-4 as the implied output procedure for this SORT statement. File-name-4 must not be open at the time of execution of the SORT statement. The SORT statement automatically initiates the processing of, releases the logical records to, and terminates the processing of file-name-4.

The terminating function is performed as if a CLOSE statement, without optional phrases, is executed for the file. The SORT statement also automatically performs the implicit functions of the return of the sorted records from the final phase of the sort operation and the moving of records from the file area for file-name-1 to the file area for file-name-4.

12. The success or failure if a sort operation can be checked by testing the VS COBOL special register, RETURN-CODE. RETURN-CODE has an implied PICTURE of 999 and must not be defined in the program. Possible values are as follows:

RETURN-CODE	MEANING
0	Successful operation.
4	No records to be sorted; input records did not meet the selection criteria, or an empty input file was specified.
8	Insufficient space in stack or I/O buffer.
12	Record size is more than 2024 bytes long.
16	Invalid sort key.
20	Unexpected program check.
24	Input records are out of order in the file to be merged; program cannot proceed.
28	Input record count problem.

START Statement -- for Consecutive Files

Function

The START statement provides a method to determine the current file status value of a workstation file. The file status value can be used to determine whether or not a subsequent READ request would wait (for the keyboard to be locked) or would be processed immediately.

General Format

```
START file-name
```

Syntax Rules

1. Access mode must be RANDOM and device type must be DISPLAY.

General Rules

1. The file must be open in the I-O mode at the time the START statement is executed.

2. Records of the consecutive file must be variable-length.

3. The only effect of the execution of the START statement is that the FILE STATUS data item, defined in the file control entry, is updated. This data item can be used to determine if a following READ would wait for operator action.

   If the second byte of the FILE STATUS data item (the "Attention Identifier" or "AID" character) is blank the keyboard is unlocked, which could cause a READ request to wait. If the AID character is not blank, the keyboard is locked, and the AID character represents a code that indicates which key (ENTER or PF key) was last used by the workstation operator. (Refer to "AID Characters" in Appendix E, File Status Key Values.)

START Statement -- for Indexed Files

Function

The START statement provides a basis for logical positioning within an indexed file for subsequent sequential retrieval of records.

General Format

$$\underline{\text{START}}\ \text{file-name}\ \left[\ \underline{\text{KEY}}\ [\text{data-name-1}]\ \left\{\begin{array}{l}\text{IS }\underline{\text{EQUAL}}\text{ TO}\\ \text{IS} = \\ \text{IS }\underline{\text{GREATER}}\text{ THAN}\\ \text{IS} > \\ \text{IS }\underline{\text{NOT}}\ \underline{\text{LESS}}\text{ THAN}\\ \text{IS }\underline{\text{NOT}} < \end{array}\right\}\ \text{data-name-2}\ \right]$$

$$[\underline{\text{INVALID}}\ \text{KEY imperative-statement}]$$

Syntax Rules

1. File-name must be the name of a file for which SEQUENTIAL or DYNAMIC access has been specified.

2. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for the file.

3. Data-name-1, if supplied, must be the name of an alternate record key specified for the file in the ALTERNATE RECORD KEY clause of the file control entry.

4. If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name-2 can reference one of the following:

   a. The primary record key associated with the file

   b. An alternate record key associated with the file

   c. Any data item subordinate to the data item specified as the primary record key associated with the file whose leftmost character position corresponds to the leftmost character position of that record key data item

   d. Any data item subordinate to the data-name-1 item whose leftmost character position corresponds to the leftmost character position of the data-name-1 item.

5. Data-name-1 and data-name-2 may be qualified.

General Rules

1. The file must be open in the input, I-O or shared mode at the time that the START statement is executed.

2. If the KEY phrase is not specified the relational operator IS EQUAL TO is implied.

3. The 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 as specified in General Rule 7. If file-name references an indexed file and the operands are of unequal size, comparison proceeds as though the longer one was truncated on the right such that its length is equal to that of the shorter. All other nonnumeric comparison rules apply.

   a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.

4. The execution of the START statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated.

5. If the KEY phrase is specified, the comparison described in General Rule 3 uses the data item referenced by data-name-2.

6. If the KEY phrase is not specified, the comparison described in General Rule 3 uses the data item referenced in the RECORD KEY clause in the file control entry for the file.

7. Upon completion of the successful execution of the START statement, a key of reference is established and used in subsequent Format 1 READ statements (Refer to "READ Statement -- for Indexed Files" earlier in this section) as follows:

   a. If the KEY phrase is not specified, the primary record key specified for file-name becomes the key of reference.

   b. If the KEY phrase is specified, and data-name-1 is not specified, the primary record key becomes the key of reference.

   c. If the KEY phrase is specified, and data-name-1 is specified, the alternate record key specified by data-name-1, becomes the key of reference.

8. If the execution of the START statement is not successful, the key of reference is undefined.

START Statement -- for Relative Files

Function

The START statement provides a basis for logical positioning within a relative file for the purpose of subsequent, sequential retrieval of records.

General Format

$$\underline{\text{START}}\ \text{file-name}\ \left[\ \underline{\text{KEY}}\ \text{IS} \left\{\begin{array}{l} \underline{\text{EQUAL}}\ \text{TO} \\ = \\ \underline{\text{GREATER}}\ \text{THAN} \\ > \\ \underline{\text{NOT}}\ \underline{\text{LESS}}\ \underline{\text{THAN}} \\ \underline{\text{NOT}} < \\ \underline{\text{NOT}}\ \underline{\text{GREATER}}\ \text{THAN} \\ \underline{\text{NOT}} > \\ \underline{\text{LESS}}\ \text{THAN} \\ < \end{array}\right\} \text{data-name}\ \right]$$

$$[\underline{\text{INVALID}}\ \text{KEY imperative-statement}]$$

Syntax Rules

1. File-name must be the name of a file with sequential or dynamic access.

2. Data-name can be qualified.

3. The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

4. If data-name is specified, it must be the data item specified in the RELATIVE KEY phrase of the associated file control entry.

NOTE

The relational characters >, <, and = are not underlined, even though they are required. This is to avoid confusing them with other symbols such as $\underline{>}$ (greater than or equal to).

General Rules

1. File-name must be opened in the INPUT or I-O mode at the time the START statement is executed.

2. If the KEY phrase is not specified, the relational operator, IS EQUAL TO, is implied.

3. 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 the data item referenced by the RELATIVE KEY clause associated with file-name. The following rules also apply:

    a. The current record pointer is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

    b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists. In this case the execution of the START statement is unsuccessful and the position of the current record pointer is undefined.

4. The execution of the START statement causes the value of the FILE STATUS data item (if any) associated with file-name to be updated.

STOP Statement

Function

The STOP statement causes a permanent or temporary suspension of the execution of the object program.

General Format

$$\underline{\text{STOP}} \begin{Bmatrix} \underline{\text{RUN}} \\ \text{literal} \end{Bmatrix}$$

Syntax Rules

1. The literal can be numeric or nonnumeric.

2. If the literal is numeric, then it must be an unsigned integer.

3. 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.

General Rules

1. If the RUN phrase is used, then the ending procedure established by the compiler is instituted.

2. If STOP literal is specified, the literal is communicated to the operator. Continuation of the object program begins with the execution of the next executable statement in sequence.

3. If a figurative constant is used instead of the literal, it cannot be a user-defined figurative constant.

4. The value of the special-register RETURN-CODE is passed to the operating system command language when a STOP RUN is issued. The value of RETURN-CODE is initially 0. Valid RETURN-CODE values range from 0 to 999. The user can supply an overriding value for RETURN-CODE, as in the following statement.

```
MOVE "99" TO RETURN-CODE.
```

STRING Statement

Function

The STRING statement allows the movement of two or more data items into a single data item. The data being moved does not have to be contiguous, nor does it have to be part of the same record description. STRING allows the movement of both partial contents, or the complete contents of the data items.

General Format

```
                                              { identifier-2 } }
STRING { { identifier-1 } ...  DELIMITED BY  { literal-2    } } ...
       { { literal-1    }                    { SIZE         } }

     INTO identifier-3

     [WITH POINTER identifier-4]

     [ON OVERFLOW imperative statement]
```

Syntax Rules

1. Each literal may be any figurative constant without the optional word ALL.

2. All literals must be described as nonnumeric literals, and all identifiers, except identifier-4, must be described implicitly or explicitly as USAGE IS DISPLAY.

3. Identifier-3 must not represent an edited data item and must not be described with the JUSTIFIED clause.

4. Identifier-4 must be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-3. The symbol "P" may not be used in the PICTURE character-string of identifier-4.

5. Where identifier-1 or identifier-2 is an elementary numeric data item, it must be described as an integer without the symbol "P" in its PICTURE character-string.

General Rules

1. All references to identifier-1, identifier-2, literal-1 and literal-2 apply equally to all recursions thereof.

2. Identifier-1 or literal-1 represents the sending item. Identifier-3 represents the receiving item.

3. Literal-2 or identifier-2 indicates the character(s) delimiting the move. If the SIZE phrase is used, the contents of the complete data item defined by identifier-1 (or literal-1) is moved. When a figurative constant is used as the delimiter, it is a single character, nonnumeric literal.

4. When a figurative constant is specified as literal-1 or literal-2, it refers to an implicit, one character data item whose usage is DISPLAY.

5. When the STRING statement is executed, the transfer of data is governed by the following rules:

   a. Those characters from the transferring field are transferred to the receiving field in accordance with the rules for alphanumeric to alphanumeric moves, except that no space filling is provided. (Refer to the MOVE Statement.)

   b. When using the DELIMITED phrase without the SIZE phrase;
      The content of the sending field is transferred to the receiving field in the sequence specified in the STRING statement.

      The transfer begins with the leftmost character of the sending field and proceeds from left to right.

      Transfer continues until one of the following conditions is met:
      - The end of the sending field is reached
      - The end of the receiving field is reached
      - The character(s) specified by literal-2, or by the content of identifier-2 are encountered. Literal-2 and identifier-2 are not transferred.

   c. When using the DELIMITED phrase with the SIZE phrase;
      - The entire content of the transferring field is transferred in the sequence specified in the STRING statement.

      - The transfer continues until all data has been transferred, or until the end of the receiving field is reached.

   d. The process is repeated until all occurences of the sending field have been processed.

6. If the WITH POINTER phrase is specified, identifier-4 must be set to an initial value greater than zero prior to the execution of the STRING statement.

7. If the WITH POINTER phrase is not specified, general rules 8 - 13 assume identifier-4 is specified with an initial value of 1.

8. If the WITH POINTER phrase is specified, the transfer of data is governed by the following rules:

   a. Characters behave as if they are moved one at a time as long as the value of identifier-4 does not exceed the length of the data item referenced by identifier-3.

   b. The value of identifier-4 behaves as if it is increased by one after each character is moved.

   c. The value of identifier-4 is not changed in any other way during the execution of the STRING statement.

9. The execution of the STRING statement causes only that portion of the data item that was referenced during the execution of the STRING statement to be changed. All other portions of the data item will contain data that was present before the execution.

10. If, during execution, identifier-4 is either less than one or exceeds the value of identifier-3, no (further) data is transferred to identifier-3. In this situation, the imperative statement in the ON OVERFLOW phrase is executed if it is present.

11. If the ON OVERFLOW phrase is not specified when the conditions described in general rule 10 are encountered, control is transferred to the next executable statement.

12. The evaluation of subscripting or indexing for the identifiers is done once, as the first operation of the execution of the statement.

13. If identifier-1 or identifier-2 occupies the same storage area as identifier-3 or identifier-4, or if identifier-3 and identifier-4 occupy the same storage area, the result of the execution of the statement is undefined. This rule holds true even if the identifiers are defined by the same data description entry. (Refer to Section 12.3.3, Overlapping Operands.)

SUBTRACT Statement

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from a numeric data item, and set the value of an item equal to the result.

General Format

Format 1

```
SUBTRACT {identifier-1} [identifier-2] ... FROM identifier-m [ROUNDED]
         {literal-1   } [literal-2   ]
         [ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT {identifier-1} [identifier-2] ... FROM {identifier-m}
         {literal-1   } [literal-2   ]          {literal-m   }
    GIVING identifier-n [ROUNDED]
        [ON SIZE ERROR imperative-statement]
```

Format 3

```
SUBTRACT {CORRESPONDING} identifier-1 FROM identifier-2 [ROUNDED]
         {CORR         }

         [ON SIZE ERROR imperative statement]
```

Syntax Rules

1. Each identifier must refer to a numeric elementary item except;

   a. In Format 2, the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

   b. In Format 3 each identifier must refer to a group item.

2. Each literal must be a numeric literal.

3. The composite of operands must not contain more than 18 digits. (Refer to Section 12.3.3, Arithmetic Statements.)

   a. In Format 1 the composite of operands is determined by using all of the operands in a given statement.

   b. In Format 2 the composite of operands is determined by using all of the operands in a given statement excluding the data item that follows the word GIVING.

c. In Format 3 the composite of operands is determined separately for each pair of corresponding data items.

4. CORR is an abbreviation for CORRESPONDING.

General Rules

1. Refer to "ROUNDED Phrase", "SIZE ERROR Phrase", and "Overlapping Operands" in Section 12.3.3, Arithmetic Statements, and The CORRESPONDING Phrase in Section 11.2.1.

2. 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.

3. 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.

4. When subtract is used, enough places are carried so as not to lose any significant digits during execution.

5. If Format 3 is used, data items in identifier-1 are subtracted from and stored in corresponding data items in identifier-2.

Examples of SUBTRACT Statement

```
SUBTRACT VOL-1 VOL-2 FROM VOL-3
```

The sum of VOL-1 and VOL-2 is subtracted from VOL-3, and the result is placed in VOL-3.

```
SUBTRACT 10 FROM VOL-1 GIVING VOL-2
```

The difference obtained by subtracting 10 from the value of VOL-1 is placed in VOL-2.

```
SUBTRACT CORR FILE1 FROM FILE2.
```

Elementary items from FILE1 are subtracted from and stored in corresponding elementary items in FILE2.

UNSTRING Statement

Function

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

General Format

```
UNSTRING identifier-1

    [ DELIMITED BY [ALL] { identifier-2 } [ OR [ALL] { identifier-3 } ] ... ]
                         { literal-1    }             { literal-2    }

    INTO { identifier-4 [DELIMITER IN identifier-5] [COUNT IN identifier-6] } ...

    [WITH POINTER identifier-7]

    [TALLYING IN identifier-8]

    [ON OVERFLOW imperative-statement]
```

Syntax Rules

1. Each literal must be a nonnumeric literal.

2. Identifier-1, identifier-2, identifier-3, and identifier-5 must reference data items described, implicitly or explicitly, as alphanumeric.

3. Identifier-4 may be described as either alphabetic, alphanumeric, or numeric (except that the symbol "P" may not be used in the PICTURE character-string), and must be described implicitly or explicitly as USAGE IS DISPLAY.

4. Identifier-6 and identifier-8 must reference integer data items (except that the symbol "P" may not be used in the PICTURE character-string).

5. Identifier-7 must be described as an elementary numeric integer data item of sufficient size to contain a value equal to 1 plus the size of the data item referenced by identifier-1. The symbol "P" may not be used in the PICTURE character-string of identifier-7.

6. The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is also specified.

7. No identifier may name a level 88 entry.

General Rules

1. All references to identifier-2, literal-1, identifier-4, identifier-5, and identifier-6 apply equally to all recursions thereof.

2. Identifier-1 represents the sending area.

3. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

4. Literal-1 or identifier-2 specifies a delimiter.

5. Identifier-6 represents the count of the number of characters within identifier-1, isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter character(s).

6. Identifier-7 contains a value that indicates a relative character position within identifier-1.

7. Identifier-8 is a counter which is incremented by 1 for each occurrence of identifier-4 accessed during the operation.

8. When a figurative constant is used as the delimiter, it stands for a single character, nonnumeric literal.

   When the ALL phrase is specified, one occurence or two or more contiguous occurences of literal-1 (figurative constant or not), or the content of the data item referenced by identifier-2, are treated as if they were only one occurence. Further, one occurence of literal-1 or the data item referenced by identifier-2 is moved to the receiving data item according to the rules specified in General Rule 13d.

9. When any examination encounters two contiguous delimiters, and the current receiving area is described as alphabetic or alphanumeric, that area is filled with spaces. If the receiving area is described as numeric, the area is filled with zeros.

10. Literal-1 or identifier-2 can contain any character in the computer's character set.

11. Each literal-1, or identifier-2, represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item and in the order given to be recognized as a delimiter.

12. When two or more delimiters are specified in the DELIMITED BY phrase, an OR condition exists between them. Each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is considered to be a single delimiter. No character(s) in the sending field can be considered a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

13. When the UNSTRING statement is initiated, the current receiving area is identifier-4. Data is transferred from identifier-1 to identifier-4 according to the following rules:

    a. If the POINTER phrase is specified, identifier-1 is examined beginning with the relative character position indicated by identifier-7. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by literal-1 or the value of identifier-2 is encountered (refer to General Rule 11).

       If the DELIMITED BY phrase is <u>not</u> specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

       If the end of identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

    c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric date item, and are moved into the current receiving area according to the rules for the MOVE statement (refer to the MOVE statement).

    d. If the DELIMITER IN phrase is specified, the delimiting character(s) is treated as an elementary alphanumeric data item and is moved into identifier-5 according to the rules for the MOVE statement (refer to the MOVE statement). If the delimiting condition is the end of identifier-1, identifier-5 is filled with spaces.

    e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into identifier-6 according the rules for an elementary move.

f. If the DELIMITED BY phrase is specified, the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to identifier-4, the current receiving area is the data item referenced by the next occurrence of identifier-4. The behavior described in paragraphs 13b and 13f is repeated until either all the characters in identifier-1 are exhausted or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase of the TALLYING phrase is the responsibility of the user.

