decsystem10

COBOL
Programmer's Reference Manual

digital

decsystem10

COBOL PROGRAMMER'S REFERENCE MANUAL

This manual reflects the software as of Version 6 of the COBOL compiler and Version 7 of LIBOL, the object-time system.

Additional copies of this manual may be ordered from: Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754
Order Code: DEC-10-LCPRA-A-D

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing August 1969
2nd Printing (Rev) July 1970
Update Pages October 1970
Update Pages December 1970
Update Pages April 1971
Update Pages June 1971
3rd Printing (Rev) October 1971
Update Pages November 1971
Update Pages October 1972
4th Printing (Rev) February 1973
5th Printing (Rev) November 1974

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

CONTENTS

FIGURES

TABLES

Foreword

This manual describes COBOL as it has been implemented in DECsystem-10. Chapter 1 tells how to get started with COBOL in DECsystem-10. Chapter 2 discusses language elements, conventions used in the manual, and the structure of a COBOL program. Chapters 3 through 6 detail the four major divisions of a COBOL program. The COBOL library is described in Chapter 7. I/O processing is discussed in Chapter 8, and Chapter 9 contains information about the Source Library Maintenance program. Several Appendices have been included in the manual. Appendices A and B contain the COBOL reserved words and the character collating sequence. The standard calling sequence, used to link COBOL programs to subprograms written in COBOL and in other languages, is described in Appendix C. Appendix D contains the command strings that can be issued to the COBOL compiler. Appendices E, F, G, H, and I describe several programs that are of use to COBOL programmers: SORT; COBRG, the report generating program; RERUN, the program that restarts interrupted programs; ISAM, the program for maintaining indexed sequential files; and COBDDT, the COBOL debugging program.

It is assumed that the reader has a knowledge of the COBOL language. This manual is intended primarily for reference and is not a tutorial guide for beginning COBOL programmers. Those wishing to learn the COBOL language are referred to the following books.

Farina, Maria V., COBOL Simplified, New Jersey, Prentice Hall, Inc., 1968.

McCameron, Fritz A., COBOL Logic and Programming, Homewood, Illinois, Richard D. Irwin, Inc., 1966.

McCracken, Daniel D. and Garbassi, Umberto, A Guide to COBOL Programming, Second Edition, New York, John Wiley and Sons, Inc., 1970.

The COBOL programmer should be familiar with the operating system commands and the editing languages of DECsystem-10. The commands are discussed in the DECsystem-10 Operating System Commands manual (DEC-10-MRDD-D). TECO is described in the TECO (Text Editor and Corrector Program) Programmer's Reference Manual (DEC-10-ETEE-D). LINED is described in a document in the DECsystem-10 Software Notebook.

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 COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

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

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

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.

They 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.

Chapter 1
Getting Started With DECsystem-10 COBOL

COBOL (Common Business-Oriented Language) is an industry-wide data processing language designed for business applications, such as payroll, inventory control, and accounts-receivable. In COBOL, the programmer can describe his data and the processing of it in simple, English-like statements. COBOL programs are written in terms that are easily recognized by the business user; thus, little programmer training is required, the programs are virtually self-documenting, and programming of desired applications is accomplished quickly and easily.

1.1 DECsystem-10 COBOL

The COBOL implemented for DECsystem-10 conforms to American National Standards Institute (ANSI) specifications as described in the document USA Standard COBOL, X3.23-1968.

DECsystem-10 COBOL operates within the DECsystem-10 operating system, thereby offering the COBOL user the business processing capability of COBOL, in addition to the many features of DECsystem-10. Some of these features are:

a. Batch and timesharing modes of operation,
b. On-line editing and debugging,
c. Sharable compiler and object code,
d. Wide choice of peripheral devices.

1.1.1 Modes of Operation

The DECsystem-10 operating system supports both timesharing and multiprogramming batch modes of operation. The COBOL programmer can use either or both of these operating environments to develop his applications. Under timesharing, a program can be written at an interactive terminal, edited and debugged while the user is on-line, and then run immediately. The turn-around time normally associated with preparing an error-free program can be substantially reduced, because the programmer receives the results of his program immediately and can determine, on the spot, whether or not his program is running properly.

Programs that need no interaction with the user, that require large amounts of data, or that are very long can be entered into the batch system through a noninteractive device such as the card reader or from disk files. Through commands in his batch job, the programmer can specify the processing that he requires, the constraints that must be fulfilled (e.g., deadlines, priorities, and time limits), and the devices that are necessary. The normal tasks required of the timesharing user (e.g., logging in and out) are performed by the batch system. Because there is no interactive dialogue, the batch job is processed faster than it would be under timesharing and the programmer need only wait for his output to be returned to him.

Whether the user needs fast interaction with the system or rapid throughput, the DECsystem-10 operating system can offer him both.

1.1.2 On-Line Editing and Debugging

To develop error-free programs, the COBOL user can take advantage of the system programs TECO and LINED and the program COBDDT. TECO (Text Editor and Corrector) and LINED (Line Editor) are the system editing programs, and COBDDT (COBOL Dynamic Debugging Technique) is the program used for debugging COBOL programs during execution.

LINED is a simple, easily-learned program for editing disk files. It performs editing on sequence-numbered lines in these files. If the file does not have sequence numbers, LINED adds a sequence number to each line. Within a line, the user can add, delete, and change characters by retyping the line. New lines can be added to the file and unwanted lines can be deleted by means of simple LINED commands. If the user desires sequence numbers in his file, or if he has to correct a file that has sequence numbers, he can take advantage of the editing capabilities of LINED. A description of LINED can be found in the Software Notebooks.

TECO is a highly sophisticated and powerful text editor. Unlike LINED, TECO is character-oriented, rather than line-oriented. This means that one or more characters in a line can be edited without the user retyping the line. TECO does not require that sequence numbers be associated with the file, nor does it create sequence numbers for the file unless the user requests them. When complicated corrections must be performed (e.g., changing or deleting a word that occurs repeatedly in a program), the programmer can use TECO to simplify his editing task. An abridged description of TECO that contains the commonly-used commands is given in the Software Notebooks. The full program description is contained in the TECO Programmer's Reference Manual (DEC-10-ETEE-D) in the Software Notebooks.

The timesharing user, when creating programs at his terminal, must use one of the editors so that his programs will be stored as files on disk. Either TECO or LINED can be used for this purpose. If he wants to create sequence-numbered files, the programmer should use LINED; if he does not want sequence numbers, the programmer should use TECO.

COBDDT is a program that allows the user to perform checkout and debugging of his programs. By typing commands to COBDDT, the user can set breakpoints in his program and examine and modify the contents of any location. COBDDT is described in detail in Appendix I.

1.1.3 Sharable Code

The COBOL compiler, like most of the system software, is divided into high and low segments so that most of its code can be shared by many users. The high segment of the compiler is reentrant; that is, it contains code that all COBOL users share. The low segment, containing tables, data names, procedure names, and the like, is unique to each user. This reentrant capability means that only one copy of the compiler must be resident in core at any one time to serve all COBOL users, thus minimizing the amount of core used by COBOL compilations. The operating system (LIBOL) is also sharable to minimize the amount of core used during execution of COBOL programs.

The user can, if he desires, compile his COBOL programs so that they are also sharable. This feature is advantageous if many people need to use the same program simultaneously.

1.1.4 Peripheral Devices

The DECsystem-10 COBOL user has available to him a wide choice of peripheral devices, whether he is operating in timesharing or batch mode. Programs and data can be entered from interactive terminals, DECtape, paper-tape reader, card reader, magnetic tape, or disk. Also, programs and data can be entered from a remote station through any input device at that station. Program listings and program output can be printed on local or remote printers and on the user's interactive terminal. They can be stored on paper tape, cards, DECtape, disk, and magnetic tape either at a remote station or the central station.

The operating system allows the programmer to reference a device with a user-assigned logical name as well as its physical name, thus allowing dynamic assignment of peripheral devices at run-time. For example, within his program, the user could assign his input file to a device named FRED. At run-time, he could then associate any desired input device with the logical name FRED by means of the ASSIGN monitor command.

To prevent slow-speed devices, such as line printers, from being tied-up by individual users, the system allows output to be spooled to these devices. When spooling, the user places his output on disk and issues a monitor command (QUEUE) to place his output into a queue to spool it to the required device. Therefore, no user has to wait for a slow-speed device to become available so that he can run his programs.

DECsystem-10 has been designed to provide maximum speed, efficiency, and ease of operation for a large number of simultaneous batch and timesharing users. Because COBOL has been fully integrated into this system, the COBOL programmer can take advantage of the full range of capabilities provided by DECsystem-10, in addition to the power of the high-level, ANSI-standard, DECsystem-10 COBOL.

1.2 USING COBOL WITH DECsystem-10

Two sample COBOL runs are shown below. The first demonstrates operations in timesharing mode; the second shows batch mode.

1.2.1 COBOL Programming in Timesharing Mode

A sample COBOL program is illustrated and described below, along with all the steps necessary to create, compile, and execute it. This example shows an interactive session, using the timesharing mode of operation. The source program format is the standard DECsystem-10 format, not the conventional COBOL format. Both formats are acceptable to the COBOL compiler; they are described in Chapter 2. The sample run is shown in its entirety first. Each part is then described in detail. The data and output of the program are also shown. Output from the system is underlined in all of the examples.

```
.LOGIN 11,141
JOB 18      5S0220G  SYSTEM #2 TTY26
PASSWORD:
1237        11-JUN-71       FRI


.MAKE PHONEY.CBL
*IIDENTIFICATION DIVISION.
PROGRAM-ID. PHONEY.
REMARKS. CONVERTS A PHONE NUMBER TO A LIST OF ALL
        ALPHABETIC COMBINATIONS REPRESENTING IT.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT LISTING, ASSIGN TO DSK.
        SELECT SORTING, ASSIGN TO DSK, DSK, DSK.
```

```
DATA DIVISION.
FILE SECTION.

FD      LISTING, VALUE OF IDENTIFICATION IS "PHONESLST".
01      PHONES, DISPLAY-7.
        02 PHONE, OCCURS 7 TIMES, PIC X.
        02 FILLER, PIC X(29).
01      MULTI-PHONES, DISPLAY-7.
        02 M-PH, OCCURS 3 TIMES, PIC X(12).

SD      SORTING.
01      SORT-REC.
        02 S-R, PIC X(7).
        02 FILLER, PIC X(29).

WORKING-STORAGE SECTION.

01      ALPHA, PIC X(30), VALUE "000111ABCDEFGHIJKLMNOPRSTUVWXY".
01      CHARS, REDEFINES ALPHA.
        02 CHAR, OCCURS 10 TIMES.
           03 C, OCCURS 3 TIMES, PIC X.

01      NUMBR.
        02 N, OCCURS 8 TIMES, PIC X.

01      INDXS.
        02 INDX, OCCURS 7 TIMES, PIC 9.

01      PHONE-STORE.
        02 PH-STORE, PIC X(7).
        02 FILLER, PIC X(29).

77      I, PIC 99, COMP.
77      J, PIC 99, COMP.
77      K, PIC 99, COMP.
77      L, PIC 99, COMP.

PROCEDURE DIVISION.

MAIN SECTION.
START.
        DISPLAY "TYPE PHONE-NUMBER".
        MOVE SPACES TO NUMBR.
        ACCEPT NUMBR.
        PERFORM GET-NUM VARYING K FROM 4 BY 1 UNTIL K > 7.
        MOVE 1111111 TO INDXS.
        MOVE 1 TO I.
        OPEN OUTPUT LISTING.
        MOVE SPACES TO PHONES.
LOOP.
        PERFORM ASSEMBLE VARYING J FROM 1 BY 1 UNTIL J > 7.
        WRITE PHONES.
        MOVE SPACES TO PHONES.
        PERFORM BUMP-INDX THRU BX.
        GO TO LOOP.
END-IT.
        CLOSE LISTING.
        SORT SORTING ON ASCENDING KEY SORT-REC,
                USING LISTING,
                OUTPUT PROCEDURE THREE-PHONES.
        STOP RUN.
```

```
*       SUBROUTINES

GET-NUM.
        MOVE K TO J.
        SET J UP BY 1.
        MOVE N (J) TO N (K).

ASSEMBLE.
        MOVE INDX (J) TO K.
        MOVE N (J) TO L.
        ADD 1 TO L.
        MOVE C (L, K) TO PHONE (J).

BUMP-INDX.
        MOVE 7 TO J.
B2.     ADD 1 TO INDX (J).
        IF INDX (J) > 3 AND J = 1, GO TO END-IT.
        IF INDX (J) > 3,        MOVE 1 TO INDX (J),
                                SET J DOWN BY 1,
                                GO TO B2.
BX.     EXIT.
*       OUTPUT PROCEDURE FOR SORT

THREE-PHONES SECTION.
START.
        OPEN OUTPUT LISTING.
L-1.
        MOVE 1 TO I.
        MOVE SPACES TO MULTI-PHONES.
L-2.
        RETURN SORTING INTO PHONE-STORE, AT END GO TO END-3-PHONES.
        MOVE PH-STORE TO M-PH (I).
        SET I UP BY 1.
        IF I > 3,       WRITE MULTI-PHONES,
                        GO TO L-1,
        ELSE GO TO L-2.
END-3-PHONES.
        IF I > 1, WRITE MULTI-PHONES.
        CLOSE LISTING.
$EX$$

EXIT

.EXECUTE PHONEY.CBL
COBCL:   PHONEY

LOADING

LOADER 2K CORE
EXECUTION

TYPE PHONE-NUMBER
263-2445

EXIT

.QUEUE LPT:=PHONES.LST
TOTAL OF 44 BLOCKS IN LPT REQUEST
.
```

The first step is logging in to the timesharing system. The monitor types a period to indicate that it is ready to receive a command. The user types the LOGIN command with his project-programmer number. The system returns a job number, the version of the monitor, the system it is running on, and then requests the user's password. The password is not echoed to the terminal to preserve the security of the user's project-programmer number. After the correct password is typed, the system types the date, time, and any pertinent messages.

```
.LOGIN 11,141
JOB 18      5S0220G  SYSTEM #2  TTY26
PASSWORD:

1237        11-JUN-71        FRI
.
```

After logging in, the user creates his COBOL program through one of the editors. TECO is used in this example. The user issues the MAKE monitor command to cause TECO to open a disk file with the specified name and extension. When TECO returns an asterisk, the user can enter his program by typing the I (Insert) command to TECO and immediately following it with the text of his program.

```
.MAKE PHONEY.CBL
*IIDENTIFICATION DIVISION.
```

To end the insert (his program), the user must press the ALTMODE key, which prints as a dollar sign ($). Whatever the user types between the I command and the concluding ALTMODE is stored in his file. To close the file and return to the monitor, the user types EX followed by two ALTMODEs.

```
$EX$$
EXIT
.
```

The sample program requests that the user type a telephone number, and then finds all the possible alphabetic combinations that can represent the number, based on the letters on the telephone dial. For example, the number 2 could be A, B, or C. Note that a 1 or 0 in a phone number does not have an alphabetic equivalent. If either is typed by the user, the number is printed for each combination of letters. All possible combinations are then printed in three groups of seven letters.

Within the FILE SECTION of the program, the record PHONES is redefined by the record MULTI-PHONES, although the REDEFINES clause is not and should not be used. If more than one 01-level record is defined for a file, the 01-level records after the first merely redefine the first. This is done because only one record area is allowed per file.

```
FD      LISTING, VALUE OF IDENTIFICATION IS "PHONESLST".
01      PHONES, DISPLAY-7.
        02 PHONE, OCCURS 7 TIMES, PIC X.
        02 FILLER, PIC X(29).
01      MULTI-PHONES, DISPLAY-7.
        02 M-PH, OCCURS 3 TIMES, PIC X (12).
```

In the FILE-CONTROL section, the user selects devices for sorting. At least three devices are required for sorting. It is recommended that the disk be used because sorting is faster on disk.

```
FILE CONTROL.
        SELECT...
        SELECT SORTING, ASSIGN TO DSK, DSK, DSK.
```

The input to the program is the telephone number that the program requests. The DISPLAY statement within the program causes the request to be typed on the user's terminal.

```
PROCEDURE DIVISION.
MAIN SECTION.
START.
        DISPLAY "TYPE PHONE-NUMBER".
```

To compile, load, and execute his COBOL program, the user issues the EXECUTE monitor command with the filename included in the command string. The system recognizes the COBOL program because of the extension (.CBL). The request for the telephone number is typed when the program begins execution, and the program pauses to await the type-in. To signal that the program has completed execution, the system types EXIT.

```
.EXECUTE PHONEY.CBL
COBOL:   PHONEY
LOADING

LOADER  2K CORE
EXECUTION
TYPE PHONE-NUMBER
263-2445

EXIT
.
```

```
.QUEUE LPT:=*.LPT
TOTAL OF 44 BLOCKS IN LPT REQUEST
```

A portion of the output listing is shown below. The program prints 2,187 combinations on about 12 pages.

```
AMDAGGJ     AMDAGGK     AMDAGGL
AMDAGHJ     AMDAGHK     AMDAGHL
AMDAGIJ     AMDAGIK     AMDAGIL
AMDAHGJ     AMDAHGK     AMDAHGL
AMDAHHJ     AMDAHHK     AMDAHHL
AMDAHIJ     AMDAHIK     AMDAHIL
AMDAIGJ     AMDAIGK     AMDAIGL
AMDAIHJ     AMDAIHK     AMDAIHL
AMDAIIJ     AMDAIIK     AMDAIIL
AMDBGGJ     AMDBGGK     AMDBGGL
AMDBGHJ     AMDBGHK     AMDBGHL
AMDBGIJ     AMDBGIK     AMDBGIL
AMDBHGJ     AMDBHGK     AMDBHGL
AMDBHHJ     AMDBHHK     AMDBHHL
AMDBHIJ     AMDBHIK     AMDBHIL
AMDBIGJ     AMDBIGK     AMDBIGL
AMDBIHJ     AMDBIHK     AMDBIHL
AMDBIIJ     AMDBIIK     AMDBIIL
AMDCGGJ     AMDCGGK     AMDCGGL
AMDCGHJ     AMDCGHK     AMDCGHL
AMDCGIJ     AMDCGIK     AMDCGIL
AMDCHGJ     AMDCHGK     AMDCHGL
AMDCHHJ     AMDCHHK     AMDCHHL
AMDCHIJ     AMDCHIK     AMDCHIL
AMDCIGJ     AMDCIGK     AMDCIGL
AMDCIHJ     AMDCIHK     AMDCIHL
AMDCIIJ     AMDCIIK     AMDCIIL
AMEAGGJ     AMEAGGK     AMEAGGL
AMEAGHJ     AMEAGHK     AMEAGHL
AMEAGIJ     AMEAGIK     AMEAGIL
AMEAHGJ     AMEAHGK     AMEAHGL
AMEAHHJ     AMEAHHK     AMEAHHL
AMEAHIJ     AMEAHIK     AMEAHIL
AMEAIGJ     AMEAIGK     AMEAIGL
AMEAIHJ     AMEAIHK     AMEAIHL
AMEAIIJ     AMEAIIK     AMEAIIL
AMEBGGJ     AMEBGGK     AMEBGGL
AMEBGHJ     AMEBGHK     AMEBGHL
AMEBGIJ     AMEBGIK     AMEBGIL
AMEBHGJ     AMEBHGK     AMEBHGL
AMEBHHJ     AMEBHHK     AMEBHHL
AMEBHIJ     AMEBHIK     AMEBHIL
AMEBIGJ     AMEBIGK     AMEBIGL
AMEBIHJ     AMEBIHK     AMEBIHL
AMEBIIJ     AMEBIIK     AMEBIIL
AMECGGJ     AMECGGK     AMECGGL
```

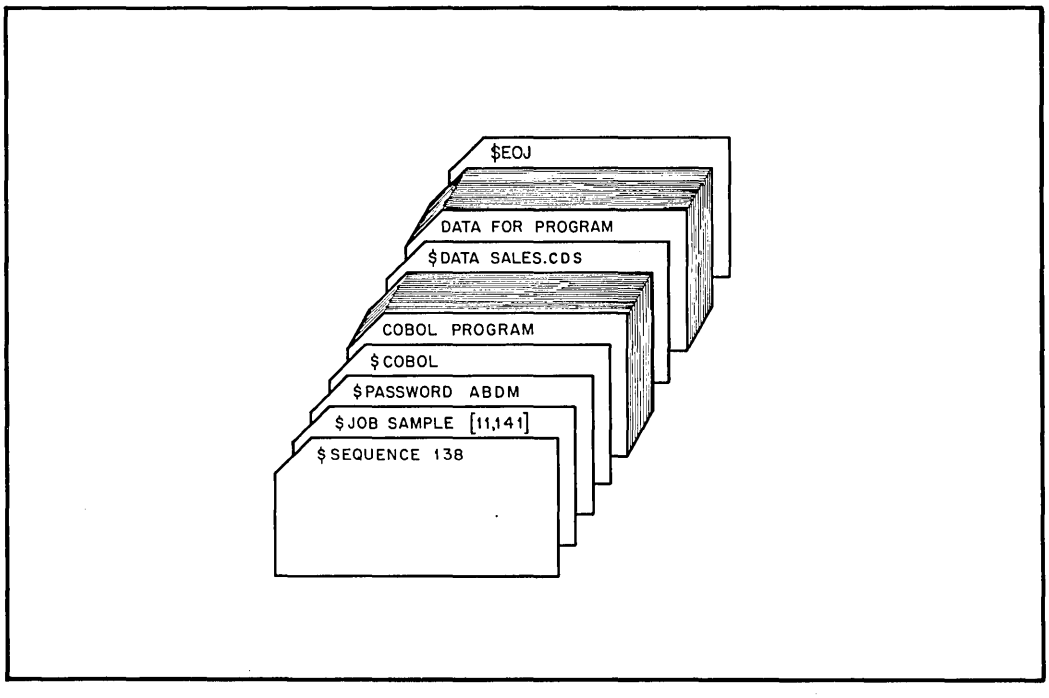
(continued on next page)

```
AMECGHJ     AMECGHK     AMECGHL
AMECGIJ     AMECGIK     AMECGIL
AMECHGJ     AMECHGK     AMECHGL
AMECHHJ     AMECHHK     AMECHHL
AMECHIJ     AMECHIK     AMECHIL
AMECIGJ     AMECIGK     AMECIGL
AMECIHJ     AMECIHK     AMECIHL
AMECIIJ     AMECIIK     AMECIIL
AMFAGGJ     AMFAGGK     AMFAGGL
AMFAGHJ     AMFAGHK     AMFAGHL
```

If the output is satisfactory to the user, he can log off the system or he can continue to perform other tasks at his terminal.

1.2.2 COBOL Programming in Batch Mode

The sample program described in this section shows the steps that are necessary to run a program using the Multiprogramming Batch system. The input card deck, which contains the program, data, and the batch control commands, is shown first. Each part of the sample deck is then described in detail. The output from the program, a listing of the program, and the job's log file are also shown.



10-0731

The programmer writes his program and punches it on cards in ASCII code to be input to the Batch system. The sample program is shown as it has been coded. The standard format is used in this example as in the previous example. The program reads a list of items sold, sorts them into ascending alphabetic order, and

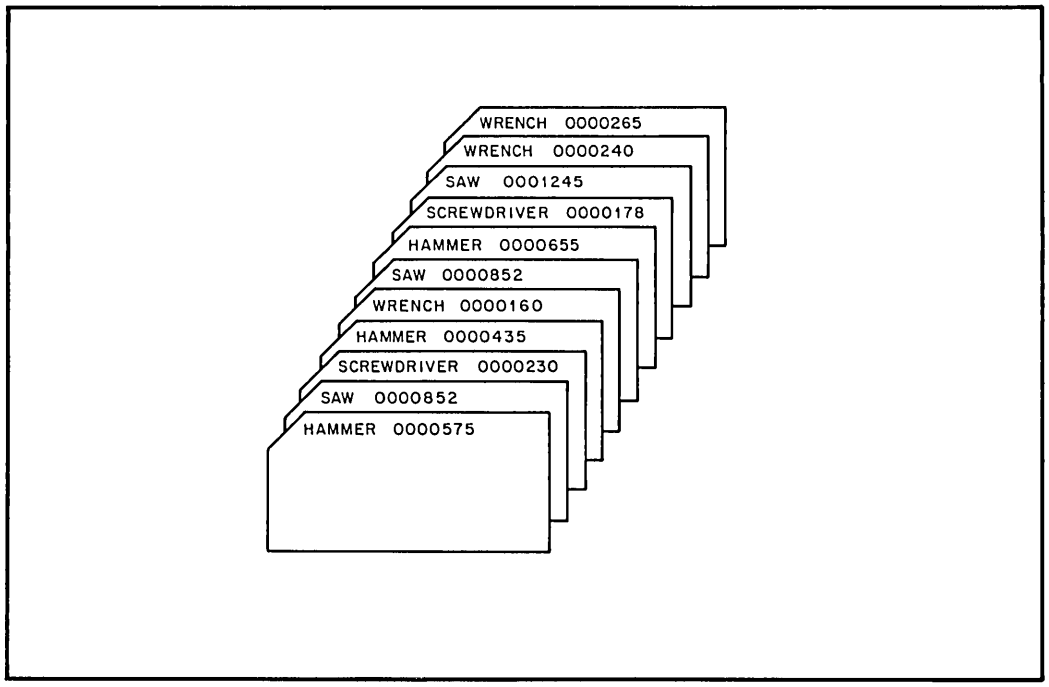
prints a report containing the items grouped according to type, the price of each item, and the subtotal for each group of items. The program requests display of the number of items sold and the gross income from the sales. These are printed in the log file.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPL.
REMARKS. READS A LIST OF SALES AND PRINTS
         A REPORT WITH SUBTOTALS FOR ITEM TYPES.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. PDP-10.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
         SELECT SALES,      ASSIGN TO DSK.
         SELECT TEMP,       ASSIGN TO DSK, DSK, DSK.
         SELECT REPT,       ASSIGN TO LPT.

DATA DIVISION.
FILE SECTION.
FD       SALES,  VALUE OF IDENTIFICATION IS "SALES CDS".
01       SALES-CARD, DISPLAY-7.
         02 ITEM,           PIC X(20).
         02 PRICE,          PIC 9(5)V99.

SD       TEMP.
01       SORT-REC.
         02 ITEM,           PIC X(20).
         02 FILLER,         PIC 9(7).

FD       REPT,   VALUE OF IDENTIFICATION IS "SALES REP".
01       REPT-LINE.
         02 R-ITEM,         PIC X(20).
         02 FILLER,         PIC X(5).
         02 R-PRICE,        PIC Z(8).99.
01       HEADER, PIC X(36).

WORKING-STORAGE SECTION.
01       REC-STORE.
         02 REC-ITM,        PIC X(20).
         02 REC-PR,         PIC 9(5)V99.