15. The content of identifier-7 is incremented by one for each character examined in identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, identifier-7 contains a value equal to the initial value plus the number of characters examined in identifier-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the content of identifier-8 contains a value equal to its value at the beginning of the execution of the statement, plus a value equal to the number of identifier-4 receving data items accessed during execution of the statement.

17. Either of the following situations causes an overflow condition:

a. An UNSTRING is initiated and the value in identifier-7 is less than 1 or greater than the size of identifier-1.

b. If, during execution, all receiving areas have been acted upon, and identifier-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative-statement is executed. If the ON OVERFLOW phrase is not specified, control is transferred to the next executable statement.

19. The evaluation of subscripting and indexing for the identifiers is done only once, as the first operation of the execution of the statement.

20. The result of the execution of this statement is undefined if any of the following conditions exist:

    a. Identifier-1, identifier-2, or identifier-3 occupies the same storage area as identifier-4, identifier-5, identifier-6, identifier-7 or identifier-8.

    b. Identifier-4, identifier-5, or identifier-6 occupies the same storage area as identifier-7, or identifier-8.

    c. Identifier-7 and identifier-8 occupy the same storage area.

    This rule holds true even if the various identifiers are defined by the same data description entry (refer to Section 12.3.3, Overlapping Operands).

USE Statement

Function

The USE statement can specify procedures for input/output error handling that supplement the standard procedures provided by the input/output control system. It can also identify the user items that are to be monitored by an associated debugging section.

In DMS/TX protocol a deadlock can occur between two tasks. DMS/TX will choose one of the tasks for deadlock rollback to allow the other task to continue. A deadlock exit address may be specified by using a deadlock declarative. This alters the default system deadlock handling process to allow the original COBOL program to retain control instead of being cancelled. For a complete discussion of DMS/TX protocol refer to Chapter 3.

General Format

Format 1

```
USE AFTER STANDARD { EXCEPTION }  PROCEDURE ON
                   { ERROR     }

    { file-name-1 [file-name-2] ... }
    { INPUT                         }
    { OUTPUT                        } .
    { I-O                           }
    { SHARED                        }
    { EXTEND                        }
```

Format 2

```
USE FOR DEBUGGING ON  { procedure-name-1   [procedure-name-2] ... }
                      { ALL PROCEDURES                            }
```

Format 3

```
USE AFTER DEADLOCK.
```

Syntax Rules

1. 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 paragraphs that define the procedures to be used.

2. The USE statement itself is never executed; it merely defines the conditions calling for the execution of the USE procedures.

3. The same file name can appear in a different specific arrangement of the format. Appearance of a file name in a USE statement must not cause the simultaneous request for execution of more than one USE procedure.

4. The words ERROR and EXCEPTION are synonymous and can be used interchangeably.

5. The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

6. Refer to Section 13.2.4, USE FOR DEBUGGING Statement, for syntax rules governing Format 2.

General Rules

Format 1

1. The designated procedures are 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.

2. After execution of a USE procedure, control is returned to the invoking routine.

3. 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 can refer to a USE statement or to the procedures associated with such a USE statement.

4. 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.

Format 2

5. Refer to Section 13.2.4, USE FOR DEBUGGING Statement, for general rules governing Format 2.

Format 3

6. The deadlock declarative specifies a deadlock exit address in the DMS/TX environment. If deadlock occurs, the original COBOL program may retain control instead of possibly being cancelled.

7. Procedure-names associated with a USE statement may be referenced in a different declarative statement or in a non-declarative procedure only with a PERFORM statement.

8. Within the USE procedure, an exit procedure can be specified by a GO TO statement. After execution of the USE procedure, control is returned to the user specified exit procedure or the entire program is cancelled if there is no exit procedure.

9. Within a USE procedure, do not execute any statement that would cause the execution of a USE procedure that had previously been invoked and had not yet returned control.

10. The user's program is cancelled upon unsuccessful setup of the deadlock declarative.

11. When the user's program is executed and deadlock occurs, any unsuccessful processing of the system deadlock handler, e.g., failure to ROLLBACK, will cause the user's program to be cancelled.

WRITE Statement -- for Consecutive Files

Function

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.

General Format

$$\underline{\text{WRITE}}\ \text{record-name}\ [\underline{\text{FROM}}\ \text{identifier-1}]$$
$$\left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix}\ \text{ADVANCING}\ \left\{\begin{matrix} \begin{Bmatrix}\text{identifier-2}\\ \text{integer}\end{Bmatrix}\begin{Bmatrix}\text{LINE}\\ \text{LINES}\end{Bmatrix} \\ \text{user-figurative-constant} \\ \underline{\text{PAGE}}\end{matrix}\right\}\right]$$

Syntax Rules

1. Record-name and identifier-1 must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division and can be qualified.

3. When identifier-2 is used in the ADVANCING phrase, it must be the name of an elementary integer data item.

4. Integer or the value of the data item referenced by identifier-2 can be 0.

5. The user-figurative constant must be defined as 2 bytes in the FIGURATIVE-CONSTANTS paragraph.

General Rules

1. The associated file must be open in the output, shared, or extend mode at the time of the execution of this statement.

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement was unsuccessful due to a boundary violation. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of the statement,

   <u>MOVE</u> identifier-1 <u>TO</u> 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.

   After execution of the WRITE statement is complete, 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. (Refer to General Rule 2.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, defined in the file control entry, to be updated.

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. 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.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. 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 is provided to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

   a. 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.

   b. If integer is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.

   c. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to General Rules 9a and 9b.

d. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to General Rules 9a and 9b.

e. 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.

f. If user-figurative constant is specified, advancing is controlled by a 2-byte write control character that is defined in the FIGURATIVE-CONSTANTS paragraph. If Bit 0 of Byte 1 is 0, then Bits 1-7 of Byte 2 indicate the number of lines to be skipped. If Bit 0 of Byte 1 is 1, then Byte 2 indicates top-of-form (HEX "01") or vertical tab (HEX "02").

10. WRITE is not valid for a consecutive file with RANDOM access.

11. When an attempt is made to write beyond the externally defined boundaries of a consecutive file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

    a. The value of the FILE STATUS data item, if described in the file control entry, is set to a value indicating a boundary violation.

    b. If a USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declaration procedure is executed.

    c. If a USE AFTER STANDARD EXCEPTION declaration is not explicitly or implicitly specified for the file, the result is undefined.

12. After the recognition of an end-of-reel or an end-of-unit of an output file that is contained on more than one physical reel/unit, the WRITE statement performs the following operations:

    a. The standard ending reel/unit label procedure

    b. A reel/unit swap

    c. The standard beginning reel/unit label procedure.

WRITE Statement -- for Indexed Files

Function

The WRITE statement releases a logical record for an OUTPUT, SHARED or INPUT-OUTPUT file.

General Format

```
WRITE record-name [FROM identifier]
   [TIMEOUT OF {data-name-1} [SECOND ]]
              {integer    } [SECONDS]]
              [HOLDER-ID IN data-name-2]
              {imperative-statement}]
              {NEXT SENTENCE       }]
         [INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division and can be qualified.

3. Data-name-1 must refer to an integer item no greater than 255.

4. Data-name-2 must be defined in the Working-Storage Section or Linkage Section and have a PICTURE of X(3).

5. The INVALID KEY phrase should be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the output, shared or I-O mode at the time of the execution of this statement.

2. The logical record released by the execution of the WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition. The logical record is also available to the program as a record of other files referenced in the same SAME RECORD AREA clause as the associated output file, as well as to the file associated with record-name.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of the statement,

   MOVE identifier 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.

   After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (Refer to General Rule 2.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of the WRITE statement causes the value of the FILE STATUS data item, if defined in the file control entry to be updated.

6. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

7. 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.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. Execution of the WRITE statement causes the contents of the record area to be released.

10. The value of the primary record key must be unique within the records in the file.

11. The data item specified as the primary record key and any alternate record keys must be set by the program to the desired values prior to the execution of the WRITE statement. (Refer to General Rule 3.)

12. If SEQUENTIAL or DYNAMIC access mode is specified for the file, records must be released to the operating system in ascending order of primary record key values.

13. If RANDOM access mode is specified, records may be released to the operating system in any program-specified order.

14. The INVALID KEY condition exists under any of the following circumstances:

    a. When SEQUENTIAL access mode is specified for a file opened in the output mode, and the value of the record key is not greater than the value of the record key of the previous record.

    b. When the file is opened in the output, shared or I/O mode, and the value of the primary record key is equal to the value of the primary record key of a record already existing in the file.

    c. When the file is opened in the output or I/O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file.

    d. When an attempt is made to write beyond the externally defined boundaries of the file.

15. 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 defined in the file-control entry, is set to a value indicating the cause of the condition. Execution of the program continues according to the rules stated in the INVALID KEY imperative-statement.

16. The alternate access paths (indexes) through which the record WRITTEN is to be available are determined by the record keys associated with record name. This list does not necessarily include all record keys associated with the file since equivalent alternate record keys may appear in some records, but not others.

17. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be nonunique only if the DUPLICATES phrase is specified for that data item. In this case the records are stored such that when records are accessed sequentially, the order of retrieval of those records is by the order of the primary key.

18. If the TIMEOUT phrase is specified and the READ cannot be completed in data-name-1 or integer seconds, then imperative-statement-1 is executed. If the number of seconds specified is 0, the timeout exit will immediately be taken if the READ cannot be completed.

19. If the HOLDER-ID phrase is specified in the TIMEOUT phrase, the logon initials of the user currently holding the resources is moved to data-name-2.

WRITE Statement -- for Relative Files

Function

The WRITE statement releases a logical record for an output or input-output file.

General Format

```
WRITE record-name [FROM identifier]

      [INVALID KEY imperative-statement]
```

Syntax Rules

1. Record-name and identifier must not reference the same storage area.

2. The record-name is the name of a logical record in the File Section of the Data Division and can be qualified.

3. The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

General Rules

1. The associated file must be open in the OUTPUT or I-O mode at the time of execution of this statement.

2. The logical record released by the execution of a WRITE statement is no longer available in the record area unless the associated file is named in a SAME RECORD AREA clause or the execution of the WRITE statement is unsuccessful due to an INVALID KEY condition. The logical record is also available to the program as a record of other files appearing in the same SAME RECORD AREA clause.

3. The results of the execution of the WRITE statement with the FROM phrase is equivalent to the execution of

   ```
   MOVE identifier TO record-name
   ```

   (in accordance with the rules for a 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 affect on the execution of this WRITE statement.

After execution of the WRITE statement, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be. (Refer to General Rule 2.)

4. The current record pointer is unaffected by the execution of a WRITE statement.

5. The execution of a WRITE statement causes the value of the FILE STATUS data item (if any) associated with the file to be updated.

6. The maximum record size for a file is established at the time the file is created and must not be subsequently changed.

7. 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.

8. The execution of the WRITE statement releases a logical record to the operating system.

9. The following rules apply to the placement of records into a file opened in output mode:

   a. If the access method is sequential, the WRITE statement causes a record to be released to the operating system. The first record will have a relative record number of one (1) and subsequent records released will have 2, 3, 4, etc. If the RELATIVE KEY data item has been specified in the file control entry for the file, the relative record number of the record just released is placed into the RELATIVE KEY data item by the operating system during executiuon of the WRITE statement.

   b. If the access mode is random or dynamic, the value of the RELATIVE KEY data item must be initialized prior to the execution of the WRITE statement. The key is initialized in the program with the relative record number to be associated with the record. That record is then released to the operating system by execution of the WRITE statement.

10. Records can be inserted into a file opened in I-O mode if the access mode is random or dynamic. The program must initialize the value of the RELATIVE KEY data item with the relative record number of the record to be inserted. Execution of the WRITE statement then causes the contents of the record area to be released to the operating system.

11. The INVALID KEY condition exists under the following circumstances:

    a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists.

   b. When an attempt is made to write beyond the externally defined boundaries of the file.

12. The execution of a WRITE statement is unsuccessful when an INVALID KEY condition is recognized. This condition also causes the contents of the record area to be unaffected and the FILE STATUS data item (if any) of the associated file to be set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated for the INVALID KEY condition. (Refer to Appendix E.)

CHAPTER 13
DEBUG FEATURES

13.1 VS DEBUG FACILITY

The VS provides a powerful interactive debugging facility which permits the programmer to examine and analyze an interrupted program and modify data values interactively from a workstation. Whenever a program is interrupted, debug processing can be entered by pressing PF10 (ENTER DEBUG PROCESSING) from the modified Command Processor menu. (A program can be interrupted by the operator at any time by pressing HELP; a program can also be interrupted by the system with a program check if a terminal execution error occurs.)

Debug processing permits the programmer to inspect the object program and data in memory, modify data values, and set program traps. The programmer can also inspect and modify the Program Control Word (PCW), General-Purpose Registers, and Floating-Point Registers. Execution of the interrupted program can be resumed at any point during debug processing.

An additional feature of the debug processor is its symbolic debug facility. If the program was compiled with symbolic debug information (by selecting SYMB=YES as a COBOL compiler option), the symbolic debug facility is automatically available when debug processing is entered. Symbolic debug permits the programmer to inspect and modify data fields of up to 65,535 bytes by symbolic data name rather than the memory address, thus eliminating the need for a program map (PMAP) or data map (DMAP).

The Inspect/Modify function of the Symbolic Debugger allows the programmer to inspect and modify memory, program registers, or the PCW. Symbolic debug also displays a "window", containing seven lines from the source program listing, which permits the programmer to scan backward and forward through the source listing. If the program is interrupted by a program check, the window displays the source line containing the verb that was executing when the error occurred, precisely identifying the immediate source of the problem.

NOTE

The window displaying the source listing is available only if the source listing was placed in a print HOLD file and SOURCE = YES was specified at the time of compilation.

13.2 DISPLAYING SUBSCRIPTED AND QUALIFIED DATA NAMES

48 characters are available on the debugger screen for entering a data name when the programmer wishes to examine the contents of a particular storage location. Because of this, there is limited support for Subscripted and Qualified data names. If an identifier requires subscripting or qualifiers to achieve a unique reference, the fully qualified name of the identifier must be specified within that 48 character limit. If the name cannot be specified within the 48 characters, the programmer must examine the desired storage location by referring to the program's PMAP and DMAP.

For example, assume that a program contains the two records described below:

```
01  TABLE1.
    05  LEVEL1          OCCURS 3 TIMES.
        10  ELEMENT     PIC XX.

01  TABLE2.
    05  LEVEL1          OCCURS 3 TIMES.
        10  ELEMENT     PIC XX.
```

If the programmer wants to examine the contents of the 3rd occurrence of ELEMENT in TABLE2, the following must be specified on the Inspect/Modify screen:

```
ELEMENT.LEVEL1.TABLE2
```

The programmer then presses ENTER, followed by a value of 3 for the subscript, followed by another ENTER. If a data name is unique, it must not be qualified on the Inspect/Modify screen. The symbolic debug facility is explained in the VS Program Development Tools Reference.

13.3 ANSI DEBUG MODULE

Because the VS interactive debug facility is significantly more powerful and convenient than the COBOL debug features, it is not anticipated that these features will be widely used in COBOL programs written for the VS. There can, however, be special cases in which some COBOL debug features will prove useful; for this reason, the COBOL debug features are supported in VS COBOL.

The features of the COBOL language that support the debug module are:

1. A special register -- DEBUG-ITEM
2. A compile time switch -- WITH DEBUGGING MODE
3. An object time switch
4. A USE FOR DEBUGGING statement
5. Debugging lines.

To these features, VS COBOL adds the READY TRACE and RESET TRACE statements.

13.3.1 DEBUG-ITEM

The reserved word DEBUG-ITEM is the name for a special, automatically generated register that supports the debugging facility. Only one DEBUG-ITEM is allocated per program. The names of the subordinate data items in DEBUG-ITEM are also reserved words.

13.3.2 Compile Time Switch--WITH DEBUGGING MODE

The WITH DEBUGGING MODE clause is written as part of the SOURCE-COMPUTER paragraph. This clause serves as a compile time switch over the debugging statements written in the program.

Function

The WITH DEBUGGING MODE clause indicates that all debugging sections and all debugging lines are to be compiled. If this clause is not specified, all debugging lines and sections are compiled as if they were comment lines.

General Format

```
SOURCE-COMPUTER.   WANG-VS [WITH DEBUGGING MODE].
```

General Rules

1. If the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, any USE FOR DEBUGGING statements, all associated debugging sections, and any debugging lines are compiled as if they were comment lines.

2. If the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph of the Configuration Section of a program, all USE FOR DEBUGGING statements and all debugging lines are compiled.

13.3.3 Object Time Switch

An object time switch dynamically activates the debugging code inserted by the compiler. This switch cannot be addressed in the program; it is controlled outside the COBOL environment. If the switch is ON, all the effects of the debugging language written in the source program are permitted. If the switch is OFF, all the effects described in the USE FOR DEBUGGING Statement (refer to Section 13.2.4) are inhibited. Recompilation of the source program is not required to provide or remove this facility, although performance can be significantly degraded when this debugging code is included in operational programs.

The object time switch has no effect on the execution of the object program if the WITH DEBUGGING MODE clause was not specified in the source program at compile time.

13.3.4 USE FOR DEBUGGING Statement

Function

The USE FOR DEBUGGING statement is used in the Procedure Division and identifies the user items that are to be monitored by the associated debugging section.

General Format

```
section-name SECTION.