77       NUM,              PIC 9(6).
77       GROSS-TMP,        PIC 9(8)V99.
77       GROSS,            PIC Z(8).99.
77       SUBTOT,           PIC 9(8)V99.
77       LAST-ITEM,        PIC X(20).
```

(continued on next page)

```
PROCEDURE DIVISION.
MAIN SECTION.
BEGIN.
        SORT TEMP ON ASCENDING KEY ITEM OF SORT-REC,
                USING SALES,
                OUTPUT PROCEDURE MAKE-REPORT.
        DISPLAY "NUMBER OF ITEMS SOLD      ", NUM.
        MOVE GROSS-TMP TO GROSS.
        DISPLAY "GROSS INCOME              ", GROSS.
        STOP RUN.

*       OUTPUT PROCEDURE FOR SORT

MAKE-REPORT SECTION.
INIT.
        OPEN OUTPUT REPT.
        MOVE ZERO TO NUM, SUBTOT, GROSS-TMP.
        MOVE "SALES REPORT" TO HEADER.
        WRITE HEADER BEFORE ADVANCING 2 LINES.

LOOP.
        RETURN TEMP INTO REC-STORE, AT END GO TO FINIS.
        IF GROSS-TMP = 0, GO TO LOOP-2.
        IF REC-ITM NOT = LAST-ITEM,
                MOVE SPACES TO R-ITEM,
                MOVE SUBTOT TO R-PRICE,
                MOVE ZERO TO SUBTOT,
                WRITE REPT-LINE BEFORE ADVANCING 2 LINES.

LOOP-2.
        SET NUM UP BY 1.
        ADD REC-PR TO GROSS-TMP.
        ADD REC-PR TO SUBTOT.
        MOVE REC-ITM TO R-ITEM.
        MOVE REC-PR TO R-PRICE.
        WRITE REPT-LINE.
        MOVE REC-ITM TO LAST-ITEM.
        GO TO LOOP.

FINIS.
        MOVE SPACES TO R-ITEM.
        MOVE SUBTOT TO R-PRICE.
        WRITE REPT-LINE BEFORE ADVANCING 2 LINES.
        MOVE "TOTAL" TO R-ITEM.
        MOVE GROSS-TMP TO R-PRICE.
        WRITE REPT-LINE.
        CLOSE REPT.
```

The data for the program, which is a group of items sold, is also placed on cards. This data is shown on the following page.



10-0732

The programmer sets up his job as an input deck containing his program, data, and Batch control commands on cards. The following Batch commands are used for this job.

```
$SEQUENCE 138
```

10-0733

The $SEQUENCE command contains the job's sequence number. This command is an installation option; if a sequence number is used, it is assigned by the installation.

```
$JOB SAMPLE [11,141]
```

10-0734

The $JOB command contains the name of the job and the user's project-programmer number. Several other parameters can be used in this command to control processing. For example, the user can assign a priority

for the job, a central processor time limit, a limit to the number of pages printed, or a deadline time. In this example, the defaults for all parameters are assumed. The parameters in the $JOB command, as well as all of the Batch control commands, are described in the DECsystem-10 Operating System Commands manual (DEC-10-MRDC-D).

```
$PASSWORD ABDM
```

10-0736

The $PASSWORD command contains the password associated with the project-programmer number in the $JOB command. This command is also an installation option.

```
$COBOL
```

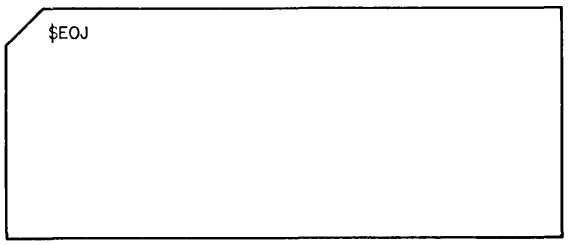
10-0737

The $COBOL command is followed immediately by the deck containing the program. The $COBOL command causes the program to be copied into a disk file with a name supplied by the Batch system. When the job is run, the program is compiled, and a listing is produced. The listing is printed with the output of the job.

```
$DATA SALES.CDS
```

10-0735

The $DATA command is immediately followed by the deck containing the data. The data is copied to a file on disk named SALES.CDS, the name that was in the $DATA command. The $DATA command causes the program (or programs) that precedes it to be executed when the job is run by the batch controller.



10-0739

The end of the job is signalled by the $EOJ card.

The user inputs his job deck through the card reader and then simply waits for his output. The Multiprogramming Batch system reads and separates the cards into disk files, runs the job, and spools the output to the line printer. The output is in the form of a printed listing that begins with a header page and ends with a trailer page. Both the header and trailer contain the user's name, project-programmer number, date, time, system name, and monitor version. Between the header and trailer are the compilation listing, the loader map, the output from the program, and the log file of the job. With the exception of the header and trailer pages and the loader map, the listing is shown. Note that the compiler assigns sequence numbers to the compilation listing even though the user did not place sequence numbers on his program.

```
P R O G R A M   E X A M P L
LN6TPN.CBL    31-OCT-73  11:37

0001     IDENTIFICATION DIVISION.
0002     PROGRAM-ID. EXAMPL.
0003     REMARKS. READS A LIST OF SALES AND PRINTS
0004              A REPORT WITH SUBTOTALS FOR ITEM TYPES.
0005     ENVIRONMENT DIVISION.
0006     CONFIGURATION SECTION.
0007     SOURCE-COMPUTER. DECsystem-10.
0008     INPUT-OUTPUT SECTION.
0009     FILE-CONTROL.
0010              SELECT SALES,    ASSIGN TO DSK.
0011              SELECT TEMP,     ASSIGN TO DSK, DSK, DSK.
0012              SELECT REPT,     ASSIGN TO LPT.
0013     DATA DIVISION.
0014     FILE SECTION.
0015     FD        SALES,  VALUE OF IDENTIFICATION IS "SALES CDS".
0016     01        SALES-CARD, DISPLAY-7.
0017              02 ITEM,          PIC X(20).
0018              02 PRICE,         PIC 9(5)V99.
0019     SD        TEMP.
0020     01        SORT-REC.
0021              02 ITEM,          PIC X(20).
0022              02 FILLER,        PIC 9(7).
0023     FD        REPT,    VALUE OF IDENTIFICATION IS "SALES REP".
0024     01        REPT-LINE.
0025              02 R-ITEM,        PIC X(20).
0026              02 FILLER,        PIC X(5).
0027              02 R-PRICE,       PIC Z(6).99.
0028     01        HEADER, PIC X(36).
```

(continued on next page)

```
0029    WORKING-STORAGE SECTION.
0030    Z1       REC-STORE.
0031             02 REC-ITM,      PIC X(20).
0032             02 REC-PR,       PIC 9(5)V99.
0033    77        NUM,            PIC 9(6).
0034    77        GROSS-TMP,      PIC 9(8)V99.
0035    77        GROSS,          PIC Z(8).99.
0036    77        SUBTOT,         PIC 9(8)V99.
0037    77        LAST-ITEM,      PIC X(20).
0038    PROCEDURE DIVISION.
0039    MAIN SECTION.
0040    BEGIN.
0041             SORT TEMP ON ASCENDING KEY ITEM OF SORT-REC,
0042                      USING SALES,
0043                      OUTPUT PROCEDURE MAKE-REPORT.
0044             DISPLAY "NUMBER OF ITEMS SOLD   ", NUM.
0045             MOVE GROSS-TMP TO GROSS.
0046             DISPLAY "GROSS INCOME           ", GROSS.
0047             STOP RUN.
0048  *          OUTPUT PROCEDURE FOR SORT
0049    MAKE-REPORT SECTION.
0050    INIT.
0051             OPEN OUTPUT REPT.
0052             MOVE ZERO TO NUM, SUBTOT, GROSS-TMP.
0053             MOVE "SALES REPORT" TO HEADER.
0054             WRITE HEADER BEFORE ADVANCING 2 LINES.
0055    LOOP.
0056             RETURN TEMP INTO REC-STORE, AT END GO TO FINIS.
0057             IF GROSS-TMP = 0, GO TO LOOP-2.
0058             IF REC-ITM NOT = LAST-ITEM,
0059                      MOVE SPACES TO R-ITEM,
0060                      MOVE SUBTOT TO R-PRICE,
0061                      MOVE ZERO TO SUBTOT,
0062                      WRITE REPT-LINE BEFORE ADVANCING 2 LINES.
0063    LOOP-2.
0064             SET NUM UP BY 1.
0065             ADD REC-PR TO GROSS-TMP
0066             ADD REC-PR TO SUBTOT.
0067             MOVE REC-ITM TO R-ITEM.
0068             MOVE REC-PR TO R-PRICE.
0069             WRITE REPT-LINE.
0070             MOVE REC-ITM TO LAST-ITEM.
0071             GO TO LOOP.
0072    FINIS.
0073             MOVE SPACES TO R-ITEM.
0074             MOVE SUBTOT TO R-PRICE.
0075             WRITE REPT-LINE BEFORE ADVANCING 2 LINES.
0076             MOVE "TOTAL" TO R-ITEM.
0077             MOVE GROSS-TMP TO R-PRICE.
0078             WRITE REPT-LINE.
0079             CLOSE REPT.

NO ERRORS DETECTED
```

```
SALES REPORT

HAMMER                          5.75
HAMMER                          4.35
HAMMER                          6.55
                               16.65

SAW                             8.52
SAW                            12.45
SAW                             8.52
                               29.49

SCREW DRIVER                    2.30
SCREW DRIVER                    1.78
                                4.08

WRENCH                          1.60
WRENCH                          2.65
WRENCH                          2.40
                                6.65

TOTAL                          56.87


15:43:52 STDAT  11-OCT-84      R57AV  SYS #40/2  SPRINT Version 2(1035) Running on CDR0
15:43:52 STCRD  $SEQUENCE 138
15:43:52 STCRD  $JOB SAMPLE [27,235]
15:44:00 STCRD  $COBOL
15:44:05 STMSG  File  DSK:LN6BB5.CBL  Created - 79 Cards Read - 11 Blocks Written
15:44:05 STCRD  $DATA SALES.CDS
15:44:07 STMSG  File  DSK:SALES.CDS  Created - 11 Cards Read - 2 Blocks Written
15:44:08 STCRD  $EOJ
15:44:08 STSUM  End of Job Encountered
15:44:08 STSUM  96 Cards Read
15:44:08 STSUM  Batch Input Request Created


15:44:11 BAJOB  BATCON Version 12(1041) running SAMPLE sequence 138 in stream 1
15:44:11 BAFIL  Input from DSKC0:SAMPLE.CTL[27,235]
15:44:11 BAFIL  Output to  DSKC0:SAMPLE.LOG[27,235]
15:44:11 BASUM  Job parameters
                Time:00:05:00   Unique:YES      Restart:YES

15:44:11 MONTR
15:44:11 MONTR  .LOGIN 27/235 /SPOOL:ALL/TIME:300/LOCATE:1/NAME:"SMITH "
15:44:13 USER   JOB 22     R57AV  SYS #40/2 TTY145
15:44:14 USER   1544    11-Oct-84       Thur
15:44:15 MONTR
15:44:15 MONTR  ..COMPIL /COMP/COB DSK:LN6BB5.CBL/LIST
15:44:16 USER   COBOL:  EXAMPL  [LN6BB5.CBL]
15:44:31 MONTR
15:44:31 MONTR  EXIT
15:44:31 MONTR
15:44:31 MONTR  ..EXECUT /REL DSK:LN6BB5.REL
15:44:33 USER   LINK:   Loading
15:44:33 USER   [LNKXCT EXAMPL Execution]
15:44:38 USER   NUMBER OF ITEMS SOLD   11
15:44:38 USER   GROSS INCOME                  56.87
15:44:39 MONTR
15:44:39 MONTR  EXIT
```

(continued on next page)

```
15:44:39 MONTR
15:44:39 MONTR  .
15:44:39 BLABL  %FIN::
15:44:39 MONTR  .DELETE DSK:LN6BB5.CBL,DSK:LN6BB5.REL
15:44:40 USER   Files deleted:
15:44:41 USER   LN6BB5.CBL
15:44:41 USER   11 Blocks freed
15:44:42 USER   LN6BB5.REL
15:44:42 USER   08 Blocks freed
15:44:42 MONTR
15:44:42 MONTR  .KJOB DSKC0:SAMPLE.LOG=/W/P/Z:4/VR:132/VS:138/VL:200/VD:D
15:44:46 K-QUE  Total of 18 blocks in 3 files in LPTS1 request
15:44:49 LGOUT  Job 22, User [27,235]  Logged off TTY145   1544  11-Oct-84
15:44:49 LGOUT  Saved all files (20 blocks)
15:44:49 LGOUT  Runtime 7.89 Sec
15:46:46 LPMSG  LPTSPL Version 6(344)  Running on LPT0
15:46:53 LPMSG  Job SAMPLE file DSKC0:LN6BB5[27,235] for [27,235] started
15:47:07 LPMSG  DSKC0:LN6BB5[27,235] Done
15:47:08 LPMSG  Job SAMPLE file DSKC0:SALES[27,235] for [27,235] started
15:47:17 LPMSG  DSKC0:SALES[27,235] Done
```

Chapter 2
Introduction to COBOL Language

The conventions, special terms, language elements, and formats acceptable to COBOL are described below to aid the user in writing COBOL programs. The source language statements are discussed in subsequent chapters.

2.1 SYMBOLS AND TERMS

The symbols and terms used in the following chapters of this manual are either necessary to describe the language or are commonly-used COBOL terms.

2.1.1 Symbols

The symbology used in this manual to illustrate the various COBOL statement formats is essentially the same as that used in other COBOL language manuals and is based on the CODASYL COBOL reference document.

Symbology	Meaning
Lower-case characters	Represent information that must be supplied by the programmer, such as values, names, and other parameters.
Upper-case characters, underscored	Key words in the COBOL lexicon that must be used when the formats of which they are a part are used.
Upper-case characters, not underscored	Other words in the COBOL lexicon that serve only to make the COBOL statement more readable. Their use is optional and has no effect on the meaning of the formats of which they are a part.
Braces	Indicate that a choice must be made from the two or more lines enclosed.

(continued on next page)

Symbology	Meaning
Brackets	Indicate an optional feature. The contents of the brackets are used according to the rules above if the feature is desired.
Ellipsis ...	Indicates that the information contained within the preceding pair of braces or brackets can be repeated at the programmer's option.

2.1.2 COBOL Terms

The terms block, record, and item have special meanings when used in a COBOL program.

Term	Meaning
Block	Signifies a logical grouping of records. This term commonly refers to a logical block of records on some storage medium.
Record	Signifies a logical unit of information. In relation to a data file, a record is the largest unit of logical information that can be accessed and processed at a time. Records can be subdivided into fields or items.
Item	Signifies a logical field or group of fields within a record. A group item is one that is further broken down into subitems (e.g., a group item called TAX might be broken down into subitems called FED-TAX and STATE-TAX). Subitems can be further broken down into other subitems. An item that has no subitems is called an elementary item.

2.2 ELEMENTS OF COBOL LANGUAGE

2.2.1 Program Structure

A COBOL program consists of four divisions. Within each division are the program statements; some are required, others are optional.

Division	Meaning
IDENTIFICATION DIVISION	Identifies the source program.
ENVIRONMENT DIVISION	Describes the computer on which the source program is to be compiled, the computer on which the object program is to run, and certain relationships between program elements and hardware devices.
DATA DIVISION	Describes the data to be processed by the object program.
PROCEDURE DIVISION	Describes the actions to be performed on the data.

2.2.2 Character Set

Within a source program statement, all ASCII characters are valid except:

a. null, delete, and carriage return (which are ignored);

b. line feed, vertical tab, form feed, and the printer control characters (20_8 through 24_8), which mark the end of a source line;

c. Control-Z, which marks the end-of-file.

The lower case ASCII characters are translated to upper case characters except when they appear in non-numeric literals.

Of this character set, 37 characters (the digits 0 through 9, the 26 letters of the alphabet, and the hyphen) can be used by the programmer to form COBOL words, such as data-names, procedure-names, and identifiers.

Punctuation characters include:

Δ	(space)	" or '	(quotation mark)
,	(comma)	(	(left parenthesis)
;	(semicolon)	)	(right parenthesis)
.	(period)	→|	(horizontal tab)

Special editing characters include:

+	(plus sign)	*	(check protection symbol)
-	(minus sign)	Z	(zero suppression)
$	(dollar sign)	B	(blank insertion)
,	(comma)	0	(zero insertion)
.	(decimal point)	CR	(credit)
		DB	(debit)

Special characters used in arithmetic expressions include:

+	(addition)	/	(division)
-	(subtraction)	**	(exponentiation)
*	(multiplication)	↑	(exponentiation)

Special characters used in conditional (IF) statements include:

=	(equal)	>	(greater than)	<	(less than)

2.2.3 Words

A COBOL word is composed of not more than 30 characters chosen from the 37 characters given in the previous section. A word is terminated by a space, period, right parenthesis, comma, semicolon, or horizontal tab. If the terminator is not a space or horizontal tab, at least one space or tab must follow the terminator.

Words used in writing COBOL source programs are of two types: COBOL reserved words and user-created words.

2.2.3.1 COBOL Reserved Words - COBOL reserved words are those words that constitute the COBOL lexicon and have a special meaning to the compiler (e.g., DIVISION, PROCEDURE, ADD); these words are listed in Appendix A. They include all the COBOL division, section, and paragraph names, descriptive clauses, procedure verbs, certain prepositions, figurative constants, and special registers. Reserved words must be spelled and used exactly as shown in the formats given in this manual.

2.2.3.1.1 Figurative Constants - Figurative constants are reserved words that specify certain fixed values. When these reserved words are to be used as figurative constants, they must not be enclosed in quotation marks; otherwise they will be treated by the compiler as nonnumeric literals.

The figurative constants are given below. Except for one case (the ALL constant), singular and plural forms are given; these forms are equivalent and can be used interchangeably.

Figurative Constant	Use
ZERO ZEROS ZEROES	Represents the value 0 or one or more of the character 0 depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	For DISPLAY-6 and DISPLAY-7 items this represents the highest value in the collating sequence. For COMP and COMP-1 items, this represents the largest number that can be placed in the machine word(s) containing the item.
LOW-VALUE LOW-VALUES	For DISPLAY-6 and DISPLAY-7 items, this represents the lowest value in the collating sequence. For COMP and COMP-1 items, this represents the smallest number (most negative) that can be placed in the machine word(s) containing the item.

(continued on next page)

Figurative Constant	Use
QUOTE QUOTES	Represents one or more quotation marks ("). It can be used anywhere that the quotation mark character (") is valid, except to delimit nonnumeric literals (see Nonnumeric Literals). QUOTE(S) is frequently used where an actual quotation mark character would erroneously appear to delimit a nonnumeric literal. For example, if the user wanted his program to type out the exact character string MOUNT TAPE LABELLED "MASTER" ON DRIVE 3 he could use the procedure statement DISPLAY "MOUNT TAPE LABELLED " QUOTE "MASTER" QUOTE " ON DRIVE 3".
ALL any-literal	Represents repetitions of the string of characters that constitute either a nonnumeric literal or a figurative constant (other than ALL any-literal). If a figurative constant is used, the ALL is redundant; thus, ZEROS and ALL ZEROS are equivalent.

Figurative constants generate a string of characters whose length is determined, based on context, by the compiler. For example, if TOTAL-AMOUNT is a five-character field, the procedure statement MOVE ALL ZEROS TO TOTAL-AMOUNT moves a string of five zeros to the field TOTAL-AMOUNT; MOVE ALL "AB" TO TOTAL-AMOUNT moves "ABABA" to TOTAL-AMOUNT. If the length cannot be determined by context, a single character (or a single-character sequence, in the case of ALL) is generated. For example, the procedure statement DISPLAY ALL QUOTES will result in the output of a single quotation mark (") to the user's terminal.

Examples of Use of Figurative Constants:

DATA DIVISION Usage:	02 AMOUNT PICTURE IS 9999.99 VALUE IS ZERO. 04 MESSAGE PICTURE IS A(10) VALUE IS SPACES.
PROCEDURE DIVISION Usage:	MOVE ZEROS TO AMOUNT. (Puts the value of zero in the AMOUNT field) MOVE SPACES TO MESSAGE. (Puts spaces in the MESSAGE field) IF TOTAL IS EQUAL TO ZERO.... EXAMINE FLD-A TALLYING LEADING ZEROS.

2.2.3.1.2 Special Registers - In addition to figurative constants, COBOL recognizes two other special reserved-word constants: TALLY and TODAY.

TALLY is the name of a fixed five-digit signed computational field. It is used primarily to hold information produced by the EXAMINE verb. However, the programmer can utilize TALLY in any situation where a signed numeric field is valid (e.g., temporary storage of any integer value of five or fewer digits).

TODAY is a 12-character alphanumeric display field that contains the current date and time. Its format is:

yymmddhhmmss

where	yy is the year (last two digits)	hh is the hour
	mm is the month	mm is the minute
	dd is the day	ss is the second

2.2.3.2 User-Created Words - User-created words are labels for the various parts of the user's data (files, records, and fields) and the user's procedures (sections and paragraphs). They can contain only the symbols 0 through 9, A through Z, and the hyphen. With the exception of procedure names, they cannot be all digits. A user-created word can neither begin nor end with a hyphen.

User-created words can be further subdivided into several categories. To understand the remainder of this manual, the user should be familiar with the following types of words.

data-name	The user-created name assigned to an item (field) within a record.
file-name	The user-created name assigned to a data file.
record-name	The user-created name assigned to a data record within a file.
procedure-name	The user-created name assigned to a paragraph or section in the PROCEDURE DIVISION. When assigned to a section, it is referred to as a section-name; and when assigned to a paragraph, it is referred to as a paragraph-name.
identifier	A user-created name used in PROCEDURE DIVISION statement formats to indicate a data-name followed, as required, by the syntactically correct combination of qualifiers, and/or subscripts, and/or indexes necessary to make reference to a unique item of data.
mnemonic-name	A user-created name assigned to a hardware device or a report code.
condition-name	A user-created name assigned to a value or range of values of the associated data item. Condition-name can also be assigned to console switch settings.
index-name	A user-created name defined in the INDEXED BY clause (see OCCURS in Chapter 5). Its function is identical to that of an index data-name.
index data-name	A user-created name defined with USAGE INDEX. Its function is identical to that of an index-name.

2.2.4 Literals

A literal is a string of characters, the value of which is identical to the characters that compose the literal. Literals are of two types: numeric and nonnumeric.

2.2.4.1 Numeric Literals - A numeric literal is a string of 1 to 18 numeric characters (0 through 9). It cannot contain any alphabetic characters. It can be preceded by a plus sign (+) or a minus sign (-); if no sign is used, the literal is assumed to be positive. A decimal point can appear anywhere in the literal except to the left of the sign or as the rightmost character. If no decimal point is used, the literal is assumed to be an integer. A numeric literal is considered to be of the numeric class; that is, it can be used legitimately as a value in arithmetic expressions.

Examples of Numeric Literals:

```
123          -123          +123          1.23456          .123456789
-.123456789          1234567890.12345678          -1234567890.12345678
```

2.2.4.2 Nonnumeric Literals - Nonnumeric literals are character strings containing from 1 to 120 characters enclosed in single or double quotation marks. The value of the literal is equal to the characters, including any spaces, enclosed by the quotation marks. Note that the compiler will accept either single or double quotation marks to enclose a literal; however, the opening and closing quotation marks must be the same type, either single or double. Any ASCII character except the quotation mark, null, delete, carriage return, and printer control can appear within a literal.

All nonnumeric literals are considered to be in the alphanumeric category; they cannot be used as values in arithmetic operations, and numeric editing cannot be performed on them. If a literal conforms to the rules for formation of a numeric literal, but is enclosed in quotation marks, it is considered to be a nonnumeric literal. That is, "120.45" is not equivalent to 120.45.

Examples of Nonnumeric Literals:

```
"A"    'THIS ACCOUNT HAS A CREDIT BALANCE'    "  RETURN  "
"-125.50"    'DEDUCT 10% IF PAID BEFORE JAN. 31ST'
```

2.2.5 Punctuation

The punctuation that can be used in source programs includes the space, comma, semicolon, and period.

The space is used to separate words, phrases and clauses. The comma and semicolon can be used interchangeably within a program to improve the appearance of the program. However, both the comma and the semicolon are treated as spaces by the compiler; they can be used any place in the program where a space is expected.

The period is used to terminate a division name, a section name, and a paragraph name. It is also used in the PROCEDURE DIVISION to terminate sentences. Paragraphs and sections are terminated by the period ending the last sentence of the paragraph or section. In the DATA DIVISION, a period must be placed after the description of a data item. Examples of the use of periods are:

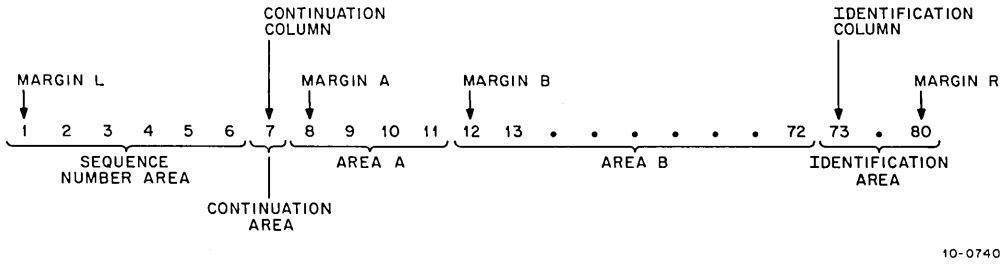
```
PROCEDURE DIVISION.
      :
INPUT SECTION.
      :
READ INFILE AT END GO TO ENDER.
      :
      :
DATA DIVISION.
FILE SECTION.
      :
      :
01 MYDATA PICTURE IS X(10).
```

2.3 SOURCE PROGRAM FORMAT

Two source program formats are acceptable to DECsystem-10 COBOL: conventional format and standard format. The compiler assumes that the source program is written in standard format unless the /S switch is included in the command string to the compiler (refer to Appendix D) or the special sequence numbers created by LINED are detected by the compiler.

2.3.1 Conventional Format

The conventional format is used when the programmer wishes his source programs to be compatible with other COBOL compilers. It is the format that is normally used when input is from the card reader. A line of conventional format is shown below; the numbers refer to card columns, although this format can be used with any input medium.



Margin L designates the leftmost (first) character position of a line.

The continuation column designates the seventh character position relative to the left margin.

Margin A designates the eighth character position relative to the left margin.

Margin B designates the twelfth character position relative to the left margin.

The identification column designates the seventy-third character position relative to the left margin.

Margin R designates the rightmost (eightieth) character position of a line.

The sequence number area is a six-character field beginning at margin L that normally contains a sequence number. The compiler ignores this field.

The continuation area occupies one character position in the continuation column. The continuation area is used whenever it is necessary to split a word, a numeric literal, or a nonnumeric literal between the end of one line and the beginning of the next, or when a comment line is to be inserted. The following rules apply to comment or continuation lines.

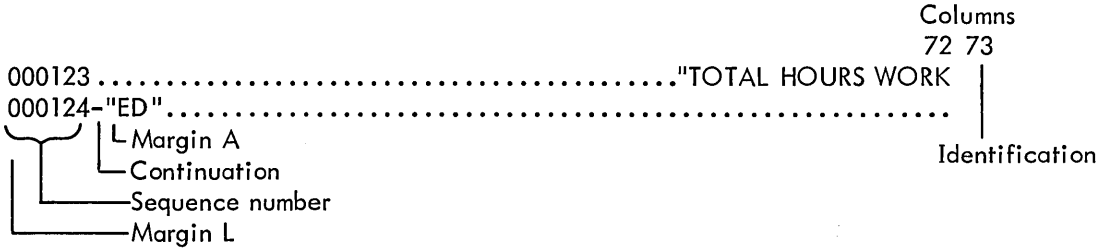
a. The programmer can insert a comment line in a program by placing an asterisk (*) in the continuation area.

b. To continue a line without splitting a word or literal, the programmer must begin the first continuing word on the second line at, or after, margin A. The continuation area is left blank. As many spaces as desired can follow the last word on the first line, or the word can continue up to the identification column (column 73).

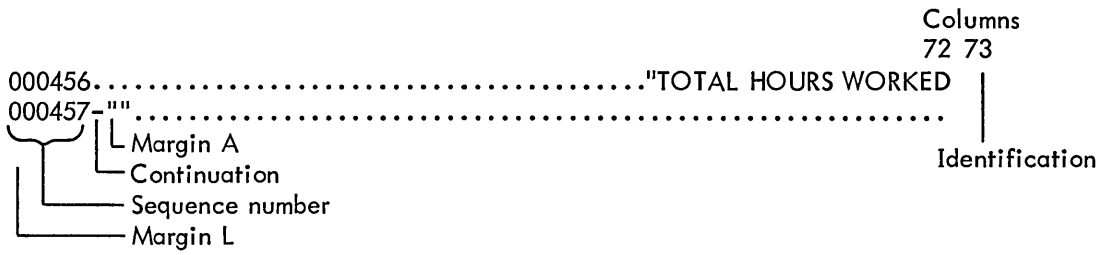
c. To split a word or numeric literal from one line to the next, the programmer places a hyphen in the continuation area of the second line and begins the first continuing character of the word or literal at, or after, margin A. As many spaces as desired can occur after the last character of the word or numeric literal on the first line, or the last character can occur immediately before the identification column.

d. To split a nonnumeric literal between two lines, the user must ensure that the last character to be placed on the first line occurs immediately before the identification column. Otherwise, all spaces following this character on the first line will be treated as part of the literal. A hyphen is placed in the continuation column of the second line, and a quotation mark is placed at, or after, margin A, followed by the first continuing character of the literal. In cases where the last character of the literal appears immediately before the identification column on the first line and only the quotation mark that ends the literal remains to be entered, the same rule applies (a hyphen in the continuation field, a quotation mark at, or after, margin A followed by the terminating quotation mark).

Examples



(continued on next page)



e. A continuation line cannot be immediately preceded by a blank line or a comment line.

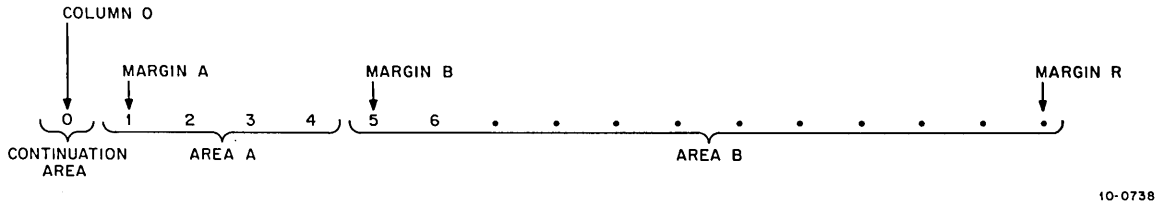
Area A occupies four character positions beginning at margin A. All division-names, section-names, and paragraph-names must begin in area A. In the DATA DIVISION, the FD entry must begin in Area A and level-number entries can begin in area A, but are not required to.

Area B occupies 61 character positions beginning at margin B and ending at column 72. All remaining entries begin in area B.

The identification area occupies eight character positions beginning at the identification column and ending at margin R. This area is reserved as an identification field into which any combination of eight or fewer characters can be punched to identify the card deck. This identification is printed on a listing of the source program.

2.3.2 Standard Format

The standard format is provided for those programmers who are more familiar with the format normally used in DECsystem-10 operations. It differs from the conventional format in that sequence numbers and identification are not used, because most DECsystem-10 programs do not require either. A line in the standard format is shown below; the numbers represent character positions.



Column 0 designates a character position that is not counted by the compiler. It is only used for comment or continuation. A hyphen, asterisk, or space in column 0 is recognized by the compiler, but it is not counted by the compiler as a character position. Thus, if the user typed a space, hyphen, or asterisk in column 0 and then typed three additional spaces, he would still be in area A, not at margin B. In other words, five spaces or a horizontal tab is required to move to margin B.

Margin A designates the first character position.

Margin B designates the fifth character position relative to Margin A (not column 0). To reach margin B, the user should type horizontal tab.

Margin R designates the rightmost character position of a line.

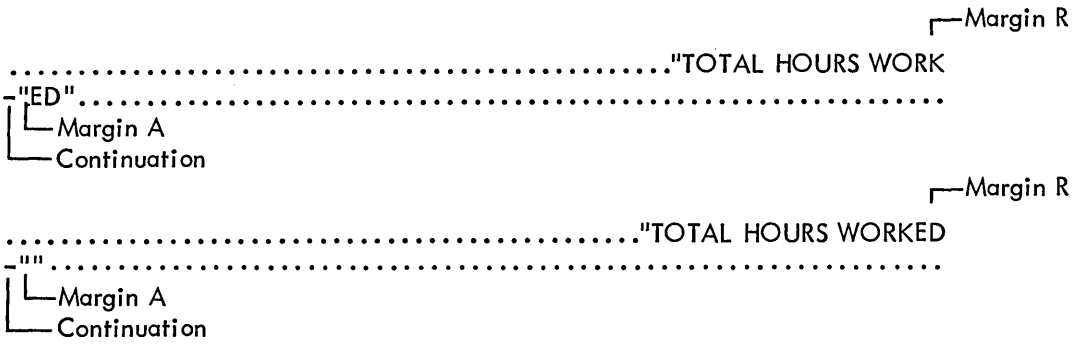
The continuation area occupies one character position in column 0. The continuation area is used whenever it is necessary to split a word, a numeric literal, or a nonnumeric literal between the end of one line and the beginning of the next, or when a comment line is to be inserted. The following rules apply to comment or continuation lines.

a. The programmer can insert a comment line in a program by placing an asterisk (*) in the continuation area.

b. To continue a line without splitting a word or literal, the programmer must begin the first continuing word on the second line at, or after, margin A. The continuation area is left blank. As many spaces as desired can follow the last word on the first line, or the word can continue up to margin R.

c. To split a word or numeric literal from one line to the next, the programmer places a hyphen in the continuation area of the second line, and begins the first continuing character of the word or literal at or after margin A. As many spaces as desired can occur after the last character of the word or numeric literal on the first line, or the last character can occur at margin R.

d. To split a nonnumeric literal between two lines, the user must ensure that the last character to be placed on the first line occurs at margin R. Otherwise, all spaces following this character on the first line will be treated as part of the literal. A hyphen is placed in the continuation column of the second line, and a quotation mark is placed at, or after, margin A, followed by the first continuing character of the literal. In cases where the last character of the literal appears at Margin R on the first line and only the quotation mark that ends the literal remains to be entered, the same rule applies (a hyphen in the continuation field, a quotation mark at, or after, margin A followed by the terminating quotation mark).



e. A continuation line cannot be immediately preceded by a blank line or a comment line.

Area A occupies four character positions beginning at margin A. All division-names, section-names, and paragraph-names must begin in area A. In the DATA DIVISION, the FD entry must begin in Area A and level-number entries can begin in area A, but are not required to begin there.

Area B occupies up to 101 character positions, beginning at margin B. All remaining entries begin in area B. On an interactive terminal, the user can reach margin B by typing horizontal-tab anywhere in area A (or in column 0). Area B is terminated by a line-feed, form-feed, or vertical tab usually preceded by a carriage return.

Chapter 3
The IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION is required in every source program and identifies the source program and the output from compilation. In addition, the user may include other documentary information such as the name of the program's author, the name of the installation, the dates on which the program was written and compiled, any special security restrictions, and any miscellaneous remarks.

3.1 GENERAL STRUCTURE

```
IDENTIFICATION DIVISION.
[PROGRAM-ID. [program-name] [comment paragraph].]
[AUTHOR. comment paragraph .]
[INSTALLATION. comment paragraph .]
[DATE-WRITTEN. comment paragraph .]
[DATE-COMPILED. comment paragraph. ]
[SECURITY. comment paragraph .]
[REMARKS. comment paragraph .]
```

3.2 TECHNICAL NOTES

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

b. The PROGRAM-ID paragraph contains the name identifying the program. The program-name may have up to six characters, and must contain only letters, digits, and the hyphen. It can be enclosed in quotation marks. The program-name cannot be a reserved word and must be unique. It cannot be the same as a section, paragraph, file, data or subprogram name.

This paragraph is optional. If it is not present, the name MAIN is assigned to the program.

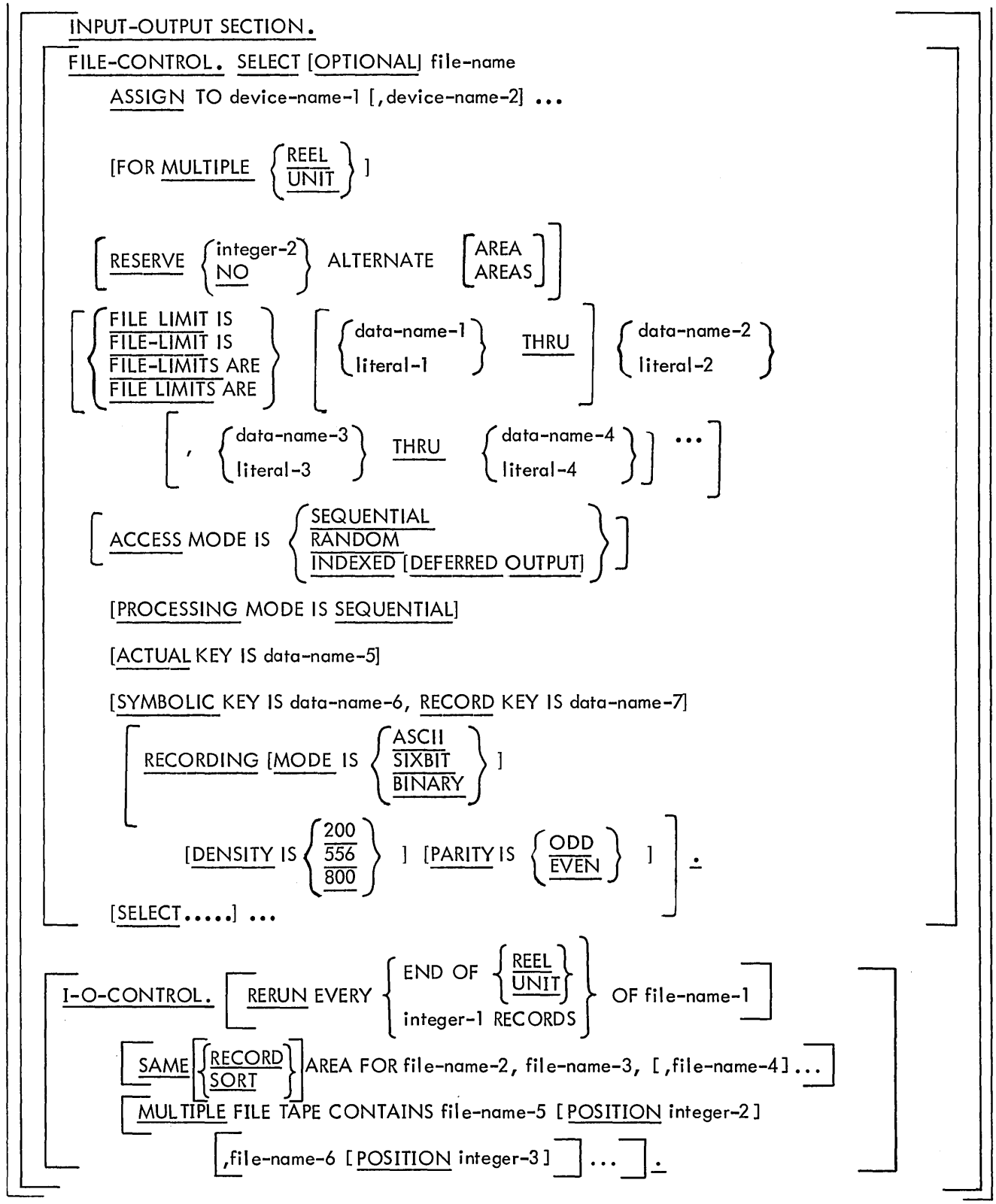
c. The remaining paragraphs are optional and, if used, may appear in any combination and in any order. A comments paragraph consists of any combination of characters from the COBOL character set organized to conform to COBOL sentence and paragraph format. All text appears as written on the output listing, except the DATE-COMPILED paragraph, which will be replaced by the current date. Reserved words can be used in any comment paragraph.

Chapter 4
The ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION allows the programmer to describe the particular computer configurations upon which the compilation and resulting object program are to be run, and to specify the hardware features of his computer configuration.

4.1 GENERAL STRUCTURE

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
[SOURCE-COMPUTER. [comment-paragraph].]
[OBJECT COMPUTER. { DECSYSTEM-10 }
                  { PDP-10       }
    [MEMORY SIZE integer-1 { CHARACTERS }
                           { WORDS      }
                           { MODULES    } ]
    [SEGMENT-LIMIT IS integer-2] .]
SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]
    [CHANNEL (m) IS mnemonic-name-2
        [,CHANNEL (n) IS mnemonic-name-3] ...]
    [SWITCH (m) { IS mnemonic-name-4 [ON STATUS IS condition-name-1]
                    [OFF STATUS IS condition-name-2]
                ON STATUS IS condition-name-1
                    [OFF STATUS IS condition-name-2]
                OFF STATUS IS condition-name-2
                    [ON STATUS IS condition-name-1] }
    [SWITCH (n) .....] ...]
        [literal-1 IS mnemonic-name-5]
        [CURRENCY SIGN IS literal-2]
        [DECIMAL-POINT IS COMMA] .
```


INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT [OPTIONAL] file-name
ASSIGN TO device-name-1 [,device-name-2] ...
[FOR MULTIPLE { REEL / UNIT }]
[RESERVE { integer-2 / NO } ALTERNATE [AREA / AREAS]]
[{ FILE LIMIT IS / FILE-LIMIT IS / FILE-LIMITS ARE / FILE LIMITS ARE } [{ data-name-1 / literal-1 } THRU { data-name-2 / literal-2 }]
[, { data-name-3 / literal-3 } THRU { data-name-4 / literal-4 }] ...]
[ACCESS MODE IS { SEQUENTIAL / RANDOM / INDEXED [DEFERRED OUTPUT] }]
[PROCESSING MODE IS SEQUENTIAL]
[ACTUAL KEY IS data-name-5]
[SYMBOLIC KEY IS data-name-6, RECORD KEY IS data-name-7]
[RECORDING [MODE IS { ASCII / SIXBIT / BINARY }]
[DENSITY IS { 200 / 556 / 800 }] [PARITY IS { ODD / EVEN }]].
[SELECT.....] ...
I-O-CONTROL. [RERUN EVERY { END OF { REEL / UNIT } / integer-1 RECORDS } OF file-name-1]
[SAME [{ RECORD / SORT }] AREA FOR file-name-2, file-name-3, [,file-name-4]...]
[MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-2]
[,file-name-6 [POSITION integer-3]] ...].

4.2 CONFIGURATION SECTION

The CONFIGURATION SECTION allows the user to describe the computer configuration on which he will run his object program. It also allows him to assign mnemonic names to certain hardware features.

```
CONFIGURATION SECTION.

[SOURCE-COMPUTER. [comment-paragraph] .]
[OBJECT-COMPUTER. {DECSYSTEM-10}
                  {PDP-10     }
      [MEMORY SIZE integer-1 {CHARACTERS}
                             {WORDS     }  ]
                             {MODULES   }
      [SEGMENT-LIMIT IS integer-2] .]

SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]
   [CHANNEL (m) IS mnemonic-name-2
        [,CHANNEL (n) IS mnemonic-name-3]...]
   [SWITCH (m) { IS mnemonic-name-4 [ON STATUS IS condition-name-1]
                    [OFF STATUS IS condition-name-2]
               { ON STATUS IS condition-name-1
                    [OFF STATUS IS condition-name-2]
               { OFF STATUS IS condition-name-2
                    [ON STATUS IS condition-name-1]
        [SWITCH (n) .....] ...]
   [literal-1 IS mnemonic-name-5]
   [CURRENCY SIGN IS [literal-2]]
   [DECIMAL-POINT IS COMMA] .
```

Technical Notes

a. This section is optional.

b. All commas and semicolons are optional. A period must terminate the entire entry in each of the three paragraphs.

SOURCE-COMPUTER

Function

The SOURCE-COMPUTER paragraph describes the computer on which the program is to be compiled.

General Format

```
[SOURCE-COMPUTER. [comment-paragraph].]
```

Technical Notes

a. This paragraph is optional.

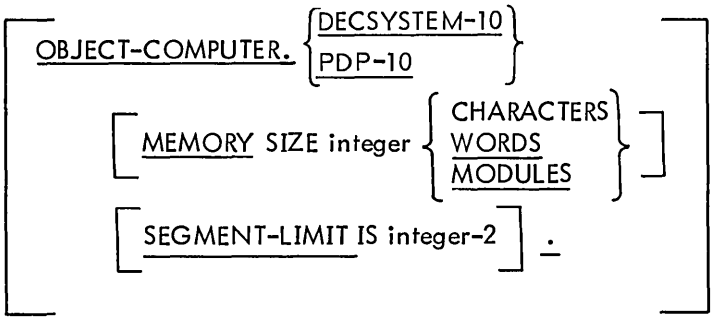
b. This paragraph is for documentation only. The comment paragraph is replaced in the listing by the word DECsystem-10.

OBJECT-COMPUTER

Function

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

General Format



Technical Notes

a. This paragraph is optional.

b. PDP-10 or DECsystem-10 must appear as the first entry following the paragraph-name OBJECT-COMPUTER.

c. The MEMORY SIZE clause is optional. If it is omitted, 262,144 WORDS are assumed. If it appears, the following ranges are applicable.

CHARACTERS	Up to 1,572,864 (262,144 words x 6 character/word)
WORDS	Up to 262,144
MODULES	Up to 256 (1 module equals 1024 words)

The MEMORY SIZE clause indicates the maximum amount of core that can be used by the run-unit. The only use of this clause is to determine the size of core to which the program can expand when sorting. If this clause is specified, the program will expand to the specified size when sorting. The sort buffer will be the specified memory size less the size of the program and LIBOL (the object-time system). If the MEMORY size clause is not specified, the sort will use a buffer whose size is equal to one-half of available core less the size of the run-unit.

d. If the SEGMENT-LIMIT clause is given, only those segments having priority numbers from 0 up to, but not including, the value of integer-2 are considered as resident segments of the program. Integer-2 must be a positive integer in the range 1 to 49.

If the SEGMENT-LIMIT clause is omitted, segments having priority numbers from 0 through 49 are considered as resident segments of the program (that is, SEGMENT-LIMIT IS 50 is assumed).

More on segmentation can be found in Chapter 6.

SPECIAL-NAMES

Function

The SPECIAL-NAMES paragraph provides a means of associating hardware devices with user-specified mnemonic names.

General Format

```
SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]
    [CHANNEL (m) IS mnemonic-name-2
        [CHANNEL (n) IS mnemonic-name-3]...]
[SWITCH (m)  { IS mnemonic-name-4 [ON STATUS IS condition-name-1]
               [OFF STATUS IS condition-name-2]
             { ON STATUS IS condition-name-1
               [OFF STATUS IS condition-name-2]
             { OFF STATUS IS condition-name-2
               [ON STATUS IS condition-name-1]
        [SWITCH (n).....] ..]
[literal-1 IS mnemonic-name-5]
[CURRENCY SIGN IS literal-2]
[DECIMAL-POINT IS COMMA] .
```

Technical Notes

a. This paragraph is optional.

b. The name CONSOLE refers to the user's Teletype console. The assigned mnemonic-name may be used with the ACCEPT and DISPLAY verbs in the PROCEDURE DIVISION to input data from and output data to the console.

c. The name CHANNEL refers to a channel on the line-printer control tape. m and n represent any integer from 1 to 8 and refer to any one of the eight channels on the tape. Control tape channels can be referred to in the ADVANCING clause of the WRITE verb in the PROCEDURE DIVISION to advance the paper form to the desired channel position. Refer to the DECsystem-10 System Reference Manual for a description of printer control tapes. For example, if the entry

CHANNEL (1) IS TOP-OF-PAGE

is included in this paragraph, the following procedure statement will print the line and then skip to the top of the next page.

IF LINE-COUNT IS GREATER THAN 50 WRITE PRINT-RECORD BEFORE ADVANCING TOP-OF-PAGE.

d. The name SWITCH refers to the hardware switches on the PDP-10 console. m and n represent any integer from 0 to 35 and refer to the corresponding console switches.

The mnemonic-name can be used in conditional expressions in the PROCEDURE DIVISION. For example, if the entry

SWITCH (4) IS INPUT-1

is included in this paragraph, the following condition is considered to be true if switch (4) is on.

IF INPUT-1 IS ON....

If a condition-name is specified for the ON or OFF STATUS of a switch, that condition-name can be used in a conditional expression. For example, if the entry

SWITCH (4) IS INPUT-1; OFF STATUS IS NO-INPUT

is included in this paragraph, the following procedure statements are functionally equivalent.

IF INPUT-1 IS OFF....

IF NO-INPUT....

e. The clause literal-1 IS mnemonic-name-5 specifies the CODE value for a particular report (refer to the CODE clause below). Literal-1 must be a nonnumeric literal enclosed in quotes, and can be from 1 through 120 characters in length.

f. The literal which appears in the CURRENCY SIGN clause must be used in PICTURE clauses (DATA DIVISION) to represent the currency symbol. If this clause is not present, only the standard $ may be used as a currency symbol in a PICTURE clause.

This literal is limited to a single printable character and must not be one of the following characters:

digits 0 through 9

alphabetic characters A, B, C, D, P, R, S, V, X, Z

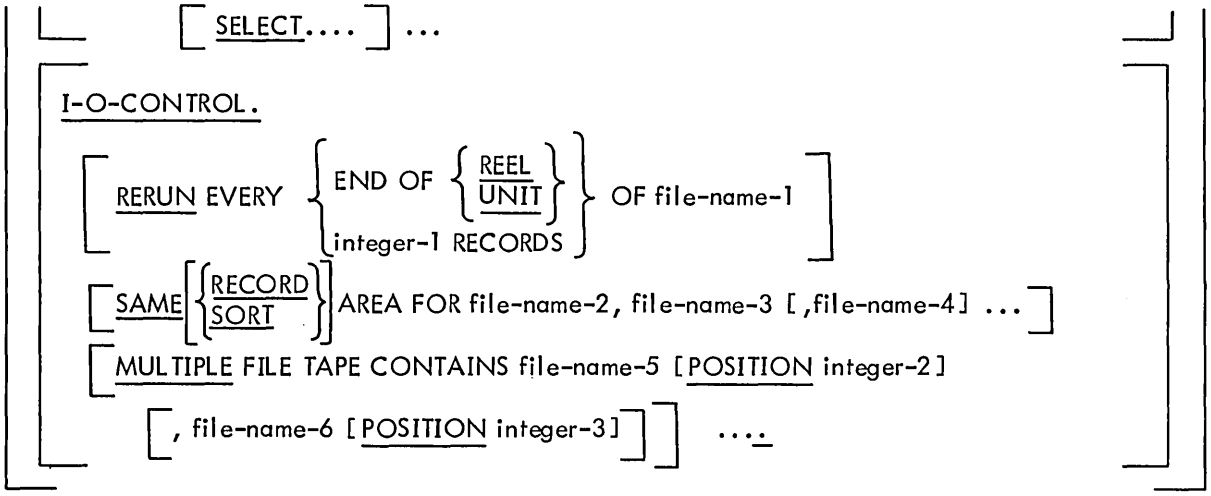
special characters * + - , . ; ( ) "

g. The clause DECIMAL-POINT IS COMMA, if present, causes the functions of the comma and the period to be interchanged in any PICTURE clause character-string and in any numeric literal.

4.3 INPUT-OUTPUT SECTION

The INPUT-OUTPUT SECTION names the files and external media required by the object program and provides information required for transmission and handling of data during execution of the object program.

```
INPUT-OUTPUT SECTION.

FILE-CONTROL. SELECT [OPTIONAL] file-name
    ASSIGN TO device-name-1 [,device-name-2]...
      [ FOR MULTIPLE { REEL } ]
                     { UNIT }
      [ RESERVE { integer-1 } ALTERNATE [ AREA  ] ]
                { NO        }           [ AREAS ]
    [ { FILE LIMIT IS    }
      { FILE-LIMIT IS    }
      { FILE-LIMITS ARE  }   [ { data-name-1 } THRU ] { data-name-2 }
      { FILE LIMITS ARE  }     { literal-1   }        { literal-2   }
        [ ,{ data-name-3 } THRU { data-name-4 } ] ... ]
           { literal-3   }      { literal-4   }
    [ ACCESS MODE IS { SEQUENTIAL                    } ]
                     { RANDOM                        }
                     { INDEXED [DEFERRED OUTPUT]     }
    [PROCESSING MODE IS SEQUENTIAL]
    [ACTUAL KEY IS data-name-5]
    [SYMBOLIC KEY IS data-name-6, RECORD KEY IS data-name-7]
    [ RECORDING [MODE IS { ASCII  } ]
                         { SIXBIT }
                         { BINARY }
      [ DENSITY IS { 200 } ]  [PARITY IS { ODD  } ]  ]  .
                   { 556 }               { EVEN }
                   { 800 }
```



Technical Notes

a. This section is optional.

b. All semicolons and commas are optional. Each SELECT statement in the FILE-CONTROL paragraph must end with a period. The entire entry in the I-O-CONTROL paragraph must end with a period.

FILE-CONTROL

Function

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows logical hardware assignments.