     USE FOR DEBUGGING ON { procedure-name-1      [procedure-name-2] ... }
                          { ALL PROCEDURES                              }
```

Syntax Rules

1. Debugging section(s), if specified, must appear together immediately after the DECLARATIVES header.

2. Except in the USE FOR DEBUGGING statement itself, there must be no reference to any nondeclarative procedure within the debugging section.

3. Statements appearing outside of the set of debugging sections must not reference procedure names defined within the set of debugging sections.

4. Except for the USE FOR DEBUGGING statement itself, statements appearing within a given debugging section can reference procedure names defined within a different USE procedure only with a PERFORM statement.

5. Procedure names defined within debugging sections must not appear within USE FOR DEBUGGING statements.

6. Any given procedure name can appear in only one USE FOR DEBUGGING statement and can appear only once in that statement.

7. The ALL PROCEDURES phrase can appear only once in a program.

8. When the ALL PROCEDURES phrase is specified, no other USE FOR DEBUGGING sections can appear in the program.

General Rules

1. In the following general rules all references to procedure-name-1 apply equally to procedure-name-2.

2. Automatic execution of a debugging section is not caused by a statement appearing in a debugging section.

3. When procedure-name-1 is specified in a USE FOR DEBUGGING statement, that debugging section is executed:

   a. Immediately before each execution of the named procedure

   b. Immediately after the execution of an ALTER statement that references procedure-name-1.

4. The ALL PROCEDURES phrase causes the effects described in General Rule 3 to occur for every procedure name in the program, except those appearing within a debugging section.

5. In the case of a PERFORM statement, which causes iterative execution of a referenced procedure, the associated debugging section is executed once for each iteration.

   Within an imperative statement, each individual occurrence of an imperative verb identifies a separate statement for the purpose of debugging.

6. Associated with each execution of a debugging section is the special register DEBUG-ITEM, which provides information about the conditions that caused the execution of a debugging section. DEBUG-ITEM has the following implicit description:

```
01  DEBUG-ITEM.
02  DEBUG-LINE      PICTURE IS X(6).
02  FILLER          PICTURE IS X VALUE SPACE.
02  DEBUG-NAME      PICTURE IS X(30).
02  FILLER          PICTURE IS X(19) VALUE SPACE.
02  DEBUG-CONTENTS  PICTURE IS X(30).
```

7. Prior to each execution of a debugging section, the contents of the data item referenced by DEBUG-ITEM are space-filled. The contents of data items subordinate to DEBUG-ITEM are then updated, according to the following general rules, immediately before control is passed to that debugging section. The contents of any data item not specified in the following general rules remains spaces.

   Updating is accomplished in accordance with the rules for the MOVE statement. The sole exception is the move to DEBUG-CONTENTS where the move is treated exactly as if it were an alphanumeric to alphanumeric elementary move with no conversion of data from one form of internal representation to another.

8. The contents of DEBUG-LINE is the source statement line number of the procedure name or statement.

9. DEBUG-NAME contains the first 30 characters of the name that causes the debugging section to be executed.

10. If the first execution of the first nondeclarative procedure in the program causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the line number of the first statement of that procedure.

    b. DEBUG-NAME contains the name of that procedure.

    c. DEBUG-CONTENTS contains START PROGRAM.

11. If a reference to procedure-name-1 in an ALTER statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the ALTER statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains the applicable procedure name associated with the TO phrase of the ALTER statement.

12. If the transfer of control associated with the execution of a GO TO statement causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the GO TO statement whose execution transfers control to procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

13. If the transfer to control from the control mechanism associated with a PERFORM statement caused the debugging section associated with procedure-name-1 to be executed, the following conditions exist:

    a. DEBUG-LINE identfies the PERFORM statement that references procedure-name-1.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains PERFORM LOOP.

14. If procedure-name-1 is a USE procedure that is to be executed the following conditions exist:

    a. DEBUG-LINE identifies the statement that causes execution of the USE procedure.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains USE PROCEDURE.

15. If an implicit transfer of control from the previous sequential paragraph to procedure-name-1 causes the debugging section to be executed, the following conditions exist:

    a. DEBUG-LINE identifies the previous statement.

    b. DEBUG-NAME contains procedure-name-1.

    c. DEBUG-CONTENTS contains FALL THROUGH.

13.3.5 Debugging Lines

A debugging line is any line with a 'D' in the indicator area. Any debugging line that consists solely of spaces from Margin A to Margin R is considered the same as a blank line.

The contents of a debugging line must be such that a syntactically correct program is formed with or without the debugging lines being considered as comment lines.

A debugging line is considered to have all the characteristics of a comment line if the WITH DEBUGGING MODE clause is not specified in the SOURCE-COMPUTER paragraph.

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 cannot be broken across two lines.

A debugging line is only permitted in the program after the OBJECT-COMPUTER paragraph.

13.3.6 READY TRACE and RESET TRACE Statements

Function

The trace statements allow selective recording of the flow of execution of paragraphs and sections within a program. These statements can appear anywhere in the COBOL program.

General Format

$$\boxed{\left\{\begin{matrix}\underline{\text{READY}}\\\underline{\text{RESET}}\end{matrix}\right\}\quad\underline{\text{TRACE}}}$$

Syntax Rules

1. After a READY TRACE statement is executed (each time execution of a paragraph or section begins) the paragraph or section name will be displayed on a file with the parameter reference name of TRACE.

2. The execution of a RESET TRACE statement terminates execution flow recording.

APPENDIX A
RESERVED WORDS

ACCEPT
ACCESS
ADD
ADVANCING
AFTER
ALARM
ALL
ALPHABETIC
ALSO
ALTER
ALTERED
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BINARY
BLANK
BLOCK
BLOCKS
BOTTOM
BUFFER
BY

CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMUNICATION
COMP
COMPRESSED
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
CONVERSION
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
CURSOR

DATA
DATABASE-NAME
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEADLOCK
DEBUG-CONTENTS
DEBUG ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELETION

DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DISPLAY-WS
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
ELSE
EMI
ENABLE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERASE
ERROR
ESI
EVERY
EXCEPTION
EXCLUSIVE
EXIT
EXTEND
EXTENRION-RIGHTS

FAC
FD
FIGURATIVE-CONSTANTS
FILE
FILE-CONTROL
FILENAME
FILLER
FINAL
FIRST
FOOTING
FOR
FREE
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
HOLDER-ID

IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
INVOKE
I-O
I-O CONTROL
IS

JUST
JUSTIFIED

KEY
KEYS

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LIST
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MESSAGE
MODE
MODIFIABLE
MODIFY
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
NODISPLAY
NO-MOD
NOT
NUMBER
NUMERIC

OBJECT
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ONLY
OPEN
OPTIONAL
OR
ORDER
ORDER-AREA
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PFKEY
PFKEYS
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINTING
PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROTECT

QUEUE
QUOTE
QUOTES

RANDOM
RANGE
RD
READ
READY
RECEIVE
RECORD
RECORDS
RECOVERY-BLOCKS
RECOVERY-STATUS
REDEFINES
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTART
RETRIEVAL
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROLL
ROLLBACK
ROUNDED
ROW
RUN

SAME

SD
SEARCH
SECOND
SECONDS
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SETTING
SHARED
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SWITCH-1
SWITCH-2
SWITCH-3
SWITCH-4
SWITCH-5
SWITCH-6
SWITCH-7
SYMBOLIC
SYNC
SYNCHRONIZED

TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIMEOUT
TIMES
TO
TOP
TRACE
TRAILING
TYPE

UNIT
UNSTRING
UNTIL
UP
UPDATE
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING
VOLUME

WANG-VS
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

+
-
*
/
**
=

APPENDIX B
COBOL COMPILER OPTIONS

The following options are provided by the VS COBOL compiler.

BIGPGT ("Compiling very large program?")

The BIGPGT (Big Program Global Table) option is used to instruct the compiler to follow a special set of procedures when compiling a very large program which exceeds a certain critical size. Since the critical size is not absolute, but can depend on a number of factors, the only way to determine whether this option is necessary is to compile the program initially with BIGPGT = NO. If the program is too large for normal compilation techniques, the compiler produces a diagnostic message instructing you to recompile using the BIGPGT option. To select this option, set BIGPGT = YES. Occassionaly your program may still be too large. In this instance, set BIGPGT=3RD.

DMAP ("Generate data division map?")

If DMAP = YES, a Data Division Map (DMAP) is produced. A DMAP lists the names and attributes of all data items defined in the program, along with their locations in the static area in memory. The DMAP contains 13 labeled columns.

NAME	-	Lists all data names defined in the program, in alphabetical order.
LVL	-	Lists the level number associated with each data name.
USAGE	-	Lists the usage specified for each data item. For example, DSP-AN means DISPLAY-alpha-numeric, DSP-NM means DISPLAY-numeric, DSP-GR means DISPLAY-group-item.)
DISPL	-	Lists the displacement, or offset, of the first byte of each data item. This value is added to the base address of the static area (in Register 14) to yield the actual address of the data item in memory.
LENGTH	-	Lists the length (in bytes) of each data item.
PICTURE	-	Lists the PICTURE defined for each data item.

OCCUR	-	Lists the number of occurrences of a data item as defined in an OCCURS clause.
R	-	Indicates that the data description entry for the data name contains a REDEFINES clause.
O	-	Indicates that the data name is a group name specified in an OCCURS clause.
V	-	Indicates that the data description entry for the data name contains a VALUE IS clause.
J	-	Invalid for this release of the compiler, since JUSTIFIED is treated as a comment.
B	-	Indicates that the data description entry for the data name contains a BLANK WHEN ZERO clause.
SIGN	-	Indicates that the data description entry for the data name contains a SIGN clause. The possible values in the sign column and their meanings are:

S	Sign is separate
T	Sign is trailing
L	Sign is leading
ST	Sign is separate and trailing
SL	Sign is separate and leading

The default provided is ST if leading, trailing and/or separate are not provided in the program.

The DMAP also lists all literals used in the program. Literals are compiled as part of the program code itself, and are not stored along with variable data in the static section.

FIPS ("FIPS Flagger?")

"FIPS" is the acronym for Federal Information Processing Standard. If FIPS = LOW, messages are generated informing the user which source statements contain syntax that is either nonstandard (Wang extensions) or above the low level for Federal Standard COBOL. If FIPS = LI, messages are generated informing the use which source statements contain syntax that is either Wang extensions or above the Low Intermediate level for Federal Standard COBOL. FIPS = NO is the default. Refer to Appendix I, A Comparison of VS, ANSI, and FIPS COBOL Standards.

FLAG ("Lowest severity error printed?")

FLAG specifies the lowest level of error severity that will cause the compiler to print a diagnostic message. Any error with a severity code greater than or equal to the specified FLAG value will cause the compiler to print a diagnostic message.

LINES ("Number of lines per page?")

LINES specifies the number of lines to be printed on each page when a source listing, PMAP, and DMAP are printed.

LOAD ("Generate object module?")

If LOAD = YES, the compiler creates an object program in VS object program format, and stores it in an output file. If LOAD = NO, no object program is produced. (In this case, the compiler does not display an output definition screen to name the output file). The NO option may be selected if only a source listing is desired.

LOWER ("Lowercase D&R input")

If LOWER = YES, the default FAC generated for alphanumeric fields to be entered in response to a DISPLAY AND READ statement will be hex 80 and will accept both uppercase and lowercase input. If LOWER = NO, the default FAC is hex 81 and causes lowercase input to shift to uppercase. This feature allows the replacement of the lowercase characters with a non-Latin character set. Such replacement is useful for international applications where programs are written in languages such as Greek, Japanese, etc.

PMAP ("Print generated code?")

If PMAP = YES, the compiler produces a PMAP (program map) for the compiled program. The PMAP contains the machine instructions generated by each COBOL verb, with the address of each instruction. A PMAP consists of five basic columns.

Column 1 - COBOL verbs and line numbers.
Column 2 - Address and object code.
Column 3 - COBOL paragraph names.
Column 4 - Assembler instructions.
Column 5 - Comments.

A map of the static area immediately follows the PMAP, beginning with the word STATIC in Column 1.

In the static area map, four main columns are used.

Column 1 -	Line number in source program on which a data name is defined, and description of data type. For data items, type is always VALUE. For data files, type is the file name to which the data name is assigned in the FILE-CONTROL paragraph. The first entry in the static area map contains the line number of the last line in the program and the descriptive word, STATIC.
Column 2 -	Location of first byte of value in static area, specified as offset from base address in Register 14.

Column 3 - Comments generated by compiler.

Column 4 - Assembler instruction generated by data item definition in source program.

Column 5 - Data name specified in source program.

NOTE

To generate a PMAP for a selected section of code rather than for the entire program, the following method can be used:

1. Insert $P in Columns 7 and 8 of the source program just prior to the first line of code in the Procedure Division for which the PMAP is to be generated.

2. Insert $PX in Columns 7 through 9 of the source program immediately following the last line of code in the Procedure Division for which the PMAP is to be generated.

3. Compile the program with compiler option PMAP = NO.

SEPSGN ("Separate character for sign?")

The SEPSGN option is used to specify whether or not the sign is to be included in the byte count or appear as a separate byte. The default condition, SEPSGN = YES, produces the sign in a separate, trailing byte. By specifying SEPSGN = NO, the sign is included. Any optional clauses coded with the SIGN clause take precedence over the SEPSGN option, which provides default sign characteristics when they are not specifically coded in the program.

SEQ ("Sequence check source?")

If SEQ = YES, the compiler performs a check to ensure that the sequence numbers (line numbers) of the source program lines are in order. Otherwise, no sequence check is performed. The default value is NO.

SOURCE ("Create source listing?")

If SOURCE = YES, the compiler produces a source listing of the compiled program, with accompanying diagnostics. If SOURCE = NO, no source listing is produced. (Diagnostics are produced in either case).

SPACE ("Single or double spaced?")

SPACE specifies the spacing between lines on the printed listings. The options are single space (SPACE = 1) or double space (SPACE = 2).

STOP ("Code generation stop level?")

STOP specifies the lowest level of error severity that will cause the compiler to abort the compilation. Any error with a severity code greater than or equal to the specified STOP value will terminate the compilation (no object program is produced).

> **Note**
>
> The STOP level should not be set above eight.

SUBCHK ("Check subscripts at run time?")

If SUBCHK = YES, the compiler generates special code that checks the ranges of subscripts during program execution, and causes a program check (execution interruption) if a subscript exceeds its defined limit. Otherwise, no check is performed on subscripts during execution.

SYMB ("Include symbolic debug data?")

If SYMB = YES, the compiler inserts symbolic debug information in the object program. If SYMB = NO, this information is not inserted, and the symbolic debug facility cannot be used to debug the object program at runtime.

TRUNC ("Truncate COMP data to picture?")

TRUNC is used to specify that truncation is to occur in certain special cases when a larger numeric data item is moved into a smaller numeric data item. If the receiving field does not have enough bytes to contain all the digits from the sending field, truncation of one or more high-order digits is automatic; this situation is not altered by TRUNC. In the special case involving packed decimal fields, however, truncation can be specified even if the receiving field has enough bytes to accommodate the sending field. This case occurs when the receiving field has an even number of digits (e.g. PICTURE S9(6) COMP). Each digit occupies a half-byte in packed decimal format. Since the low-order half-byte is taken by the sign digit, the number of half-bytes available for digits is always odd. If the number of digits is even, the high-order half-byte in the field is unused, and is set to 0. For example, the packed decimal field generated by a data item with a PICTURE of S9(6) COMP containing the value 123456 looks like 0123456F, where each pair of digits occupies one byte, and the 'F' is the sign digit. Note that although the PICTURE specifies six digits, the resulting value occupies four bytes in memory. If TRUNC = YES, and a numeric data item of seven digits (or more) is moved into this field, the leftmost digit is truncated, even though the receiving field contains an available half-byte to store it. If TRUNC = NO (the default value), the leftmost digit is placed in the high-order half-byte of the receiving field.

Note that if the receiving field contains more than seven digits, any digits beyond the seventh are truncated automatically, irrespective of the value of TRUNC.

XREF ("Print cross-reference?")

If XREF = YES, a cross-reference listing of all data names referenced in the program is produced. The cross-reference listing contains all referenced data names and literals (listed alphabetically), the source program line on which each is initially defined, and all other lines on which each data name is referenced.

APPENDIX C
FIELD ATTRIBUTE CHARACTERS

C.1 FACs AND FAC VALUES

An important part of the workstation screen is its ability to be divided into fields. Fields, although not visible, are important as they control the operation of the workstation both from the keyboard and in communication with the computer. A field is defined as all of the characters from one Field Attribute Character to the next or to the end of the row. Field Attribute Characters take a location of the display. They always display as a blank no matter what their value. Each row is considered to have a Field Attribute Character just before the first character in the row and just after the last character in the row. These Field Attribute Characters do not take up space on the screen. They have a default value of hexadecimal "8C". These default Field Attribute Characters allow the use of 80 character lines. Any location of the screen can be a Field Attribute Character. The characters from the start of each row to the first Field Attribute Character in the row display in low intensity, as though there were a low intensity Field Attribute Character before the first character in the row. The Field Attribute Character controls the mode of display for all following characters until another Field Attribute Character or until the end of row is encountered.

All the characters of a field have the same attributes, which are defined by the Field Attribute Character preceding the field. The possible attributes are defined in Table C-1.

NOTE

In the following table and elsewhere, bits are numbered consecutively from 0. Thus the first bit is Bit 0; the second bit is Bit 1, and so on.

Table C-1. Field Attribute Character Values

BIT	Field Description
0	Must be 1 (this is the Field Attribute Character indicator)
1	Modified data tag = 0 Field has not been modified = 1 Field has been modified
2	Underscore option = 0 Field is not underscored = 1 Field is underscored
3, 4	Display control = 00 Intensified display = 01 Low intensity display = 10 Blinking display = 11 Nondisplay
5	Protect bit = 0 Modifiable field = 1 Protected field
6, 7	Valid keyable data specification = 00 Alphanumeric uppercase and lowercase = 01 Alphanumeric uppercase shift = 10 Numeric only = 11 Reserved

C.2 DISPLAY CHARACTERISTICS FOR WORKSTATION SCREEN FIELDS

The possible display characteristics for fields of the workstation screen are:

Intensified display - The characters in this field will be displayed in higher intensity than those in a low intensity display field.

Low intensity display - The characters of this field will be displayed on the screen at normal intensity.

Blinking display - The characters in this field will be displayed alternately in intensified display and normal display mode. The display will change modes at a fixed rate of about 3 times a second.

Nondisplay - The characters in the field will not be displayed on the screen. The field will look as if it were all blanks.

Modifiable - Any or all of the positions of this field can be changed by the operator.

Protected - No position of this field can be modified by the operator.

Alphanumeric - Allows keying in of any character on the keyboard.

Uppercase shift - Letters will be displayed and stored only as uppercase. This is without regard to whether SHIFT or LOCK are depressed. All other keys will respond to the SHIFT and LOCK keys as they normally would.

Numeric only - Only the characters 0-9, decimal point (.), or minus (-) may be entered into this field or the keystroke is ignored and the alarm sounds.

Reserved - This is not a valid combination at this time. It is intended for addition of later options. Its use may result in unpredictable results.

C.3 LIST OF FIELD ATTRIBUTE CHARACTERS

BRIGHT	MODIFY	ALL	NOLINE	80
BRIGHT	MODIFY	UPPERCASE	NOLINE	81
BRIGHT	MODIFY	NUMERIC	NOLINE	82
BRIGHT	PROTECT	ALL	NOLINE	84
BRIGHT	PROTECT	UPPERCASE	NOLINE	85
BRIGHT	PROTECT	NUMERIC	NOLINE	86
DIM	MODIFY	ALL	NOLINE	88
DIM	MODIFY	UPPERCASE	NOLINE	89
DIM	MODIFY	NUMERIC	NOLINE	8A
DIM	PROTECT	ALL	NOLINE	8C
DIM	PROTECT	UPPERCASE	NOLINE	8D
DIM	PROTECT	NUMERIC	NOLINE	8E
BLINK	MODIFY	ALL	NOLINE	90
BLINK	MODIFY	UPPERCASE	NOLINE	91
BLINK	MODIFY	NUMERIC	NOLINE	92
BLINK	PROTECT	ALL	NOLINE	94
BLINK	PROTECT	UPPERCASE	NOLINE	95
BLINK	PROTECT	NUMERIC	NOLINE	96
BLANK	MODIFY	ALL	NOLINE	98
BLANK	MODIFY	UPPERCASE	NOLINE	99
BLANK	MODIFY	NUMERIC	NOLINE	9A
BLANK	PROTECT	ALL	NOLINE	9C
BLANK	PROTECT	UPPERCASE	NOLINE	9D
BLANK	PROTECT	NUMERIC	NOLINE	9E
BRIGHT	MODIFY	ALL	LINE	A0
BRIGHT	MODIFY	UPPERCASE	LINE	A1
BRIGHT	MODIFY	NUMERIC	LINE	A2
BRIGHT	PROTECT	ALL	LINE	A4
BRIGHT	PROTECT	UPPERCASE	LINE	A5
BRIGHT	PROTECT	NUMERIC	LINE	A6
DIM	MODIFY	ALL	LINE	A8
DIM	MODIFY	UPPERCASE	LINE	A9
DIM	MODIFY	NUMERIC	LINE	AA
DIM	PROTECT	ALL	LINE	AC
DIM	PROTECT	UPPERCASE	LINE	AD
DIM	PROTECT	NUMERIC	LINE	AE
BLINK	MODIFY	ALL	LINE	B0
BLINK	MODIFY	UPPERCASE	LINE	B1
BLINK	MODIFY	NUMERIC	LINE	B2
BLINK	PROTECT	ALL	LINE	B4
BLINK	PROTECT	UPPERCASE	LINE	B5
BLINK	PROTECT	NUMERIC	LINE	B6

BLANK	MODIFY	ALL	LINE	B8
BLANK	MODIFY	UPPERCASE	LINE	B9
BLANK	MODIFY	NUMERIC	LINE	BA
BLANK	PROTECT	ALL	LINE	BC
BLANK	PROTECT	UPPERCASE	LINE	BD
BLANK	PROTECT	NUMERIC	LINE	BE

ALL	- Uppercase, lowercase and alphanumeric characters
NUMERIC	- Digits 0 - 9, decimal and minus sign
LINE	- Underlines displayed character
NOLINE	- Character not underlined
UPPERCASE	- Alphanumeric uppercase

APPENDIX D
WORKSTATION SCREEN ORDER AREA

D.1 USE OF THE ORDER AREA

The order area bytes are used to specify screen control actions and to indicate row/column address for data transfer to/from the display.

The content of the order area and the interpretation of the fields in the area is different for a READ and a REWRITE. The following chart illustrates the differences:

Byte	On READ	On REWRITE
0	Row number	Row number
1	Reserved	Write Control Character (WCC)
2	Cursor Column Address	Cursor Column Address (if cursor bit set in WCC)
3	Cursor Row Address	Cursor Row Address

The row number is the starting screen line number for data transfer. The byte is always set (by the system) to the value specified for the data name of the RELATIVE KEY phrase.

D.1.1 Interpretation of the Write Control Character

The Write Control Character (WCC) is interpreted as follows:

Table D-1. Write Control Character

Bit	Explanation
0	UNLOCK keyboard (LOCK if zero)
1	Sound alarm
2	Position cursor
3	Roll down
4	Roll up
5	ERASE rest of modifiable fields
6	ERASE and protect rest of screen
7	Reserved (must be zero)
WCC codes:	If the specified bit is 1, the noted action will take place.

Programs control activity at the workstation via commands contained in the workstation record. Legal commands include:

1. Unlock keyboard
2. Lock keyboard
3. Sound alarm
4. Position cursor
5. Row up/down
6. Erase screen

Unlock the keyboard (Hex "80")

After the record is written to the screen, and after sounding of the alarm, if this command is specified, the AID character will be set to blank and the keyboard will then be unlocked.

If the bit is zero, the keyboard will be locked before any data is transmitted to the workstation. If the keyboard is already locked, setting this bit to zero will not change the status of the keyboard. The normal method to get the keyboard locked is to wait for the operator to press one of the computer communication keys. If the bit is zero and the keyboard is locked, the AID character will not change. However, if the command locks the keyboard, the AID character will be set to an apostrophe (Hex "21").

Sound alarm (Hex "C0")

This will cause the alarm to sound before the data is transmitted to the screen.

Position the cursor (Hex "A0")

After data is transferred to the screen, the cursor will be positioned. If the cursor column address position of the order area is Hex "00", the cursor will be positioned to the first modifiable location located at or following the start of the row just written. (This option will act as if a TAB key were pressed with the starting position of the cursor one location before the start of the row just written, except that if there are no modifiable fields on the rest of the screen, the cursor will be positioned to the first location in the specified line.) If the cursor column address is not zero, it will be interpreted as the column within the row just written where the cursor is to be positioned. (Note this has values of 1 - 80.)

Roll Down (Hex "10")

This will cause the bottom line of the screen to be lost and each line above it to be copied into the next lower line. This will proceed until the row specified in the order area has been copied. The specified row will then be set to blanks and the WRITE will proceed. An attempt to write more than one line in a single command with "roll down" will result in an error.

Roll Up (Hex "08")

This will cause the row specified in the order area to be lost and each line below it to be copied into the next higher line (e.g., Line 1 will be replaced by the contents of Line 2). This will proceed until the last row of the screen has been copied. The last row will then be set to blanks and the WRITE will proceed on the last line of the screen. An attempt to write more than one line in a single command with "roll up" specified will result in an error.

Erase rest of modifiable fields (Hex "04")

Before data is transferred to the screen, all modifiable locations starting at the row address specified in the order area to the end of the screen will be set to blanks.

Erase and protect rest of screen (Hex "02")

All locations of the screen at and after the row address specified in the order area to the end of the screen are set to the hex "8C" before the data is transferred to the screen.

This causes there to be no modifiable locations after the data that is written.

D.1.2 Interpretation of the Order Area on a READ

The first byte of the order area is inspected before the data transfer and is used to specify the starting row number for the read. If this row number is not in the range 1 - 24 (binary), the command will be terminated. This byte is not changed by the READ.

The third and fourth bytes of the order area are set by the READ to the address of the cursor at the time of the read. The first byte of the two will contain the row number (1 - 24 binary) and the second will contain the column number (1 - 80 binary) of the current cursor location. These two bytes are not inspected before the read.

The second byte of the order area is not inspected or modified, but is to be supplied as binary zeroes for compatibility with future options.

D.1.3 Interpretation of the Order Area on a REWRITE

Neither the order area nor the mapping area is changed on a REWRITE. The first byte of the order area on a REWRITE is interpreted as the row number at which the REWRITE is to start. If this row number is not in the range 1 through 24, the command will be terminated.

The second byte of the order area is interpreted as the Write Control Character (WCC). If the "set cursor address" bit is set in the WCC, the next two bytes of the order area are interpreted as a cursor column and row address.

If the "set cursor address" bit is set in the WCC and the cursor address byte is set to a value between 1 and 80 inclusive, after the REWRITE completes the cursor will be positioned to that column. If this option is taken, the cursor will be positioned in the row specified by the fourth byte of the order area. If the "set cursor address" bit is set in the WCC and the cursor address byte is set to a value of zero (Hex "00"), this will act as if the cursor were positioned one location before the first location in the specified row and the TAB key were struck. If there are no modifiable positions on the screen after the REWRITE command, the cursor would then be positioned to the first location in the specified row.

If the "set cursor address" bit is set in the WCC and the cursor column address byte has a value other than 0 - 80, or the cursor row address is a value other than 1 - 24, the command will be terminated.

D.2 MAPPING AREA CONTROL

The mapping area is defined in the program in the area immediately after the order area. The mapping area contains the data transmitted either to or from the screen. The first location of the mapping area corresponds to the first character of the row specified in the first byte of the order area. Byte 81 of the mapping area would correspond to the first byte of the next row. If the starting row number and the length of the mapping area are such that locations in the mapping area would extend past the end of the screen, the command will be terminated. Note that, although the mapping area's first position will always correspond to the start of a row, the only restriction on the end of the mapping area is that it not extend past the end of the screen. It is recommended that the mapping area be the full screen (1920 bytes) as smaller quantities result in noticeable flicker.

No mapping area need be supplied for a 4-byte READ or REWRITE (order area only).

D.3 DISPLAYABLE CHARACTERS

The CRT screen is capable of displaying 24 rows of 80 characters each. Every position of the screen is capable of displaying any of the possible displayable characters. A special symbol called a cursor, is displayed beneath a character position to indicate where the next character entered from the keyboard will be stored. The cursor is displayed on the screen when data can be keyed by the operator. If it is not displayed, the keyboard is locked. This has no effect on the display or the computer interface with the workstation, but does directly effect the data entry from the keyboard. Each position of the screen is referenced by its row and column numbers. The first position of the screen (upper left corner) is called Row 1, Column 1. The columns are numbered from left to right and the rows from top to bottom. Position two is the second character from the left on the first line.

The characters of the following table constitute all of the displayable characters:

Table D-2. Displayable Characters

	(space)
!	(exclamation point)
"	
#	
$	
%	
&	
'	
(	
)	
*	
+	
-	
.	
/	
0 1 2 3 4 5 6 7 8 9	
:	
;	
<	
=	
>	
]	
@	
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	
[	(open bracket)
	(back slash)
]	(close bracket)
	(circumflex)
_	(under bar)
a b c d e f g h i j k l m n o p q r s t u v w x y z	
	(lozenge)
	(solid character)

APPENDIX E
FILE STATUS KEY VALUES

E.1 I-O STATUS

If the FILE STATUS clause is specified in a file control entry, a value is placed into the specified 2-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement and before any applicable USE procedure is executed, to indicate to the COBOL program the status of that input/output operation.

E.2 CONSECUTIVE FILES

E.2.1 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' - Indicates Successful Completion
'1' - Indicates AT END
'2' - Indicates INVALID KEY
'3' - Indicates Permanent Error
'6' - Indicates an I/O Cancel Condition
'7' - Indicates a TIMEOUT Condition
'8' - Indicates a Shared Mode Error
'9' - Indicates Special Conditions.

The meanings of the above indications are as follows:

'0' - Successful Completion. The input/output statement was successfully executed.

'1' - AT END. The sequential READ statement was unsuccessful 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 an attempt to read a record with a relative record key higher than that associated with the last record in the file.

'3' - Permanent Error. The input/output statement was unsuccessfully executed as the result of a boundary violation or as the result of an input/output error, such as data check, parity error, or transmission error.

'6' - An input/output cancel condition has occurred, but the cancel messages have been suppressed as a result of a user request.

'7' - A TIMEOUT condition exists after an attempt to execute an input/output statement in shared mode.

'8' - The input/output statement was unsuccessfully executed as a result of an error occurring while the file was opened in shared mode.

'9' - Special Conditions. The input/output statement was unsuccessfully executed as a result of a condition, such as an invalid function or function sequence, that is unique to the file or device.

E.2.2 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. For consecutive files other than workstation files, this character will contain a value as follows:

1. When Status Key 1 contains a value of '0' indicating successful completion, Status Key 2 contains a value of '0' indicating that no further information concerning the input/output operation is available.

2. When Status Key 1 contains a value of '1' indicating an AT END condition, Status Key 2 can contain a value of '0' or '1', with the following meanings.

   '0' - Indicates that no further information concerning the input/output operation is available.

   '1' - Indicates that an end of volume has been reached for a tape file and that the user-program indicates no automatic volume switch is desired.

3. When Status Key 1 contains a value of '2' indicating an INVALID KEY, Status Key 2 contains a value of '3' indicating that the supplied record number is equal to or greater than the highest record number in the file.

4. When Status Key 1 contains a value of '3' indicating a permanent error condition, Status Key 2 can contain a value of '0' or '4', with the following meanings:

'0' - Indicates a hardware error occurred when an input/output operation was attempted. Refer to SVC CHECK in the VS Operating System Services Reference. This value is not returned for program related errors.

'4' - Indicates that an attempt has been made to write beyond the externally defined boundaries of a file.

5. When Status Key 1 contains a value of '6' indicating an input/output cancel condition with cancel messages suppressed, Status Key 2 contains a value of '0' indicating that no further information is available. To set the error message flag, process the file in nonshared mode.

6. When Status Key 1 contains a value of '7' indicating that a TIMEOUT condition exists after attempting an input/output operation in shared mode, Status Key 2 contains a value of '0' indicating that no further information is available.

7. When Status Key 1 contains a value of '8' indicating an unsuccessful input/output operation for a shared mode file, Status Key 2 can contain a value of '2', '3', '4', '5', or '6', with the following meanings:

   '2' - Indicates an unsuccessful attempt by the system to update a file label.

   '3' - Indicates a shared mode malfunction that can be corrected only by doing an Initial Program Load (IPL). Refer to the VS System Operator's Reference.

   '4' - Indicates an attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.

   '5' - Indicates an attempt was made to update a file for which the user has READ-only access.

   '6' - Indicates an invalid sequence of function requests, for example, attempting to do a START HOLD on a file while another file is already held.

8. When Status Key 1 contains a value of '9' indicating a special condition, Status Key 2 can contain a value of '5', or '7' with the following meanings:

   '5' - Indicates an invalid function request or an invalid sequence of function requests for a given combination of device type, open mode, and file orqanization, for example, attempting to WRITE a record while the file is opened in the input mode.

   '7' - Indicates an attempt to execute an input/output operation for a record of invalid length.

For the workstation file, when Status Key 1 contains a value of '0' indicating successful completion, Status Key 2 contains the terminating attention identifier (AID) character. (Refer to Section E.4, AID Characters.) When Status Key 1 contains a value of '3' indicating a permanent error, Status Key 2 contains a value of '4' indicating that invalid information has been supplied to the workstation order area. For example, the cursor position has been specified as Row 25 Column 10.

E.3 INDEXED FILES

E.3.1 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' - Indicates Successful Completion
'1' - Indicates AT END
'2' - Indicates INVALID KEY
'3' - Indicates Permanent Error
'6' - Indicates an I/O Cancel Condition
'7' - Indicates a TIMEOUT Condition
'8' - Indicates a Shared Mode Error
'9' - Indicates Special Conditions.

The meanings of the above indications are as follows:

'0' - Successful Completion. The input/output statement 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 unsuccessful as the result of boundary violation or as a result of an input/output error, such as data check, parity error, or transmission error.

'6' - An input/output cancel condition has occurred, but the cancel messages have been suppressed as a result of a user request.

'7' - A TIMEOUT condition exists after an attempt to execute an input/output statement in shared mode.

'8' - The input/output statement was unsuccessfully executed as a result of an error occurring while the file was opened in shared mode.

'9' - Special Conditions. The input/output statement was unsuccessfully executed as a result of a condition, such as an invalid function or function sequence, that is unique to the file or device.

E.3.2 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:

1. When Status Key 1 contains a value of '0' indicating successful completion, Status Key 2 may contain a value of '0' or '2', with the following meanings:

   '0' - Indicates that no further information is available concerning the input/output operation.

   '2' - Indicates a duplicate key in an alternate indexed file. This value is returned in two cases: when at least one more record exists with the same alternate key value as the record that has just been READ; or when the record just written, by a WRITE or REWRITE, created a duplicate key value for at least one alternate record key.

2. When Status Key 1 contains a value of '1' indicating an at end condition, Status Key 2 contains a value of '0' indicating that no further information concerning the input/output operation is available.

3. When Status Key 1 contains a value of '2' indicating an INVALID KEY condition, Status Key 2 can contain a value of '1', '2', '3', or '4', with the following meanings:

   '1' - Indicates a sequence error for a sequentially accessed indexed file. The ascending sequence requirements of successive record key values have been violated, or the record key value has been changed by the COBOL program between the successful execution of a READ statement and the execution of the next REWRITE statement for that file.

   '2' - Indicates a duplicate record key value. An attempt has been made to WRITE or REWRITE a record that would create a duplicate record key in an indexed file.

   '3' - Indicates 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' - Indicates a boundary violation. Either an attempt has been made to WRITE beyond the externally defined boundaries of an indexed file, or an START has been issued for a record key greater than the highest record key value in the file.

4. When Status Key 1 contains a value of '3' indicating a permanent error condition, Status Key 2 can contain a value of '0' or '4', with the following meanings:

   '0' - Indicates a hardware error occurred when an input/output operation was attempted. Refer to SVC CHECK in the VS Operating System Services Reference. This value is not returned for program related errors.

   '4' - Indicates that an attempt has been made to write beyond the externally defined boundaries of a file.

5. When Status Key 1 contains a value of '6' indicating an input/output cancel condition with cancel messages suppressed, Status Key 2 contains a value of '0' indicating that no further information is available. To set the error message flag, process the file in nonshared mode.

6. When Status Key 1 contains a value of '7' indicating that a TIMEOUT condition exists after attempting an input/output operation in shared mode, Status Key 2 contains a value of '0' indicating that no further information is available.

7. When Status Key 1 contains a value of '8' indicating an unsuccessful input/output operation for a shared mode file, Status Key 2 can contain a value of '0', '2', '3', '4', '5', or '6', with the following meanings:

   '0' - Indicates an INVALID KEY condition. The the value specified for the key in a READ OR START statement is not found in the file.

   '2' - Indicates an unsuccessful attempt by the system to update a file label.

   '3' - Indicates a shared mode malfunction that can be corrected only by doing an Initial Program Load (IPL). Refer to the VS System Operator's Reference.

   '4' - Indicates an attempt to REWRITE a variable-length record whose record length is greater than the maximum record size specified for the file.

   '5' - Indicates an attempt was made to update a file for which the user has READ-only access.

'6' - Indicates an invalid sequence of function requests, for example, attempting to do a START HOLD on a file while another file is already held.

8. When Status Key 1 contains a value of '9' indicating a special condition, Status Key 2 can contain a value of '5', '7', or '8', with the following meanings:

   '5' - Indicates an invalid function request or an invalid sequence of function requests for a given combination of device type, open mode, and file orqanization, for example, attempting to REWRITE a record in the shared mode after the HOLD has been released by an intervening READ WITH HOLD on another file.

   '7' - Indicates an attempt to execute an input/output operation for a record of invalid length.

   '8' - Indicates the use of a nonexistent alternate record key when attempting to REWRITE or WRITE a record of an alternate indexed file.

E.4 <u>RELATIVE FILES</u>

E.4.1 <u>Status Key 1</u>

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' - Indicates Successful Completion
'1' - Indicates AT END
'2' - Indicates INVALID KEY
'3' - Indicates Permanent Error
'9' - Indicates Special Conditions.

The meanings of the above indications are as follows:

'0' - Successful Completion. The input/output statement 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.

Duplicate Key
No Record Found
Boundary Violation

'3' - Permanent Error. The input/output statement was unsuccessful as the result of boundary violation or as a result of an input/output error, such as data check, parity error, or transmission error.

'9' - Special Conditions. The input-output statement was unsuccessfully executed as a result of a condition that is specified by the operating system. This value is used only to indicate a condition not indicated by other defined values of status key 1, or by specified combinations of the values of status key 1 and status key 2.

E.4.2 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:

1. If no further information is available concerning the input-output operation, status key 2 contains a value of '0'.

1. When Status Key 1 contains a value of '2' indicating an INVALID KEY, Status Key 2 is used to designate the cause of that condition as follows:

   '2' - Indicates a duplicate key value. An attempt has been made to write a record that would create a duplicate key in a relative file.

   '3' - Indicates 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' - A boundary violation. An attempt has been made to write beyond the externally-defined boundaries of a relative file.

2. When Status Key 1 contains a value of '9' indicating a special condition, Status Key 2 can contain a value of '7', or '8', with the following meanings:

   '7' - Indicates an attempt to execute an input/output operation for a record of invalid length.

   '8' - Indicates the use of a nonexistent alternate record key when attempting to REWRITE or WRITE a record of an alternate indexed file.

E.5 INVALID KEY Condition

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

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

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

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, 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.

E.6 AID Characters

The AID (Attention Identifier) character is modified when the operator presses the ENTER Key, HELP Key and the Program Function Keys. The following table illustrates the AID character values corresponding to each of the actions:

Table E-1. Attention ID (AID) Configurations

AID	Hex Character (ASCII)	Graphic Character	AID	Hex Character (ASCII)	Graphic Character
Keyboard unlocked	20	' ' (blank)	Locked by write	21	'
ENTER key	40	@			
PF key 1	41	A	PF key 17	61	a
PF key 2	42	B	PF key 18	62	b
PF key 3	43	C	PF key 19	63	c
PF key 4	44	D	PF key 20	64	d
PF key 5	45	E	PF key 21	65	e
PF key 6	46	F	PF key 22	66	f
PF key 7	47	G	PF key 23	67	g
PF key 8	48	H	PF key 24	68	h
PF key 9	49	I	PF key 25	69	i
PF key 10	4A	J	PF key 26	6A	j
PF key 11	4B	K	PF key 27	6B	k
PF key 12	4C	L	PF key 28	6C	l
PF key 13	4D	M	PF key 29	6D	m
PF key 14	4E	N	PF key 30	6E	n
PF key 15	4F	O	PF key 31	6F	o
PF key 16	50	P	PF key 32	70	p

APPENDIX F
PRINTER CONTROL CHARACTERS

This appendix describes the characters used in controlling WRITE of a record to a printer file. The 2-byte printer control characters are defined in the Figurative-Constants, in the Environment Division of the COBOL program. The format of the WRITE statement for printer files using printer control characters is as follows:

WRITE print-record AFTER ADVANCING user-figurative-constant.

The user-figurative-constant is formed by setting on the bits corresponding to the desired function. Table F-1 describes the bits used in the 2-byte printer control area.

Table F-1. Printer Control Characters

Control		Function
Byte	Bit	
0	0	Line or channel spacing select 0 = Space number of lines specified in the second control byte 1 = Skip to the channel specified in the second control byte
	1	0 = Space before printing 1 = Space after printing
	2	0 = Normal width characters 1 = Double width (expanded print) characters
	3	1 = Actuate hardware alarm
	4-7	RESERVED
1	0	RESERVED
	1-7	Binary number of lines to space (0-127) or channel for skip (1-12)

For more detailed information on printer record formats and options, refer to the VS Principles of Operation. Features such as the printer hardware alarm and channel skipping are not available on all VS printers; before coding the Figurative-Constant for the feature, be sure it is supported on the printer.

Table F-2 illustrates the Figurative-Constant settings for specified actions on a WRITE statement for printer files.

Table F-2. Figurative-Constant Settings for Printer Control

Figurative-Constant	Function
0004	Space 4 lines before printing.
4004	Space 4 lines after printing.
8001	Skip to Channel 1 on the printer (if the printer supports channel skipping).
2003	If the printer supports expanded print, space 3 lines before printing and print the line using expanded print characters; if the printer does not support expanded print, space 3 lines before printing (the expanded print bit will be ignored).
707F	(Assume the printer supports all features.) 1. Print the line using expanded print characters. 2. Actuate the hardware alarm. 3. Space 127 lines after printing.

APPENDIX G
INTERMEDIATE RESULTS

This appendix describes when intermediate results are used in arithmetic computations and the algorithms used to determine the characteristics of the intermediate result.

Intermediate results are used in a COMPUTE statement that specifies a series of arithmetic operations, or in arithmetic expressions contained in an IF, PERFORM, or SEARCH statement. When an arithmetic statement contains only a single pair of operands, an intermediate result is not used.

A series of arithmetic operations is treated as a succession of operations performed according to the Evaluation Rules of Subsection 12.3.2. The result of each successive operation is defined to be an intermediate result.

The intermediate result has an implied PICTURE and USAGE that are defined as follows:

Let OP1 and OP2 be the operands of the function.

ROP1 -	Indicates the number of digits to the right of the decimal point in OP1.
LOP1 -	Indicates the number of digits to the left of the decimal point in OP1.
ROP2 -	Indicates the number of digits to the right of the decimal point in OP2.
LOP2 -	Indicates the number of digits to the left of the decimal point in OP2.
RI -	Indicates the number of digits to the right of the decimal point in the intermediate result.
LI -	Indicates the number of digits to the left of the decimal point in the intermediate result.
RF -	Indicates the number of digits to the right of the decimal point in the final result field.
LF -	Indicates the number of digits to the left of the decimal point in the final result field.

For addition (OP1 + OP2):
    RI = max (ROP1,ROP2)
    LI = max (LOP1,LOP2) + 1