General Format

```
FILE-CONTROL. SELECT [OPTIONAL] file-name

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

    [FOR MULTIPLE { REEL }]
                  { UNIT }

    [RESERVE { integer-1 } ALTERNATE [AREA ]]
             { NO        }           [AREAS]

    [{ FILE LIMIT IS   }
     { FILE-LIMIT IS   }
     { FILE-LIMITS ARE }  [{ data-name-1 } THRU ]  { data-name-2 }
     { FILE LIMITS ARE }  { literal-1    }         { literal-2    }

        [, { data-name-3 } THRU { data-name-4 }]  ...]
           { literal-3   }      { literal-4   }

    [ACCESS MODE IS { SEQUENTIAL                    }]
                    { RANDOM                        }
                    { INDEXED [DEFERRED OUTPUT]     }

    [PROCESSING MODE IS SEQUENTIAL]

    [ACTUAL KEY IS data-name-5]

    [SYMBOLIC KEY IS data-name-6, RECORD KEY IS data-name-7]

        [RECORDING [MODE IS { ASCII  }  ]
                            { SIXBIT }
                            { BINARY }

          [DENSITY IS { 200 }  ] [PARITY IS { ODD  }]] .
                      { 556 }                { EVEN }
                      { 800 }

    [SELECT ....] ...
```

Technical Notes

a. This section is optional.

b. All semicolons and commas are optional. Each SELECT clause must end with a period.

c. The SELECT and ASSIGN clauses must appear before any other clause shown. The other clauses can be in any order.

d. The individual clauses are described on the following pages in the order shown above.

SELECT

Function

The SELECT statement names each file that is to be described in the DATA DIVISION, and assigns each file to a particular device.

General Format

SELECT [OPTIONAL] file-name ASSIGN TO device-name-1 [, device-name-2]...

Technical Notes

a. Each file described in the DATA DIVISION must be named once and only once as a file-name in a SELECT statement. Conversely, each file named in a SELECT statement must have a File Description entry in the DATA DIVISION. Each file-name must be unique within a program.

b. The key word OPTIONAL is required for input files that are not necessarily present each time the object program is run. When an OPEN statement is executed for a file that has been declared OPTIONAL, the question IS file-name PRESENT? is typed and the operator responds with YES or NO. If the response is YES, the file is processed normally; if the response is NO, the first READ statement executed for that file will immediately take the AT END or INVALID KEY path.

c. The ASSIGN clause specifies the file medium. Device-names can be either physical device-names or logical device-names.

Physical device-names are fixed mnemonic-names that are associated with specific peripheral devices. When specified in an ASSIGN clause, a physical device-name assigns the associated file to that device. Physical device-names are described in the DECsystem-10 Operating System Commands manual, which appears in the DECsystem-10 Software Notebooks.

Logical device-names are names created by the programmer. They can contain up to six characters, consisting of any combination of letters and digits. At object execution time, each logical device-name must be assigned to a physical device by means of the monitor ASSIGN command (refer to the DECsystem-10 Operating System Commands manual, DEC-10-MRDC-D, for details of the ASSIGN command).

d. More than one device can be assigned to a file to avoid delay when switching from one reel or unit to the next. When more than one device is specified, the object program automatically uses the next device, in a cyclic manner, when an end-of-reel condition is detected.

e. At least three, but no more than six, devices must be assigned to a sort file. These devices must be retrievable, i.e., disk, DECtape, or magnetic tape.

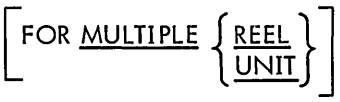
f. If the access mode is INDEXED, and two devices are assigned, the first device contains the index portion of the file and the second contains the data portion of the file.

FOR MULTIPLE

Function

The FOR MULTIPLE clause specifies that a file occupies more reels than the number of devices assigned.

General Format



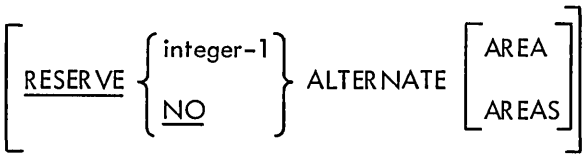
Technical Notes

a. Whether or not multiple devices are assigned, the FOR MULTIPLE clause must be included for any file that occupies (or might occupy) more reels than the number of devices assigned.

Function

The RESERVE clause allows the user to specify the number of input-output buffer areas to be allocated by the compiler to this file.

General Format



Technical Notes

a. If the access mode is RANDOM or INDEXED, this clause is ignored and only one buffer area is assigned.

b. If the NO option is used, only two buffer areas will be allocated.

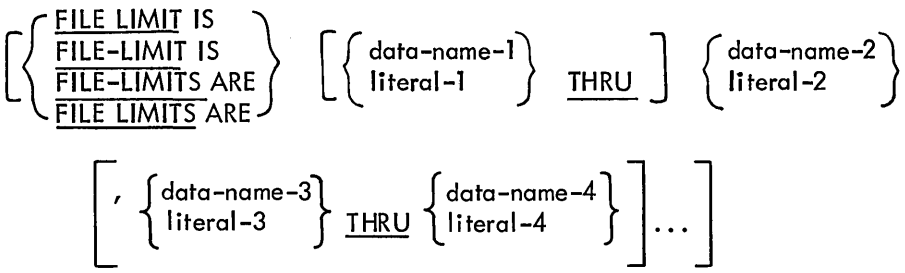
c. If the integer-1 option is used, the integer specifies the number of buffer areas to be assigned in addition to the two areas always assigned by the compiler. Integer-1 must be positive.

FILE-LIMIT

Function

The FILE-LIMIT clause is used to define the logical limits of a file whose access mode is RANDOM.

General Format



Technical Notes

a. The FILE-LIMIT clause is required only for files whose access mode is RANDOM; it is optional for files with SEQUENTIAL access mode residing on mass-storage devices. It is ignored in all other cases.

b. The words FILE and LIMIT (or LIMITS) can be separated by space or hyphen.

c. Every data-name used in this clause must be defined as USAGE COMP and must be integers of 10 digits or less.

d. Each pair of operands represents a logical portion of the file. If the first operand of the first pair is not specified, it assumed to be 1.

e. The operands represent logical record numbers relative to the beginning of the file.

f. The logical beginning of a random-access file is considered to be that record represented by the first operand of the FILE-LIMIT clause. The logical end of a random-access file is considered to be that record represented by the last operand.

g. The value of data items specified in this clause is utilized by the object-time operating system only when the file is opened by an OPEN statement.

h. If a file whose access mode is RANDOM is processed sequentially, the FILE-LIMIT clause is ignored. Thus, records with keys higher than the upper FILE-LIMIT can be created.

ACCESS MODE

Function

The ACCESS MODE clause specifies the way in which a file will be accessed.

General Format

```
[ ACCESS MODE IS { SEQUENTIAL                      } ]
                 { RANDOM                          }
                 { INDEXED [DEFERRED OUTPUT]       }
```

Technical Notes

a. The ACCESS MODE clause is required for random-access and indexed-sequential files. It is ignored for sequential files.

b. If ACCESS MODE IS SEQUENTIAL and the file is on a random-access device, the random-access records are obtained or placed sequentially. That is, the next logical record is made available from the file on a READ statement execution, and an output record is placed into the next available area on a WRITE statement execution. Thus sequential access processing on a random-access device is functionally similar to the processing of a magnetic tape file.

c. If ACCESS MODE IS RANDOM, the contents of the data item associated with the ACTUAL KEY specifies which record, relative to the beginning of the file, is made available by a READ statement, or where the record is to be placed by a WRITE statement.

d. If ACCESS MODE IS INDEXED, the contents of the data item associated with the SYMBOLIC KEY specifies which record is made available by a READ statement, or where the record is to be placed by a WRITE statement, or which record is to be deleted by a DELETE statement, or which record will be replaced by a REWRITE statement.

e. The DEFERRED OUTPUT option of the INDEXED ACCESS MODE causes the operating system to output a block of an indexed sequential file only when another block must be brought into core. Normally, to ensure security for the file, a block is output every time a record is written, even if records are written successively in the same block.

PROCESSING MODE

Function

The PROCESSING MODE clause specifies that the file is to be processed sequentially.

General Format

[PROCESSING MODE IS SEQUENTIAL]

Technical Notes

a. This clause is for documentation only; records are always processed in the order in which they are accessed.

ACTUAL KEY

Function

The ACTUAL KEY clause specifies which record is read or written in a random-access file.

General Format

[ACTUAL KEY IS data-name]

Technical Notes

a. The ACTUAL KEY clause is valid only for files whose access mode is RANDOM; it must be specified for those files. This clause cannot be used for files whose access modes are INDEXED or SEQUENTIAL.

b. The ACTUAL KEY data-name must be defined in the DATA DIVISION as a COMPUTATIONAL item of ten or fewer digits. The PICTURE can contain only the characters S and 9.

SYMBOLIC KEY

Function

The SYMBOLIC KEY clause specifies the record in an indexed-sequential file that is read, written, deleted, or rewritten.

General Format

```
[SYMBOLIC KEY IS data-name-1, RECORD KEY IS data-name-2]
```

Technical Notes

a. The SYMBOLIC KEY clause is valid only for files whose access mode is INDEXED; it must be specified for those files (refer to the READ statement in the PROCEDURE DIVISION).

b. The SYMBOLIC KEY data-item must be defined in the DATA DIVISION, and must not appear in the record area of the file to which it pertains. It must agree with the description of the RECORD KEY data item in class, usage, size, and number of decimal places.

c. The RECORD KEY data item must be defined as an item in the record area of the file to which it pertains. Though the RECORD KEY is described in only one of the records, it is assumed to occupy the same position in all records for that file.

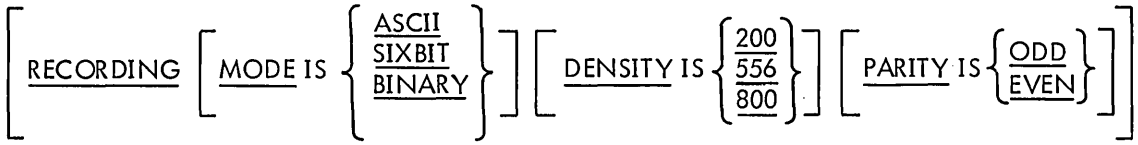
d. The RECORD KEY is required to describe the location in the record area of the key for the file. The contents of the RECORD KEY data item must be unique for each record in the file, and cannot be equal to LOW-VALUES (refer to the READ, WRITE, REWRITE, and DELETE statements in the PROCEDURE DIVISION).

RECORDING MODE/DENSITY/PARITY

Function

The RECORDING clause specifies the mode in which a file is recorded, and/or the density at which a magnetic tape is recorded, and/or the parity at which a magnetic tape is recorded.

General Format



Technical Notes

a. The RECORDING MODE clause allows the user to record data on the device in a format other than that used in core. For further explanation of recording modes, refer to Chapter 8.

b. The DENSITY and PARITY clauses are valid only for magnetic tape, and are ignored for all other devices. If the DENSITY clause is not present, tapes are recorded in the density standard for the installation. The density at an installation can be modified by the .SET DENSITY command, which is described in the DECsystem-10 Operating System Commands manual. If the PARITY clause is not present, ODD is assumed.

I-O-CONTROL

Function

The I-O-CONTROL paragraph specifies the points at which a rerun dump is to be performed, the memory area which is to be shared by different files, and the location of files on a multiple-file reel.

General Format

```
[ I-O-CONTROL.
  [ RERUN EVERY { END OF { REEL } } OF file-name-1 ]
                { { UNIT }     }
                { Integer-1 RECORDS }
  [ SAME [ { RECORD } ] AREA FOR file-name-2, filename-3 [, file-name-4] ... ]
         [ { SORT   } ]
  [ MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-2]
      [, file-name-6 [POSITION integer-3]] ... ] ... . ]
```

Technical Notes

a. This paragraph is optional.

b. The RERUN clause specifies when a rerun dump is to be performed.

The dump is always written onto a disk file, using the program's low segment name as the filename, and an extension of CKP. If the program has no filename because it was never SAVEd, the program name (from the PROGRAM-ID paragraph in the IDENTIFICATION DIVISION) is used as a filename, with the extension CKP.

If the END OF { UNIT } option is used, a rerun dump is taken at the end of each input or output reel of the specified file.
{ REEL }

If the integer-1 RECORDS option is used, a rerun dump is taken whenever a number of logical records equal to a multiple of integer-1 is either read or written for the file.

A rerun dump is not taken if any files are open for input/output (updating), or if any file is open on a device other than magnetic tape, disk, or teletype or if an indexed sequential file is open.

c. The SAME AREA clause specifies that two or more files are to use the same area during processing; this includes all buffer areas and the record area.

If the RECORD option is specified, the files share only the record area (i.e., the area in which the current logical record is processed). If the RECORD option is not used, only one of the named files can be open at one time.

The SORT option is used for sort files. However, this option need not be specified because all sort files always use the same sort area.

d. The MULTIPLE FILE clause is required when more than one file shares the same physical reel of tape. This clause is invalid for media other than magnetic tape.

Regardless of the number of files on a single reel, only those files defined in the program may be listed. If all files residing on the tape are listed in consecutive order, the POSITION option need not be given. If any file on the tape is not listed, the POSITION option must be included; integer-2, integer-3, etc., specify position of the file relative to the beginning of the tape.

All files on the same reel of tape must be ASSIGNED to the same device in the FILE-CONTROL paragraph.

Not more than one file on the same reel of tape can be open at one time.

Chapter 5
The DATA DIVISION

The DATA DIVISION, required in every COBOL program, describes the characteristics of the data to be processed by the object program.

This data can be divided into five major types:

a. Data contained in files, both input and output.

b. Data initially stored as part of the program (e.g., constant data such as messages, tables of fixed values, and the like), or data developed during processing (e.g., intermediate information such as partial arithmetic results).

c. Data in a subprogram that is passed from the program calling it.

d. Data used in a communications environment.

e. Data to be printed in a report, and the format used to print such data.