```
For subtraction (OP1 - OP2):
    RI = max (ROP1,ROP2)
    LI = max (LOP1,LOP2) + 1

For multiplication (OP1 * OP2):
    RI = ROP1 + ROP2
    LI = LOP1 + LOP2 + 2

For division (OP1 / OP2):
    The quotient:
        RI = max (ROP1 - ROP2, ROP1, ROP2, RF),
            plus one if ROUNDED is specified for the final result
        LI = LOP1 + ROP2
    The remainder:
        RI = ROP2 + (RI of the quotient)
        LI = LOP2 - (RI of the quotient)

For exponentiation (OP1 ** OP2):
    RI = ROP1
    LI = max (2 * LOP1 + 1, LF)

For comparison (OP1 compared with OP2):
    RI = max (ROP1,ROP2)
    LI = max (LOP1,LOP2)
```

In any instance if RI + LI exceeds 30, truncation of all but the 30 least significant digits will occur.

The usage of the intermediate data item will be either BINARY or COMPUTATIONAL.

The usage is BINARY in the following cases.

1. OP1 and OP2 both have usage BINARY.
2. One operand has usage BINARY and the other is an index name.

The usage is COMPUTATIONAL in all other cases.

When the usage is COMPUTATIONAL, the implied PICTURE is S9(m)V9(n), where m is the value of LI and n is the value of RI, recognizing that m + n may exceed 18, but will not exceed 30.

It is the programmer's responsibility to make sure that the operands of any arithmetic statement are defined with enough decimal places to give the desired accuracy in the final result.

APPENDIX H
PASSING PARAMETERS TO COBOL SUBROUTINES

This appendix describes the protocol required for passing parameters from one COBOL program to another COBOL program. The CALL statement names the subroutine that is to be executed; in this discussion, the program calling the subroutine is called the "calling program", while the subroutine being called is called the "called program".

The format of the CALL statement is

```
CALL literal-1 [USING identifier-1 [identifier-2]...]
```

In the calling program, the following rules must be observed:

1. The subroutine-name in the calling program must be identical to the PROGRAM-ID of the called program.

2. The parameters to be passed in the USING list must be defined in either the File Section (except for the FD name), the Working-Storage Section or the Linkage Section of the calling program.

3. If the parameters are in the Linkage Section of the calling program, the calling program must itself have been called from another program.

4. Recursive calls are not allowed. A program cannot call itself, and neither can any chain of CALL statements refer back to a calling program. Thus PROGRAM A issuing a CALL to PROGRAM B, which issues a CALL to program C, is legal; while PROGRAM A issuing a CALL to PROGRAM B, which issues a CALL to PROGRAM A, is illegal. Illegal CALL sequences produce unpredictable results at runtime.

5. The FD name of a file cannot be passed to a COBOL subroutine. However, as the address of the User File Block, it can be passed to an assembler subroutine.

In the called program, the parameters are received by the USING phrase of the Procedure Division. The format is

```
PROCEDURE DIVISION [USING data-name-1] [data-name-2]...].
```

The USING phrase identifies a program as a called subroutine. In the called program observe the following rules:

1. The parameters in the USING list must be defined in the Linkage Section.

2. The parameters in the USING list are 01 or 77 level items.

3. The length of the parameters in the USING list of the called program corresponds with the length of the parameters in the calling program. The correspondence is positional; that is, the nth parameter of the USING list of the called program corresponds with the nth parameter of the USING list of the calling program. The structure of the parameters do not have to correspond, but their length does.

The VS LINKER is run to combine the called program and the calling program. The VS LINKER combines program files and resolves addresses so that the called program and the calling program can communicate. Refer to VS PROGRAM DEVELOPMENT TOOLS for information on the VS LINKER.

Figure H-1 is a COBOL calling program that passes a table entry with 3 levels of OCCURS. The table entry is passed in three ways: with integer occurrence numbers, with INDEXED BY index names, and with subscripted data names.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  CALLTABL.
000300*****************************************************************
000500*  This program illustrates 3 methods of passing a table       *
000600* entry to a COBOL subroutine.  The table entry (TABLE-ENTRY)  *
000700* occurs in THREE-LEVEL-TABLE, and can be referenced using     *
000800* either integer occurrence numbers (TABLE-ENTRY (1, 2, 3));   *
000900* index data items defined in the INDEXED BY clause            *
001000* (TABLE-ENTRY (X, Y, Z)); or subscripted data items defined in*
001100* Working-Storage (TABLE-ENTRY (SUB1, SUB2, SUB3)).            *
001300*****************************************************************
001400 ENVIRONMENT DIVISION.
001500 DATA DIVISION.
001600 WORKING-STORAGE SECTION.
001700 01  THREE-LEVEL-TABLE.
001800     03  LEVEL1    OCCURS 5 TIMES INDEXED BY X.
001900         05  LEVEL2   OCCURS 6 TIMES INDEXED BY Y.
002000             07  LEVEL3  OCCURS 7 TIMES INDEXED BY Z.
002100                 09 TABLE-ENTRY    PICTURE IS XXX.
002200 77  SUB1  COMPUTATIONAL VALUE IS +1   PICTURE IS S99.
002300 77  SUB2                VALUE IS +2   PICTURE IS S99.
002400 01  SUB3  BINARY        VALUE IS +3.
002500 PROCEDURE DIVISION.
002600 CALL-USING-TABLE-ENTRY.
002700*
002800* The table entry (TABLE-ENTRY) is referenced by integer
002900* occurrence numbers in the table THREE-LEVEL-TABLE.
003000*
003100     MOVE "ABC" TO TABLE-ENTRY (1, 2, 3).
003200     CALL "CALLSUBR" USING TABLE-ENTRY (1, 2, 3).
003300*
003400* The following call will reference the same table entry using
003500* index data items, after setting the index items to the
003600* desired occurrence values.
003700*
003800     SET X TO 1.
003900     SET Y TO 2.
004000     SET Z TO 3.
004100     CALL "CALLSUBR" USING TABLE-ENTRY (X, Y, Z).
004200*
004300* The following call uses subscripted data items and will
004400* reference the identical table entry.
004500*
004600     CALL "CALLSUBR" USING TABLE-ENTRY (SUB1, SUB2, SUB3).
004700     STOP RUN.
```

Figure H-1. Calling Program Passing Table Entry Parameter

Figure H-2 is the program CALLSUBR called by the program of Figure H-1 and having the proper 3-byte entry in the Linkage Section to receive the table entry.

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  CALLSUBR.
000210*****************************************************************
000230*   The subroutine CALLSUBR receives table entry TABLE-ENTRY.   *
000240* TABLE-ENTRY is the elementary item in the Linkage Section.    *
000250* Since CALLSUBR receives the address of the table entry, it    *
000255* does not have any knowledge that it has been passed the       *
000260* element of a table -- therefore it references the elementary *
000265* item.                                                         *
000275*****************************************************************
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 LINKAGE SECTION.
000600 01  TABLE-ENTRY            PICTURE IS XXX.
002100 PROCEDURE DIVISION  USING TABLE-ENTRY.
002200 BEGINIT.
002300     DISPLAY "We are in CALLSUBR.  Table entry  = "  TABLE-ENTRY.
002500 GOBACK.
002600     EXIT PROGRAM.
```

Figure H-2. Called Program Receiving Table Entry Parameter

APPENDIX I
A COMPARISON OF VS, ANSI AND FIPS COBOL STANDARDS

The 1974 American National Standards Institute (ANSI) specifications for COBOL (ANSI X3.23-1974) are divided into 12 modules organized according to processing functions. These modules define different features of the language. The modules are : the Nucleus, Table Handling, Sequential I/O, Relative I/O, Indexed I/O, Sort-Merge, Report Writer, Segmentation, Libary, Debug, Inter-Program Communication, and Communication. The Nucleus contains features that are necessary for internal processing. The Table Handling module contains features necessary for defining and referencing tables. The I/O modules contain features necessary for the definition and access of external files either organized sequentially, identified and accessed by relative record numbers, or identified and accessed by the values of a key. The Sort-Merge module allows a COBOL program to perform sorts. The Report-Writer provides for the production of printed reports. The Segmentation module provides for the overlaying of Procedure Division sections at compile-time. The Library module provides for the inclusion into a program of external COBOL text. The Debug module allows the user to specify the conditions under which items are monitored for debugging during execution. The Inter-Program Communication module allows a program to communicate with one or more other programs. The Communications module provides the ability to communicate with local or remote communications devices.

All the modules except Report-Writer contain a higher and lower level. In each case, the features of the lower level are supported in the higher level, except that the higher level may remove some restrictions that are enforced in the lower level.

The Federal Information Processing Standard (FIPS) for COBOL (FIPS PUB 21-1) is based on the ANSI standard. FIPS divides the features of ANSI COBOL into four levels identified as Low, Low-Intermediate, High-Intermediate, and High. As Table I-1 shows, each FIPS level is composed of the features from the high or low levels of ANSI modules. The numbers in the table refer to the levels of the corresponding ANSI module. 1 designates the low level; 2 designates the high level. A dash in the table denotes that the corresponding module is omitted from a particular FIPS level.

Table I-1. Federal Information Processing Standard

	Low Level	Low Intermediate Level	High Intermediate Level	High Level
Nucleus	1	1	2	2
Table Handling	1	1	2	2
Sequential I-O	1	1	2	2
Relative I-O	--	1	2	2
Indexed I-O	--	--	--	2
Sort-Merge	--	--	1	2
Report Writer	--	--	--	--
Segmentation	--	1	1	2
Library	--	1	1	2
Debug	--	1	2	2
Inter-Program Communication	--	1	2	2
Communication	--	--	2	2

Table I-2 lists all the features of VS and ANSI COBOL, organized according to the source-program divisions. The column entitled "VS" indicates whether the feature is supported in VS COBOL. A "Y" in the column indicates that the feature is supported. "C" indicates that the feature is accepted by the compiler as a comment entry only. "E" indicates that the feature is a VS extension to the ANSI standard. "I" indicates that the feature is an implementor name. "N" indicates that the feature is not supported. Note, however, that all ANSI reserved words are recognized as such by the VS compiler and can not be user-defined words. (Refer to Appendix A, VS COBOL Reserved Words.)

The column entitled ANSI indicates whether a feature conforms to the ANSI standard. A dash in the column indicates that the feature is absent from the standard. A three-character abbreviation indicates the module to which a feature belongs. The number preceding the abbreviation indicates the level to which the feature belongs. 1 indicates the lower level; 2 indicates the higher level. The meanings of the abbreviations are as follows.

Abbreviation	Meaning
NUC	Nucleus
TBL	Table Handling
SEQ	Sequential I/O
REL	Relative I/O
INX	Indexed I/O
IPC	Inter-Program Communication
SEG	Segmentation
STR	Sort-Merge
LIB	Library
DEB	Debug
RPW	Report Writer
COM	Communication

Table I-2. SUMMARY OF DIFFERENCES

SUMMARY OF DIFFERENCES IN LANGUAGE CONCEPTS

ELEMENT	ANSI	VS
LANGUAGE CONCEPTS		
Character Set		
Characters used in words 0-9 A-Z - (hyphen)	1 NUC	Y
Characters used in punctuation " ( ) space	1 NUC	Y
, (comma) ; (semicolon)	2 NUC	Y
=	2 LIB	N
Characters used in editing B + - . , Z * $ 0 CR DB /	1 NUC	Y
Characters used in arithmetic operations + - * / **	2 NUC	Y
Characters used in relation conditions > < =	2 NUC	Y
Characters used in subscripting +	1 TBL	Y
Characters used in indexing + -	1 TBL	Y
Double character substitution allowed	1 NUC	Y
Separators		
" ( ) . space	1 NUC	Y
, (comma) ; (semicolon)	2 NUC	Y
A space which is part of a separator may be one or more space characters	1 NUC	Y
Character-Strings		
COBOL words		
Maximum of 30 characters	1 NUC	Y
User-defined words		
Names must be unique if referenced	1 NUC	Y
Names may be qualified for uniqueness	2 NUC	Y
Alphabet-name	1 NUC	C
Cd-name	1 COM	N
Condition-name	2 NUC	Y
Data-Name	1 NUC	Y
Must begin with alphabetic character	1 NUC	N
Need not begin with alphabetic character	2 NUC	Y
File-name	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
Index-name	1 TBL	Y
Level-number	1 NUC	Y
Library-name	1 LIB	Y
Mnemonic-name	1 NUC	Y
Paragraph-name	1 NUC	Y
Program-name	1 NUC	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN LANGUAGE CONCEPTS (continued)

ELEMENT	ANSI	VS
User-defined words (continued)		
Record-name	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
Report-name	1 RPW	N
Routine-name	1 NUC	C
Section-name	1 NUC	Y
Segment-number	1 SEG	Y
Text-name	1 LIB	N
User-figurative-constant	--	E
System-names		
Computer-name	1 NUC	Y
Implementor-name	1 NUC	Y
Reserved words		
Required words	1 NUC	Y
Key words	1 NUC	Y
Special characters words		
Arithmetic operators + - * / **	2 NUC	Y
Arithmetic operators used in subscripting +	1 TBL	Y
Arithmetic operators used in indexing + -	1 TBL	Y
Relation characters > < =	2 NUC	Y
Optional words	1 NUC	Y
Special purpose words		
Figurative constants: ZERO, SPACE, HIGH-VALUE, LOW-VALUE, QUOTE	1 NUC	Y
Figurative constants: ZEROS, ZEROES, SPACES, HIGH-VALUES, LOW-VALUES, QUOTES	2 NUC	Y
Figurative constants: ALL literal	2 NUC	N
Special registers		
LINAGE-COUNTER	2 SEQ	N
LINE-COUNTER	1 RPW	N
PAGE-COUNTER	1 RPW	N
DEBUG-ITEM	1 DEB	Y
Literals		
Numeric literals: 1 through 18 digits	1 NUC	Y
Nonnumeric literals: 1 through 120 characters	1 NUC	
PICTURE character-strings	1 NUC	Y
Comment-entries	1 NUC	Y
Uniqueness of Reference		
Qualification	2 NUC	Y
Subscripting (data-name/literal)	1 TBL	Y
Subscripting (index-name)	1 TBL	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN LANGUAGE CONCEPTS (continued)

ELEMENT	ANSI	VS
Reference Format		
Sequence number	1 NUC	Y
Continuation of lines		
Continuation of nonnumeric literal	I NUC	Y
Continuation of COBOL word, numeric literal, PICTURE character-string	2 NUC	N
Intervening comment lines allowed	1 NUC	Y
Blank lines	1 NUC	Y
Comment lines		
Asterisk (*) comment line	1 NUC	Y
Stroke (/) comment line	1 NUC	Y
Percent sign (%) comment line	--	E
Debugging line with D in indicator area	1 DEB	Y
Source Program Structure		
Identification Division required	1 NUC	Y
Environment Division required	1 NUC	Y
Data Division required	1 NUC	Y
Procedure Division required	1 NUC	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN IDENTIFICATION DIVISION

ELEMENT	ANSI	VS
IDENTIFICATION DIVISION		
PROGRAM-ID paragraph		
Program-name	1 NUC	Y
AUTHOR paragraph	1 NUC	Y
INSTALLATION paragraph	1 NUC	Y
DATE-WRITTEN paragraph	1 NUC	Y
DATE-COMPILED paragraph	2 NUC	Y
SECURITY paragraph	1 NUC	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN ENVIRONMENT DIVISION

ELEMENT	ANSI	VS
ENVIRONMENT DIVISION		
Configuration Section		
Configuration Section is required	1 NUC	N
Configuration Section is optional	--	E
SOURCE-COMPUTER paragraph		
SOURCE-COMPUTER paragraph is required	1 NUC	N
SOURCE-COMPUTER paragraph is optional	--	E
Computer-name	1 NUC	Y
WITH DEBUGGING MODE clause for debugging lines	1 DEB	Y
WITH DEBUGGING MODE clause for debugging sections	1 DEB	Y
OBJECT-COMPUTER paragraph	1 NUC	C
Computer-name	1 NUC	C
MEMORY SIZE clause	1 NUC	C
PROGRAM COLLATING SEQUENCE clause	1 NUC	C
SEGMENT-LIMIT clause	2 SEG	N
SPECIAL-NAMES paragraph		
ALPHABET clause	1 NUC	C
STANDARD-1 option	1 NUC	C
NATIVE option	1 NUC	C
Implementor-name option	1 NUC	N
Literal option	2 NUC	N
CURRENCY SIGN clause	1 NUC	Y
DECIMAL-POINT clause	1 NUC	Y
SWITCH-1,..., SWITCH-7	--	I
IS mnemonic-name option	1 NUC	Y
ON STATUS IS condition-name option	1 NUC	Y
OFF STATUS IS condition-name option	1 NUC	Y
FIGURATIVE-CONSTANTS paragraph	--	E

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN ENVIRONMENT DIVISION (continued)

ELEMENT	ANSI	VS
Input-Output Section	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
FILE-CONTROL paragraph	1 SEQ	Y
	1 REL	Y
	1 INX1	Y
	1 SRT	Y
	1 RPW	N
File control entry	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
SELECT clause	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
OPTIONAL phrase	2 SEQ	N
	2 REL	N
	2 INX	N
ACCESS MODE clause		
SEQUENTIAL	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
RANDOM	1 REL	Y
	1 INX	Y
DYNAMIC	2 REL	Y
	2 INX	Y
RELATIVE KEY phrase	1 REL	Y
ALTERNATE RECORD KEY clause	2 INX	Y
Integer option	--	E
WITH DUPLICATES phrase	2 INX	Y
ASSIGN clause	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
"DISK", "DISPLAY", "PRINTER", or "TAPE" option	--	I
NODISPLAY phrase	--	E
CURSOR clause	--	E
BUFFER clause	--	E
PFKEY clause	--	E

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN ENVIRONMENT DIVISION (continued)

ELEMENT	ANSI	VS
File control entry (continued)		
FILE STATUS clause	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
ORGANIZATION clause		
SEQUENTIAL	1 SEQ	Y
	1 RPW	N
RELATIVE	1 REL	Y
INDEXED	1 INX	Y
RECORD KEY clause	1 INX	Y
RESERVE AREA clause	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	1 RPW	N
I-O-CONTROL paragraph		
MULTIPLE FILE TAPE clause	2 SEQ	N
RERUN clause	1 SEQ	C
	1 REL	C
	1 INX	C
SAME AREA clause	1 SEQ	N
	1 REL	C
	1 INX	Y
SAME RECORD AREA clause	2 SEQ	Y
	2 REL	Y
	2 INX	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION

ELEMENT	ANSI	VS
DATA DIVISION		
File Section	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y
	1 RPW	N
File description entry	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
FD level indicator	1 SEQ	Y
	1 REL	Y
	1 INX	Y
BLOCK CONTAINS clause		
Integer RECORDS/CHARACTERS	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
Integer-1 TO integar-2 RECORDS/CHARACTERS	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	1 RPW	N
CODE-SET clause	1 SEQ	C
	1 RPW	N
DATA RECORDS clause	1 SEQ	C
	1 REL	C
	1 INX	C
LABEL RECORDS clause	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
LINAGE clause	2 SEQ	N
FOOTING phrase	2 SEQ	N
TOP phrase	2 SEQ	N
BOTTOM phrase	2 SEQ	N

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION (continued)

ELEMENT	ANSI	VS
File description entry (continued)		
RECORD clause		
Integer-1 CHARACTERS	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
Integer-4 TO integer-5 CHARACTERS	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	1 RPW	N
COMPRESSED phrase	--	E
REPORT clause	1 RPW	N
VALUE OF clause		
Implementor-name IS literal	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
Implementor-name IS data-name	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	1 RPW	N
Series	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	2 RPW	N
RECOVERY-BLOCKS IS	--	E
RECOVERY-STATUS IS	--	E
DATABASE-NAME IS	--	E
Sort-merge file description entry	1 SRT	Y
SC level indicator	1 SRT	Y
DATA RECORDS clause	1 SRT	Y
RECORD clause		
Integer-1 CHARACTERS	1 SRT	Y
Integer-4 TO integer-5 CHARACTERS	1 SRT	Y
COMPRESSED phrase	--	E
Record description entry in File Section	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 SRT	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION (continued)

ELEMENT	ANSI	VS
Working-Storage Section	1 NUC	Y
Record description entry	1 NUC	Y
77 level description entry	1 NUC	Y
Linkage Section	1 IPC	Y
Record description entry	1 IPC	Y
77 level description entry	1 IPC	Y
Communication Section	1 COM	N
Communication description entry	1 COM	
CD level indicator	1 COM	
FOR INPUT clause	1 COM	
INITIAL phrase	2 COM	
END KEY clause	1 COM	
MESSAGE COUNT clause	1 COM	
MESSAGE DATE clause	1 COM	
MESSAGE TIME clause	1 COM	
SYMBOLIC QUEUE clause	1 COM	
SYMBOLIC SOURCE clause	1 COM	
SYMBOLIC SUB-QUEUE-1 clause	2 COM	
SYMBOLIC SUB-QUEUE-2 clause	2 COM	
SYMBOLIC SUB-QUEUE-3 clause	2 COM	
STATUS KEY clause	1 COM	
TEXT LENGTH clause	1 COM	
FOR OUTPUT clause	1 COM	
DESTINATION COUNT clause	1 COM	
Must be one	1 COM	
Must be one or greater	2 COM	
DESTINATION TABLE clause	2 COM	
INDEXED BY phrase	2 COM	
ERROR KEY clause	1 COM	
SYMBOLIC DESTINATION clause	1 COM	
STATUS KEY clause	1 COM	
TEXT LENGTH clause	1 COM	
Record description entry	1 COM	

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION (continued)

ELEMENT	ANSI	VS
Report Section	1 RPW	N
Report description entry	1 RPW	
RD level indicator	1 RPW	
CODE clause	1 RPW	
CONTROL clause	1 RPW	
PAGE clause	1 RPW	
Report group description entry	1 RPW	
The following clauses appear in record description entry, data description entry, 77 level description entry, report group description entry, or workstation screen description entry:		
BLANK WHEN ZERO clause	1 NUC 1 RPW	Y
COLUMN NUMBER clause	1 RPW	N
COLUMN clause (workstation screen)	--	E
Data-name clause	1 NUC 1 RPW	Y
FILLER clause	1 NUC	Y
GROUP INDICATE clause	1 RPW	N
JUSTIFIED clause	1 NUC 1 RPW	Y
Level-number clause	1 NUC	Y
01 through 10; one digit representation	1 NUC	Y
01 through 49; one or two digit presentation	2 NUC 1 RPW	Y
66	2 NUC	N
77	1 NUC	Y
88	2 NUC	Y
LINE NUMBER clause	1 RPW	N
LINE clause (workstation screen)	--	E
NEXT GROUP clause	1 RPW	N
OBJECT clause (workstation screen)	--	E
OCCURS clause	1 TBL	Y
Integer Times	1 TBL	Y
ASCENDING/DESCENDING Key clause	2 TBL	Y
INDEXED BY phrase	1 TBL	Y
Integer-1 TO interger-2 TIMES DEPENDING ON phrase	2 TBL	N

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION (continued)

ELEMENT	ANSI	VS
PICTURE clause	1 NUC	Y
	1 RPW	N
Character-string has a maximum of 30 characters	1 NUC	Y
	1 RPW	N
Data characters: X 9 A	1 NUC	Y
	1 RPW	N
Operational symbols: S V	1 NUC	Y
	1 RPW	N
Operational symbols: P	1 NUC	Y
	1 RPW	N
Nonfloating insertion characters B + - . , $ 0 CR DB /	1 NUC	Y
	1 RPW	N
B allowed in alphabetic item	1 NUC	Y
	1 RPW	N
Replacement or floating insertion characters $ + - Z *	1 NUC	Y
	1 RPW	N
Currency sign substitution	1 NUC	Y
	1 RPW	N
Decimal point substitution	1 NUC	Y
	1 RPW	N
RANGE clause (workstation screen)	--	E
REDEFINES clause	1 NUC	Y
May not be nested	1 NUC	Y
May be nested	2 NUC	N
Redefining of 01 levels may be greater than size of original area	1 NUC	Y
Redefining of non-01 levels must be equal to size of original area	1 NUC	N
Redefining of non-01 levels must be less than or equal to size of original area	--	Y
ROW clause (workstation screen)	--	E
NAMES clause	2 NUC	N
SIGN clause	1 NUC	Y
SOURCE clause	1 RPW	N
SOURCE clause (workstation screen)	--	E
SUM clause	1 RPW	N
SYNCHRONIZED clause	1 NUC	C
TYPE clause	1 RPW	N
USAGE clause	1 NUC	Y
	1 RPW	N
BINARY	--	E
COMPUTATIONAL	1 NUC	Y
DISPLAY	1 NUC	Y
	1 RPW	N
DISPLAY-WS	--	E
INDEX	1 NUC	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN DATA DIVISION (continued)

VALUE clause	1 NUC	Y
	1 RPW	N
Literal	1 NUC	Y
	1 RPW	N
Literal series	2 NUC	N
Literal-1 THROUGH Literal-2	2 NUC	N
Literal range series	2 NUC	N
User-figurative-constant	--	E

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION

ELEMENT	ANSI	VS
PROCEDURE DIVISION		
Procedure Division header	1 NUC	Y
USING phrase	1 IPC	Y
Declarative procedures	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
	1 DEB	Y
Arithmetic expressions	2 NUC	Y
Binary arithmetic operators + - * / **	2 NUC	Y
Unary arithmetic operators +-	2 NUC	Y
Conditional expressions	1 NUC	Y
Simple condition	1 NUC	Y
Relation condition	1 NUC	Y
Relational operators		
[NOT] GREATER THAN	1 NUC	Y
[NOT] >	2 NUC	Y
[NOT] LESS THAN	1 NUC	Y
[NOT] <	2 NUC	Y
[NOT] EQUAL TO	1 NUC	Y
[NOT] =	2 NUC	Y
Comparison of numeric operands	1 NUC	Y
Comparison of nonnumeric operands		
Operands must be of equal size	1 NUC	N
Operands may be unequal in size	2 NUC	Y
Comparison of index-names and/or index data items	1 TBL	Y
Class condition	1 NUC	Y
NUMERIC	1 NUC	Y
ALPHABETIC (uppercase alphabetic characters)	1 NUC	Y
Condition-name condition	2 NUC	Y
Switch-status condition	1 NUC	Y
Sign condition	2 NUC	Y
Modified Data Tag condition	--	E
Figurative Constant condition	--	E

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
Conditional expressions (continued)		
Complex condition	2 NUC	Y
Logical operators AND OR NOT	2 NUC	Y
Negated condition	2 NUC	Y
Combined condition	2 NUC	Y
Parenthesized conditions	2 NUC	Y
Abbreviated combined relation conditions	2 NUC	Y
Arithmetic statements	1 NUC	Y
Arithmetic operands limited to 18 digits	1 NUC	Y
Composite of operands limited to 18 digits	1 NUC	Y
ACCEPT statement	1 NUC	Y
Identifier	1 NUC	Y
Only one transfer of data	1 NUC	Y
No restriction on number of transfers of data	2 NUC	N
FROM mnemonic-name phrase	2 NUC	N
FROM DATE/DATE/TIME phrase	2 NUC	Y
ACCEPT MESSAGE COUNT statement	1 COM	N
ADD statement	1 NUC	Y
Identifier/literal	1 NUC	Y
Identifier/literal series	1 NUC	Y
TO identifier	1 NUC	Y
TO identifier series	2 NUC	N
GIVING identifier	1 NUC	Y
GIVING identifier series	2 NUC	N
ROUNDED phrase	1 NUC	Y
SIZE ERROR phrase	1 NUC	Y
CORRESPONDING phrase	2 NUC	Y
ALTER statement	1 NUC	Y
CALL statement	1 IPC	Y
Literal	1 IPC	Y
Identifier	2 IPC	N
USING phrase	1 IPC	Y
Data-name	1 IPC	Y
Identifier	--	E
ON OVERFLOW phrase	2 IPC	N
CANCEL statement	2 IPC	N
Literal	2 IPC	
Identifier	2 IPC	

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
CLOSE statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
File-name	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
File-name series	2 SEQ	Y
	1 REL	N
	1 INX	Y
	1 RPW	N
REEL/UNIT phrase	1 SEQ	Y
	1 RPW	N
FOR REMOVAL phrase	2 SEQ	Y
	1 RPW	N
WITH NO REWIND phrase	2 SEQ	Y
	1 RPW	N
WITH LOCK phrase	2 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
COMPUTE statement	2 NUC	Y
Arithmetic expression	2 NUC	Y
Identifier series	2 NUC	N
ROUNDED phrase	2 NUC	Y
SIZE ERROR phrase	2 NUC	Y
END-Compute phrase	2 NUC	N
COPY statement	1 LIB	Y
OF/IN library-name	2 LIB	Y
REPLACING phrase	2 LIB	N
Pseudo-text	2 LIB	N
Identifier	2 LIB	N
Literal	2 LIB	N
Word	2 LIB	N
DECLARATIVES option	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
	1 DEB	Y
END DECLARATIVES phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
	1 DEB	Y
DELETE statement	1 REL	Y
	1 INX	Y
INVALID KEY phrase	1 REL	Y
	1 INX	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
DISABLE statement	1 COM	N
INPUT phrase	1 COM	N
TERMINAL phrase	2 COM	N
OUTPUT phrase	1 COM	N
KEY phrase	1 COM	N
DISPLAY statement	1 NUC	Y
Only one transfer of data	1 NUC	Y
No restriction on number of transfers of data	2 NUC	N
Identifier/literal	1 NUC	Y
Identifier/literal series	1 NUC	Y
UPON mnemonic-name phrase	2 NUC	N
DISPLAY AND READ statement	--	E
DIVIDE statement	1 NUC	Y
BY identifier/literal	1 NUC	Y
INTO identifier	1 NUC	Y
INTO identifier series	2 NUC	N
GIVING identifier	1 NUC	Y
GIVING identifier series	2 NUC	N
ROUNDED phrase	1 NUC	Y
REMAINDER phrase	2 NUC	Y
SIZE ERROR phrase	1 NUC	Y
ENABLE statement	1 COM	N
INPUT phrase	1 COM	N
TERMINAL phrase	2 COM	N
OUTPUT phrase	1 COM	N
KEY phrase	1 COM	N
ENTER statement	1 NUC	C
EXIT statement	1 NUC	Y
EXIT PROGRAM statement	1 IPC	Y
FREE statement	--	E
GENERATE statement	1 RPW	N
Data-name	1 RPW	N
Report-name	1 RPW	N
GO TO statement	1 NUC	Y
Procedure-name is optional	2 NUC	N
DEPENDING ON phrase	1 NUC	Y
HOLD statement	--	E
IF statement	1 NUC	Y
Only imperative statements	1 NUC	N
Imperative and/or conditional statements	2 NUC	Y
Nested IF statements	2 NUC	Y
THEN optional word	--	E
NEXT SENTENCE phrase	1 NUC	Y
ELSE phrase	1 NUC	Y
FAC OF...[NOT] EQUAL....phrase	--	E
FAC OF...ALTERED phrase	--	E
Figurative-constant...ON/OFF phrase	--	E
INITIATE statement	1 RPW	N

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
INSPECT statement	1 NUC	Y
TALLYING phrase	1 NUC	Y
TALLYING phrase series	2 NUC	Y
REPLACING phrase	1 NUC	Y
REPLACING phrase series	2 NUC	Y
TALYING and REPLACING phrases	1 NUC	Y
Only single character data item	1 NUC	Y
Multi-character data item	2 NUC	N
MERGE statement	2 SRT	Y
ASCENDING/DESCENDING KEY phrase	2 SRT	Y
COLLATING SEQUENCE phrase	2 SRT	Y
USING phrase	2 SRT	Y
OUTPUT PROCEDURE phrase	2 SRT	Y
GIVING phrase	2 SRT	Y
USING/GIVING file must be sequential file	2 SRT	Y
MOVE statement	1 NUC	Y
TO identifier	1 NUC	Y
TO identifier series	1 NUC	Y
CORRESPONDING phrase	2 NUC	N
WITH CONVERSION phrase	--	E
TO FAC OF phrase	--	E
FAC OF...TO phrase	--	E
TO ORDER-AREA OF phrase	--	E
ORDER-AREA OF...TO phrase	--	E
MULTIPLY statement	1 NUC	Y
BY identifier	1 NUC	Y
BY identifier series	2 NUC	N
GIVING identifier	1 NUC	Y
GIVING identifier series	2 NUC	N
ROUNDED phrase	1 NUC	Y
SIZE ERROR phrase	1 NUC	Y
OPEN statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
File-name	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
File-name series	2 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
INPUT phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
WITH NO REWIND phrase	2 SEQ	N
REVERSED phrase	2 SEQ	N

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
OPEN statement (continued)		
OUTPUT phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
WITH NO REWIND phrase	2 SEQ	N
	1 RPW	N
I-O-phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
EXTEND phrase	2 SEQ	Y
	2 REL	Y
	2 INX	Y
	1 RPT	N
INPUT, OUTPUT, I-O, and EXTEND series	2 SEQ	Y
	2 REL	Y
	2 INX	
	1 RPT	N
INPUT, OUTPUT, and I-O series	1 REL	Y
	1 INX	
SHARED phrase and SHARED series	--	E
PERFORM statement	1 NUC	Y
THROUGH procedure-name phrase	1 NUC	Y
TIMES phrase	1 NUC	Y
UNTIL phrase	2 NUC	Y
VARYING phrase	2 NUC	Y
AFTER phrase	2 NUC	N
READ statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
NEXT phrase	2 SEQ	Y
	2 REL	Y
	2 INX	Y
INTO phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
AT END phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
READ statement (continued)		
KEY phrase	2 INX	Y
INVALID KEY phrase	1 REL	Y
	1 INX	Y
ALTERED phrase	--	E
MODIFIABLE phrase	--	E
TIMEOUT phrase	--	E
WITH HOLD phrase	--	E
READY/RESET TRACE statements	--	E
RECEIVE statement	1 COM	N
MESSAGE phrase	1 COM	N
SEGMENT phrase	2 COM	N
INTO phrase	1 COM	N
NO DATA phrase	1 COM	N
END-RECEIVE phrase	1 COM	N
RELEASE statement	1 SRT	Y
FROM phrase	1 SRT	Y
RETURN statement	1 SRT	Y
INTO phrase	1 SRT	Y
AT END phrase	1 SRT	Y
REWRITE statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
FROM phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
INVALID KEY phrase	1 REL	Y
	1 INX	Y
AFTER phrase	--	E
ROLLBACK statement	--	E
SEARCH statement	2 TBL	Y
VARYING phrase	2 TBL	Y
AT END phrase	2 TBL	Y
WHEN phrase	2 TBL	Y
WHEN phrase series	2 TBL	Y
SEARCH ALL statement	2 TBL	Y
AT END phrase	2 TBL	Y
WHEN phrase	2 TBL	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
SEND statement	1 COM	N
FROM identifier phrase (portion of a message)	2 COM	N
FROM identifier phrase (complete message)	1 COM	N
WITH identifier phrase	2 COM	N
WITH ESI phrase	2 COM	N
WITH EMI phrase	1 COM	N
WITH EGI phrase	1 COM	N
BEFORE/AFTER ADVANCING phrase	1 COM	N
Integer-1 LINE/LINES	1 COM	N
Identifier LINE/LINES	1 COM	N
Mnemonic-name	2 COM	N
PAGE	1 COM	N
SET statement	1 TBL	Y
Index-name/identifier TO	1 TBL	Y
Index-name UP BY DOWN BY	1 TBL	Y
Figurative-constant...ON/OFF phrase	--	E
SORT statement	1 SRT	Y
ASCENDING/DESCENDING KEY phrase	1 SRT	Y
COLLATING SEQUENCE phrase	2 SRT	Y
INPUT PROCEDURE phrase	1 SRT	Y
USING phrase	1 SRT	Y
File-name series	2 SRT	Y
OUTPUT PROCEDURE phrase	1 SRT	Y
GIVING phrase	1 SRT	Y
DUPLICATES phrase	--	E
START statement	2 REL	Y
	2 INX	Y
KEY phrase	2 REL	Y
	2 INX	Y
INVALID KEY phrase	2 REL	Y
	2 INX	Y
Use with ORGANIZATION IS SEQUENTIAL	--	E
STOP statement	1 NUC	Y
RUN	1 NUC	Y
Literal	1 NUC	Y
STRING statement	2 NUC	Y
DELIMITED BY series	2 NUC	Y
WITH POINTER phrase	2 NUC	Y
ON OVERFLOW phrase	2 NUC	Y

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
SUBTRACT statement	1 NUC	Y
Identifier/literal	1 NUC	Y
Identifier/literal series	1 NUC	Y
FROM identifier	1 NUC	Y
FROM identifier series	2 NUC	N
GIVING identifier	1 NUC	Y
GIVING identifier series	2 NUC	N
ROUNDED phrase	1 NUC	Y
SIZE ERROR phrase	1 NUC	Y
SUPRESS statement	1 RPW	N
TERMINATE statement	1 RPW	N
UNSTRING statement	2 NUC	Y
DELIMITED BY phrase	2 NUC	Y
DELIMITER IN phrase	2 NUC	Y
COUNT IN phrase	2 NUC	Y
WITH POINTER phrase	2 NUC	Y
TALLYING phrase	2 NUC	Y
ON OVERFLOW phrase	2 NUC	Y
USE statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
	1 RPW	N
EXCEPTION/ERROR PROCEDURE phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON file-name	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON file-name series	2 SEQ	Y
	2 REL	Y
	2 INX	Y
ON INPUT	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON OUTPUT	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON I-O	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON EXTEND	1 SEQ	Y
	1 REL	Y
	1 INX	Y
ON SHARED	--	E
BEFORE REPORTING phrase	1 RPW	N

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
USE statement (continuec)		
FOR DEBUGGING phrase	1 DEB	Y
Procedure-name	1 DEB	Y
ALL PROCEDURES	1 DEB	Y
Cd-name	2 DEB	N
File-name	2 DEB	N
ALL REFERENCE OF identifier-1	2 DEB	N
AFTER DEADLOCK	--	E
WRITE statement	1 SEQ	Y
	1 REL	Y
	1 INX	Y
FROM phrase	1 SEQ	Y
	1 REL	Y
	1 INX	Y
BEFORE/AFTER ADVANCING phrase	1 SEQ	Y
Integer LINE/LINES	1 SEQ	Y
Mnemonic-name	2 SEQ	N
PAGE	1 SEQ	Y
User-figurative-constant	--	E
AT END-OF-PAGE/EOP phrase	2 SEQ	N
INVALID KEY phrase	1 REL	Y
	1 INX	Y
TIMEOUT phrase	--	E

Table I-2. SUMMARY OF DIFFERENCES (continued)

SUMMARY OF DIFFERENCES IN PROCEDURE DIVISION (continued)

ELEMENT	ANSI	VS
SEGMENTATION		
Segment-number may be 0 through 99	1 SEG	C
Fixed segment-number range is 0 through 49	1 SEG	N
Nonfixed segment-number range is 50 through 99	1 SEG	N
Segment-number 0 cannot be overlaid	2 SEG	N
Same segment-numbers must be together	1 SEG	N
Same segment-numbers need not be together	2 SEG	N

APPENDIX J
EXTENSION-RIGHTS

J.1 INTRODUCTION

This appendix describes the use of extension rights. Extension rights is a feature of DMS Sharing that allows a program to request the exclusive right to request resources incrementally, claiming them as needed. Like DMS/TX, the use of extension rights allows multiple users simultaneous access to data. DMS/TX, however, offers a more complete solution to data consistency and concurrency.

Under DMS Sharing, programs using the standard HOLD statement request all the needed resources at the same time. Before additional resources are requested, any resources already held must be released. Extension rights allow a program to hold additional resources without releasing resources already held. However, a program cannot acquire extension rights if it is currently holding resources. In order to request extension rights, a program must first release, by means of the FREE ALL statement, any resources it may be holding.

A program requests extension rights by coding HOLD EXTENSION-RIGHTS. Only one program may hold extension rights at a time. The programmer may code the TIMEOUT and HOLDER-ID phrases to cause the program to wait for the request to be granted and to return the user-ID of the current extension rights holder, if the request cannot be granted in the specified time.

Once extension rights are obtained, the program requests the needed resources by means of the standard HOLD and HOLD LIST statements. It is not necessary to release currently held resources before requesting new resources. The resources held by the program remain held until they are released together (by coding FREE ALL--refer to Section J.2).

Extension rights do not guarantee that the requested resources are not already held by another program, nor do extension rights held by one program prevent other programs from requesting and obtaining resources. When a program obtains extension rights, any resource currently held by another program remains held. Another program can request resources if it is not already holding resources, but it cannot request additional resources unless it releases the resources it already has. The restriction that only one program at a time can have extension rights is imposed to avoid the possibility of deadlock, the condition in which two programs cannot proceed because each is holding resources needed by the other.

HOLD EXTENSION-RIGHTS should be used with extreme caution. Extension rights should be invoked only if the program absolutely requires the claim-as-you-go strategy; that is, if it cannot be determined beforehand what resources will be needed and if none of the resources should be released until all of the held resources have been processed. For example, a customer may be ordering several different items. When updating the order file, it is not known beforehand which inventory records must be updated; yet all the inventory records should be updated before the order processing is completed. Extension rights allow the program to request selected records of the inventory file as needed.

After all needed resources have been identified and held, extension rights should be released as soon as possible, so that another program can obtain extension rights. Extension rights are released by the FREE statement--refer to Section J.2.

To prevent program cancellation in the event of an unsuccessful execution, code the ON ERROR clause with the HOLD EXTENSION-RIGHTS statement. Thus, the ON ERROR clause causes the accompanying imperative statement to be executed and the special register RETURN-CODE to contain the return code.

Figure J-1 is a complete COBOL program illustrating the COBOL statements that support the use of extension rights. In this example, it is not known beforehand which records of EMPLOYEE-FILE are needed, since the value of DEPARTMENT is entered at the workstation. The extension rights allow the program to hold PERSONNEL-FILE and then hold whatever records of EMPLOYEE-FILE are needed, without first releasing PERSONNEL-FILE.

Lines 40 to 46 illustrate the HOLD EXTENSION-RIGHTS statement, with TIMEOUT and HOLDER-ID phrases. If extension rights are held by another program, this program waits 5 seconds (specified in the TIMEOUT phrase). If the extension rights are not released within 5 seconds, the data item WHO-HAS-IT contains the ID of the user running the program that holds the extension rights, and a message is displayed.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  EXRIGHTS.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EMPLOYEE-FILE
      ASSIGN TO "EMPLOYEE",      "DISK",      NODISPLAY,
      ORGANIZATION    IS INDEXED
      ACCESS MODE
      IS DYNAMIC
      RECORD KEY     IS EMPLOYEE-NUMBER.
    SELECT PERSONNEL-FILE
      ASSIGN TO "PERSONS",     "DISK",     NODISPLAY,
      ORGANIZATION     IS INDEXED
      ACCESS MODE      IS DYNAMIC
      RECORD KEY       IS PERSONNEL-RECORD-NUMBER.
DATA DIVISION.
FILE SECTION.
FD  EMPLOYEE-FILE
    LABEL RECORDS ARE STANDARD.
01  EMPLOYEE-RECORD.
    03  EMPLOYEE-NUMBER.
      05 DEPARTMENT PIC 9(3).
      05 FILLER PIC 99.
    03  EMPLOYEE-NAME      PIC X(20).
FD  PERSONNEL-FILE
    LABEL RECORDS ARE STANDARD.
01  PERSONNEL-RECORD.
    03  PERSONNEL-RECORD-NUMBER    PIC 9(5).
    03  PERSONNEL-DATA    PIC X(20).
WORKING-STORAGE SECTION.
77  WHO-HAS-IT        PIC X(3).
PROCEDURE DIVISION.
START-PROGRAM.
    PERFORM HOLD-RIGHTS THRU END-HOLD.
    STOP RUN.
HOLD-RIGHTS.
    OPEN SHARED EMPLOYEE-FILE.
    OPEN SHARED PERSONNEL-FILE.
    HOLD EXTENSION-RIGHTS
         TIMEOUT OF 5 SECONDS
         HOLDER-ID IN WHO-HAS-IT
     DISPLAY WHO-HAS-IT
     " is holding   EXTENSION-RIGHTS."
       GO TO HOLD-RECORDS.
    DISPLAY "EXTENSION-RIGHTS are held by this program.".
```

Figure J-1. Holding Extension Rights in COBOL

```
000047 HOLD-RECORDS.
000048     HOLD RECORDS OF PERSONNEL-FILE FOR RETRIEVAL
000049            TIMEOUT OF 5 SECONDS
000050            HOLDER-ID IN WHO-HAS-IT
000051        DISPLAY WHO-HAS-IT
000052       " is holding PERSONNEL-FILE."
000053       GO TO CLOSE-FILES.
000054     DISPLAY "PERSONNEL-FILE is held by this program.".
000055     ACCEPT DEPARTMENT.
000056     HOLD RECORDS OF EMPLOYEE-FILE FOR UPDATE
000057     WITH KEYS INITIAL 3 CHARACTERS OF EMPLOYEE-NUMBER
000058          TIMEOUT OF 5  SECONDS
000059          HOLDER-ID IN  WHO-HAS-IT
000060        DISPLAY WHO-HAS-IT
000061       " is holding records with first 3 characters of "DEPARTMENT
000062-      " in EMPLOYEE-FILE."
000063     GO TO CLOSE-FILES.
000064     DISPLAY "Records with first 3 characters of "DEPARTMENT " in
000065-      " EMPLOYEE-FILE  are held by this program.".
000066 RELEASE-RIGHTS.
000067     FREE EXTENSION-RIGHTS.
000068     DISPLAY "The EXTENSION-RIGHTS are released, but the resources
000069-      " are still held.".
000070 RELEASE-RESOURCES.
000071     FREE ALL.
000072     DISPLAY "PERSONNEL-FILE and EMPLOYEE-FILE are no longer held
000073-      " by this program, but both files are still open.".
000074 CLOSE-FILES.
000075     CLOSE EMPLOYEE-FILE, PERSONNEL-FILE.
000076 END-HOLD.
000077     EXIT.
```

Figure J-1. Holding Extension Rights in COBOL (continued)

J.2 FREE EXTENSION-RIGHTS STATEMENT

A program should release resources and/or extension rights when the need for them has been satisfied. This is done by coding the FREE statement. If the program does not code the FREE statement, other programs are prevented from obtaining needed resources. The FREE statement can either free all resources including extension rights (FREE ALL) or free extension rights only (FREE EXTENSION-RIGHTS). Thus, by coding FREE EXTENSION-RIGHTS as soon the needed resources have been held, a program can continue to hold the resources for whatever processing might be necessary without preventing another program from obtaining the extension rights.

Figure J-1 illustrates FREE EXTENSION-RIGHTS on line 66 and FREE ALL on line 70.

J.3 HOLD EXTENSION-RIGHTS FORMAT

```
                        [TIMEOUT OF  {data-name-1} [SECOND ]
HOLD EXTENSION-RIGHTS   [            {integer    } [SECONDS]
                        [[ON ERROR imperative-statement-2]
                                [HOLDER-ID IN data-name-2]
                               {imperative-statement-1}  ]
                               {NEXT SENTENCE         }  ]
```

General Rules

1. HOLD EXTENSION-RIGHTS allows a program the exclusive right to hold more than one file and/or a range of records of an indexed file, without implicitly releasing resources already held. No resources are actually held by HOLD EXTENSION-RIGHTS.

2. Only one program can have extension-rights at a time. It is recommended that as soon as the resources have been held, a FREE EXTENSION-RIGHTS be issued so that another program can obtain extension-rights.

3. If the TIMEOUT phrase is specified, and the HOLD EXTENSION-RIGHTS cannot be completed in data-name-1 or integer-1 seconds, then imperative-statement-1 is executed. If the number of seconds specified is zero, the timeout exit will immediately be taken if the HOLD EXTENSION-RIGHTS cannot be completed. If the HOLDER-ID phrase is specified in the TIMEOUT phrase, the logon initials of the user currently holding the extension-rights are moved to data-name-2. Data-name-2 must be defined in the Working-Storage Section or Linkage Section and have a PICTURE of X(3).

4. If the TIMEOUT phrase is not specified and the HOLD EXTENSION-RIGHTS cannot be satisfied, the program waits until the user holding the extension-rights frees them.

5. If there is an ON ERROR clause, any unsuccessful execution of this statement (with non-zero return code) will cause the ERROR imperative statement to be executed. In this situation the special register RETURN-CODE contains the return code of the statement being executed.

6. If there is no ON ERROR clause, the user's program will be cancelled upon sunsuccessful execution of the statement.

J.4 FREE EXTENSION-RIGHTS FORMAT

```
FREE ALL [EXTENSION-RIGHTS]
```

General Rules

1. FREE ALL releases from hold status all resources (including extension-rights) held by the program.

2. FREE ALL EXTENSION-RIGHTS removes extension-rights from the user, but does not release any resources held.

APPENDIX K
SEGMENTATION

K.1 INTRODUCTION

COBOL segmentation is the ability to store different portions of an object program's Procedure Division in the same section of main memory. The segmentation function is not supported in VS COBOL. The VS virtual storage system eliminates the necessity to do so. However, for purposes of compatability with other systems, VS COBOL does supports the Segmentation syntax.

Although it is not required, the Procedure Division of a source program is usually written as a consecutive group of sections, each of which is composed of a series of related operations that perform a particular function. Programs written in this manner thus consist of a group of logical subdivisions. Segmentation allows the programmer to both physically as well as logically divide the Procedure Division.

When segmentation is used, the entire Procedure Division must be written in sections. Each section must then be classified as belonging either to the fixed portion or to the independent portion of the program. Classification is accomplished by the assigning of segment-numbers. Segment-numbers may be any integer from 0 thru 99.

K.1.1 Fixed Portion

The fixed portion is that part of the object program logically treated as if it were always in memory. It cannot be overlayed by any other portion of the program. Fixed portions are assigned segment-numbers from 0 thru 49.

K.1.2 Independent Portion

The independent portion is that part of the object program which can overlay, and be overlaid by, another idependent segment. Independent segments are assigned segment-numbers from 50 thru 99.

A segment is in its initial state if the control mechanisms for all PERFORM and ALTER statements within the segment have not been altered. 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, it is also in its initial state when:

1. The transfer is the result of the implicit transfer of control between consecutive statements from a segment with a different segment-number.

2. The transfer is the result of the implicit transfer of control between a SORT or MERGE statement in a segment with a different segment-number, and an associated input or output procedure in that independent segment.

3. The transfer is an explicit transfer of control to that segment from a segment with a different segment-number (with the exception noted in Rule 2, below).

An independent segment is in its last-used state on subsequent transfer of control when:

1. The transfer is an implicit transfer to that segment from a segment with a different segment-number (except as noted in rules 1 and 2 above).

2. The transfer is an explicit transfer to that segment as a result of the execution of an EXIT PROGRAM statement.

K.1.3 Segmentation Classification

Segment-numbers are assigned according to the segment's logic requirements, frequency of use, and relationship to other segments. Segment-number assignments should be made as follows:

1. Sections which must be available for reference at all times, or which are referrred to very frequently, should be classified as permanent segments. Sections referred to less frequently can be assigned as independent segments.

2. The more frequently a section is referred to, the lower its segment-number.

3. Sections which frequently communicate with one another should be given the same segment-numbers.

K.1.4 Segmentation Control

The logical sequence of a program is the same as the physical sequence, except for specific transfers of control. If any reordering of the object program is required, the compiler provides control transfers to maintain the logical flow specified in the source program. The compiler also provides all controls necessary for a segment to operate whenever the segment is used. It is not mandatory to transfer control to any particular paragraph within a section. Control can be transferred to any paragraph in the Procedure Division.

K.2 SEGMENT-NUMBERS

Section classification is accomplished by means of a system of segment-numbers. The segment-number is included in the section header.

General Format

```
section-name SECTION    [segment-number] .
```

Syntax Rules

1. The segment-number must be an integer ranging from 0 thru 99.

2. If the segment-numbr is ommitted from the section header, the segment-number is assumed to be 0.

3. Fixed segments are assigned segment-numbers 0 thru 49. Independent segments are assigned segment-numbers 50 thru 99. Sections in the declaratives are assigned segment-numbers less than 50.

General Rules

1. All sections having the same segment-number constitute a program segment.

2. Segments with segment-numbers 0 through 49 belong to the fixed portion of the object program.

3. Segments with segment-numbers 50 through 99 are independent segments.

K.3 SEGMENTATION RESTRICTIONS

When segmentation is used, there are restrictions on the ALTER, PERFORM, SORT, and MERGE statements.

K.3.1 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. All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in a fixed segment.

K.3.2 PERFORM Statement

A PERFORM statement that appears in a section that is in a fixed segment can have any declarative sections within its range whose execution is caused within that range. Additionally, that PERFORM statement can have only one of the following:

1. Sections and/or paragraphs wholly contained in one or more fixed segments.

2. 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 any decalartive sections whose execution is caused within that range. Additionally, that PERFORM statement can have only one of the following:

1. Sections and/or paragraphs wholly contained in one or more non-independent segments.

2. Sections and/or paragraphs wholly contained in the same independent segment as that PERFORM statement.

K.3.3 SORT and MERGE Statement

If a SORT or MERGE statement appears in a fixed segment, any input or output procedure referenced by that statement must appear either totally within fixed segments or wholly contained in a single, independent segment.

If a SORT or MERGE statement appears in an independent segment, any input or output procedure referenced by that statement must be contained either totally within fixed segments or wholly within the same independent segment as that SORT or MERGE statement.

DOCUMENT HISTORY

SUMMARY OF CHANGES

FOR THE 5TH EDITION OF THE VS COBOL REFERENCE MANUAL

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
TECHNICAL CHANGES	Advanced Sharing	3-1 to 3-9, 11-47 to 10-49 H-1 to H-3
	PROGRAM-ID paragraph	8-3
	01 and 77 levels	10-3
	Numeric data	10-30
	USAGE clause	10-44
	VALUE IS clause	10-46, 10-47
	OBJECT clause	10-57
	Procedure Division	11-1
	Exponentiation	11-7, 11-8
	DISPLAY AND READ statement	11-37
	OPEN statement	11-67
	PERFORM statement	11-75
	READ statement	11-82
	STOP compiler option	B-4
EDITORIAL CHANGES	Miscellaneous editorial changes	1-1, 1-2, 4-26, 4-30, 10-14, 10-46, 10-56 11-29, B-5, E-7, Index-1 to Index-11

Summary of Changes for the Fourth Edition

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
TUTORIAL DISCUSSION	Introduction to VS COBOL extensions	1-1 to 1-8
	Disk file processing	2-1 to 2-35
	Disk file processing with ADMS	3-1 to 3-33
	Workstation file processing	4-1 to 4-39
	Print file processing	5-1 to 5-6
	Tape file processing	6-1 to 6-3
NEW VS COBOL FEATURES	Advanced data management system	3-1 to 3-33, 10-61 to 10-62 11-46, 11-48 to 11-50, 11-74, 11-75, 11-90, 11-91, 11-118, H-1 to H-7
	Comment lines	7-15
	WITH DEBUGGING MODE clause	9-3, 12-3, 12-4
	SWITCH STATUS clause	9-5, 9-6, 11-13
	NODISPLAY clause	9-12, 9-16, 10-20
	LABEL RECORDS clause	10-14
	INSPECT statement	11-56 to 11-62

Summary of Changes for the Fourth Edition (continued)

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
TECHNICAL CHANGES	SEARCH statement	11-97 to 11-99
	USE FOR DEBUGGING statement	9-3, 11-110, 12-3 to 12-8
	FIPS flagger	B-5
	Parentheses	7-2
	RELATIVE KEY clause	9-11, 9-12
	BUFFER SIZE clause	10-11
	ALTERNATE RECORD KEY clause	9-15, 9-16
	RESERVE nn AREAS clause	9-15, 9-16
	BLOCK CONTAINS clause	10-11
	RECORD CONTAINS clause	10-15, 10-16
	COMPRESSED phrase	10-15, 10-6
	VALUE OF clause	10-17 to 10-20
	OCCURS clause	10-28, 10-52, 10-59
	REDEFINES clause	10-38, 10-39
	VALUE IS clause	10-46 to 10-48, 10-56
	SOURCE or VALUE clause	10-56
	OBJECT clause	10-57, 11-38
	RANGE clause	10-55, 11-38
	Relation conditions	11-9 to 11-12
	ACCEPT statement	11-9 to 11-21
	CALL USING statement	11-2, 11-25, 11-26, I-1 to I-3
	DISPLAY AND READ statement	11-37 to 11-39

Summary of Changes for the Fourth Edition (continued)

TYPE	AFFECTED COBOL FEATURES	AFFECTED PAGES
	MOVE statement	11-63 to 11-66
	MOVE WITH CONVERSION statement	11-66, 11-67
	OPEN statement (consecutive files)	11-69 to 11-71
	Read statement (indexed files)	11-85 to 11-89
	READ WITH HOLD statements	11-83, 11-86, 11-87
	REWRITE statement	11-95, 11-96
	STOP figurative-constant statement	11-106
	WRITE statement (indexed files)	11-115 to 117
	Symbolic debug facility	12-1
	DEBUG-ITEM	12-6 to 12-8
	PMAP compiler option	B-1
	SEPSGN compiler option	B-5
	FILE STATUS codes	E-1 to E-7
EDITORIAL CHANGES	Miscellaneous editorial changes	Throughout

Summary of Changes for the Third Edition

TYPE	DESCRIPTION	AFFECTED PAGES
NEW FEATURES	. condition-names Level 88	48, 51, 52, 66, 67, 70, 91, 92
	. labeled tape support	55, 63
	. additional compiler additions	281
TECHNICAL CHANGES	. VALUE OF clause	55, 63
	. LABEL RECORDS clause	60
EDITORIAL CHANGES	. Miscellaneous editorial changes	63, 66, 67, 92, 279, 280, 281

INDEX

WANG

Customer Comment Form

Publication Number 800-1201-06

Title VS COBOL REFERENCE MANUAL

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other ______________

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other ______________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? ______________

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) ______________

Do you have any other comments or suggestions? ______________

Name ______________ **Street** ______________

Title ______________ **City** ______________

Dept/Mail Stop ______________ **State/Country** ______________

Company ______________ **Zip Code** ________ **Telephone** ______________

Thank you for your help.

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
TECHNICAL PUBLICATIONS
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

The completed order form should be mailed to:
WANG LABORATORIES, INC.
Supplies Division
51 Middlesex St.
No. Chelmsford MA 01863

To Order by Phone, Call:
(800)225-0234
From Mass., Hawaii, and Alaska
(617)256-1400
TELEX 951-743

Order Form for Wang Manuals and Documentation

① Customer Number (If Known)					
② Bill To:			Ship To:		
③ Customer Contact: ( ) ( ) Phone Name			④ Date	Purchase Order Number	
⑤ Taxable Yes ☐ No ☐	⑥ Tax Exempt Number	⑦ Credit This Order to A Wang Salesperson Please Complete	Salesperson's Name	Employee No.	RDB No.
⑧ Document Number	Description	Quantity	⑨ Unit Price	Total Price	
⑩ Authorized Signature / Date			Sub Total		
☐ Check this box if you would like a free copy of the **Corporate Publications Literature Catalog** (700-5294)			Less Any Applicable Discount		
			Sub Total		
			LocalStateTax		
			Total Amount		

Ordering Instructions

1. If you have purchased supplies from Wang before, and know your Customer Number, please write it here.
2. Provide appropriate Billing Address and Shipping Address.
3. Please provide a phone number and name, should it be necessary for WANG to contact you about your order.
4. Your purchase order number and date.
5. Show whether order is taxable or not.
6. If tax exempt, please provide your exemption number.
7. If you wish credit for this order to be given to a WANG salesperson, please complete.
8. Show part numbers, description and quantity for each product ordered.
9. *Pricing extensions and totaling can be completed at your option; Wang will refigure these prices and add freight on your invoice.*
10. Signature of authorized buyer and date.

Wang Supplies Division Terms and Conditions

1. **TAXES** — Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** — Delivery will be F.O.B. Wang's plant. Customer will be billed for freight charges; and unless customer specifies otherwise, all shipments will go best way surface as determined by Wang. Wang shall not assume any liability in connection with the shipment nor shall the carrier be construed to be an agent of Wang. If the customer requests that Wang arrange for insurance the customer will be billed for the insurance charges.
3. **PAYMENT** — Terms are net 30 days from date of invoice. Unless otherwise stated by customer, partial shipments will generate partial invoices.
4. **PRICES** — The prices shown are subject to change without notice. Individual document prices may be found in the Corporate Publications Literature Catalog (700-5294)
5. **LIMITATION OF LIABILITY** — In no event shall Wang be liable for loss of data or for special, incidental or consequential damages in connection with or arising out of the use of or information contained in any manuals or documentation furnished hereunder.

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 NO. CHELSMFORD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
Supplies Division
c/o Order Entry Dept.
M/S 5511
51 Middlesex St.
No. Chelmsford, MA 01863

Fold

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-1201-06
1-84