To handle these five types of data, the DATA DIVISION consists of four sections:

a. The FILE SECTION, which describes the characteristics and the data formats for each file processed by the object program.

b. The WORKING-STORAGE SECTION, which contains any fixed values and the working areas in which intermediate data can be stored.

c. The LINKAGE SECTION, which describes the data in a subprogram that is available through a calling program.

d. The COMMUNICATION SECTION, which describes the data items used to store special information about communications data and the communications network.

e. The REPORT SECTION, which describes the data and the format of a report.

5.1 FILE SECTION

In the FILE SECTION, the characteristics of each file to be processed are described by two types of entries.

The first type of entry, the file description, describes the physical aspects of the file. These aspects include:

a. How the logical data records of the file are physically grouped into blocks on the file medium.

b. The maximum length of a logical record, which cannot exceed 4095 characters.

c. Whether or not the file contains header and trailer labels and, if so, whether the format of these labels is standard or nonstandard.

d. The names of the records contained in the file.

e. The names of any reports in the file.

The second type of entry, the data description, describes the data formats of the logical records in the files.

The FILE SECTION begins with the section-header FILE SECTION. If present, it must be the first section in the DATA DIVISION.

5.2 WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is used to define (1) data (such as constant values and messages) that is to be initially stored when the object program is loaded, and (2) areas that are to be used for storing intermediate results. The WORKING-STORAGE SECTION is similar to the FILE SECTION, except that the WORKING-STORAGE SECTION cannot contain FD, SD, or RD entries and it can contain level-77 items.

The WORKING-STORAGE SECTION begins with the section-header WORKING-STORAGE SECTION.

5.3 LINKAGE SECTION

The LINKAGE SECTION is used in a subprogram to describe data that is available through a calling program. The LINKAGE SECTION can appear only in a subprogram, and can be placed anywhere in the DATA DIVISION after the FILE SECTION. The structure is the same as that of the WORKING-STORAGE SECTION with certain restrictions. These restrictions are:

1. The VALUE clause can only be used in condition-name (level-88) entries.
2. The VALUE OF IDENTIFICATION (or ID), the VALUE OF DATE-WRITTEN, and the VALUE OF USER NUMBER cannot appear in this section.
3. The OCCURS clause with the DEPENDING phrase cannot appear in this section.
4. The SYMBOLIC KEY and ACTUAL KEY data items cannot be defined in this section.
5. The data items in the FILE-LIMITS clause cannot be defined in this section.
6. Subscripted items cannot appear in this section.

No storage is allocated in a subprogram for the data described in the LINKAGE SECTION. Instead, at run-time, the PROCEDURE DIVISION references to items in the LINKAGE SECTION are

resolved by equating the data items in the called program to the data items in the calling program. The compiler finds the references to equate in the USING clause of the CALL statement in the calling program and in the USING clause of the ENTRY statement or PROCEDURE DIVISION header of the called program. The identifiers in the USING clause of the CALL statement refer to entries in the FILE, WORKING-STORAGE, or LINKAGE SECTION in the calling program. The identifiers in the USING clause of the ENTRY statement or the PROCEDURE DIVISION header refer to entries in the LINKAGE SECTION in the called program. The identifiers are equated by COBOL according to their positions in the USING clauses. Thus:

```
CALLING PROGRAM                    CALLED PROGRAM
      :                                  :
DATA DIVISION.                     DATA DIVISION.
FILE SECTION.                      FILE SECTION.
FD...                              LINKAGE SECTION.
01 MAIN...                         01 SUB...
02 MAIN1...                        02 SUB1...
02 MAIN2...                        02 SUB2...
      :                                  :
PROCEDURE DIVISION.                PROCEDURE DIVISION.
      :                            ENTRY ENTRPT USING SUB.
CALL ENTRPT USING MAIN.                  :
      :                            EXIT PROGRAM.
```

The identifier MAIN is defined in the FILE SECTION of the calling program and the identifier SUB is defined in the LINKAGE SECTION of the called program. At run-time, they are equated to each other by the compiler so that the identifier SUB can use the location of MAIN and the data contained therein. Refer to Section 8.11 for more information on subprograms.

Each 01- or 77-level item in the LINKAGE SECTION must have a unique name because it cannot be qualified. Also, each 01- and 77-level item must correspond to a word-aligned item of the same size or larger in the calling program. Word-aligned items start at the beginning of a computer word. All 01- and 77-level items fulfill this requirement; items that do not, can be made to do so by means of the SYNCHRONIZED LEFT statement.

5.4 COMMUNICATION SECTION

The COMMUNICATION SECTION is the section where data items that establish a link between a COBOL Message Processing Program (MPP) and the Message Control System (MCS-10) are defined. These data items are used whenever the MPP interacts with MCS-10. Within the COMMUNICATION SECTION, the user defines communication-description entries (called CD-entries) for input communication functions and output communication functions.

A CD-entry causes a data record to be defined. This record, called a CD-record, contains all of the special data items required by that CD-entry. These special data items are called CD-items. The formats and rules for input and output CD-entries are described in the MCS-10 Programmer's Procedures Manual (DEC-10-CMCPA-A-D) along with a description of MCS-10 and the procedures for writing Message Processing Programs.

The COMMUNICATION SECTION begins with the section-header COMMUNICATION SECTION. If present, it must appear after the FILE SECTION and before the REPORT SECTION.

5.5 REPORT SECTION

The REPORT SECTION provides the facility for producing reports by allowing the programmer to specify the physical appearance of a report rather than requiring him to specify the detailed procedures necessary to produce the report. A report consists of data presented in a particular format.

The data for a report can be read from a file or another part of the program or can be summed within the REPORT SECTION. The format of the report is given in the record description and report group entries in the REPORT SECTION.

The REPORT SECTION begins with the section-header REPORT SECTION, and must follow the FILE SECTION, the WORKING-STORAGE SECTION and the LINKAGE SECTION.

5.6 DATA DESCRIPTIONS

5.6.1 Elementary Items and Group Items

The basic user-defined datum in a COBOL program is called an elementary item; it may be referenced directly only as a unit. An elementary item may be associated with contiguous elementary items to form sets of data items called group items. Group items may be associated with other group items and/or elementary items to form more inclusive group items. Thus, an elementary item may be contained within one or more group items, and a group item may contain more than one elementary item.

5.6.2 Level Numbers

Level numbers indicate a hierarchy in which data items are ranged. The highest level is 01, which signifies that the data item is a record within a file named in an FD clause (or is a contiguous area in the WORKING-STORAGE SECTION). Level numbers of 02 through 49 indicate items that are subordinate to a 01-level data item. For example, an employee record can be described in the following manner.

```
01 EMPLOYEE-RECORD
     02 NAME
        03 FIRST-NAME  PICTURE IS A(6)
        03 MIDDLE-INITIAL  PICTURE IS A
        03 LAST-NAME  PICTURE IS A(20)
     02 BADGE-NUMBER  PICTURE IS X(5)
     02 SALARY-CLASS  PICTURE IS X(2)
```

Within a record description, the level numbers indicate which items are contained within higher-level items. That is, in the above example, the items that have a 03 level are subordinate to NAME, which has a 02 level, which is in turn subordinate to EMPLOYEE-RECORD, which has a 01 level. The example also shows elementary items (those that contain PICTURE clauses) contained within group items. In this example, EMPLOYEE-RECORD is a group item, NAME is a group item contained within a group item, and FIRST-NAME is an elementary item contained within the group item NAME. An item at a 01 level can be an elementary item as well as a group item as long as it is referenced as a unit. For example:

```
01 EMPLOYEE-RECORD  PICTURE IS A(34)
```

shows the same record as above, but in this case the record is always operated on as a single entity.

Three other level numbers are available to the COBOL programmer: 77, 66, and 88.

Items with a level number of 77 are noncontiguous elementary items that are written only in the WORKING-STORAGE SECTION to define constant values and to store intermediate results.

Level-66 data items are those items that contain an explicitly specified portion of a record, or even the whole record. A data item at a level of 66 is used in a RENAMES clause to regroup items within a record. After a record is described, a level-66 data item RENAMES a portion of that record. The level-66 data item can be a regrouping of the whole record, a group within the record, or a combination of group and elementary items. For example:

```
01 EMPLOYEE-RECORD
     02 NAME
        03 FIRST-NAME...
        03 MIDDLE-INITIAL...
        03 LAST-NAME...
     02 BADGE-NO....
     02 SALARY-CLASS...
     66 PERSONNEL-REC RENAMES NAME THRU BADGE-NO
     66 PAY-REC RENAMES LAST-NAME THRU SALARY-CLASS
```

When the level-66 item PAY-REC is referenced, the items LAST-NAME, BADGE-NO., and SALARY-CLASS are referenced as a unit. The programmer can thus regroup portions of a record for differing purposes.

Level-88 items are condition-names that cause a value or a range of values to be assigned to a data item. The data item that is assigned the value is then considered a conditional variable. The condition name (i.e., the level-88 item) is used in a conditional expression in the PROCEDURE DIVISION instead of the conditional variable. For example:

```
03 BADGE-NO....
     88 FIRST-BADGE VALUE IS A0001
     88 LAST-BADGE VALUE IS Z9999
```

In a comparison, the following statements would then be equivalent:

Conditional Variable	Condition-Name
IF BADGE-NO. IS EQUAL TO A0001...	IF FIRST-BADGE...
IF BADGE-NO. IS EQUAL TO Z9999...	IF LAST-BADGE...

5.6.3 Records and Files

Records can be divided into two categories: those associated with a file and those not associated with a file. A file is the highest level of data organization in COBOL; and in DECsystem-10 COBOL, a file represents a collection of data records held on some external medium, i.e., not wholly in real or virtual core. Records not associated with a file are those values stored in the WORKING-STORAGE and LINKAGE SECTIONS or sum counters in the REPORT SECTION.

5.7 QUALIFICATION

Any data item that is to be referenced must be uniquely identified. This unique identification can be achieved by the assignment of a unique name to each item. However, in many applications this is tedious and inconvenient (1) because of the large number of names required, and (2) because items containing the same type of information in different records would have different names. Therefore, qualification is introduced to allow similar items and certain records to have identical names.

Qualification means giving enough information about the item to specify it uniquely. In COBOL, this information is the name of the group items containing it, in order of increasing inclusiveness. It is not necessary to name each group containing it, but only enough groups so that no other item with the same name as the original item could be identically qualified. It is also unnecessary to name each successively higher group containing the item until a unique qualification is made. Any set of names that uniquely describe the item can be used.

Example:

```
01      RECORD-1.          01      RECORD-2.
  02      ITEM-1.            02      ITEM-2.
    03      SUB-ITEM.          03      SUB-ITEM.
      04      FIELD PIC X.       04      FIELD PIC X.
```

FIELD in the left-hand example can be referenced uniquely in any of the following ways:

```
FIELD OF SUB-ITEM OF ITEM-1 OF RECORD-1.
FIELD OF SUB-ITEM OF ITEM-1.
FIELD OF SUB-ITEM IN RECORD-1.
FIELD IN ITEM-1 OF RECORD-1.
FIELD IN RECORD-1.
FIELD IN ITEM-1.
```

Since the connectives OF and IN are equivalent, they may be used interchangeably.

The only data items which need have unique names are level-77 items and records not associated with files, since they are not contained in any higher level data structure. Records associated with files may be qualified by the file name, as may any item contained within the record. File names must be unique.

Level-66 items may be qualified only (1) by the name of the record with which they are associated and (2) by the name of any file with which that record is associated.

5.8 SUBSCRIPTING AND INDEXING

In some applications, it is convenient for the programmer to specify a set of data values as a table rather than assign a name to each element of the set. A table (or array) is a set of homogeneous items stored together in core for use by the program. The table elements are defined in the program by an OCCURS clause appearing in the description of a data item. The data item thus defined represents not one item but a set of items having the identical format. To refer to one of the elements of the set, subscripting or indexing is used. In DECsystem-10 COBOL, subscripting and indexing are identical in use and can be used interchangeably. However, the manner in which they are defined differs. Subscripting is defined simply by the fact that an item has an OCCURS clause in its description. For example,

```
01  RATE-TABLE
    02 AREA OCCURS 25 TIMES
```

describes AREA as 25 elements of RATE-TABLE. To refer to an element of RATE-TABLE, the user must subscript it, i.e., AREA (10) is the tenth element (or occurrence) or AREA. A subscript can be a data-name to which an integer value has been assigned as well as an integer. Thus, AREA (DIST), when DIST has been assigned the value 10, is the same as AREA (10).

To specify indexing, the user must add the INDEXED BY option to the OCCURS clause. Thus,

```
01  RATE-TABLE
    02 AREA OCCURS 25 TIMES INDEXED BY IND
```

defines AREA as 25 elements of the table and defines IND as the index by which each element of the table can be indexed, i.e., AREA (IND) is an element in the table. The index-name IND is treated

exactly like the data-name DIST because the compiler recognizes an index-name as being exactly the same as a data-name. An item defined as an index in an OCCURS clause has an implicit USAGE of INDEX, and is equivalent to a data item that is declared USAGE INDEX. However, this USAGE is included in DECsystem-10 COBOL for compatibility with other compilers because an item whose USAGE is INDEX (implicit or explicit) is treated as if its USAGE were COMPUTATIONAL. This is because the PDP-10 does not have special index registers to handle index-data-items. The registers in the PDP-10 are not dedicated to indexes; all computation is done in these registers. In fact, a data-name that is used as a subscript can be explicitly declared as USAGE INDEX. It will be treated as a COMPUTATIONAL data item by the compiler.

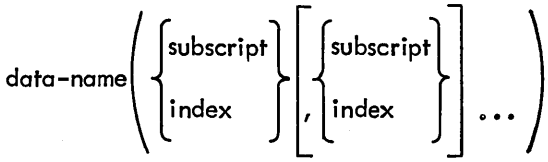
In COBOL, tables can be 1-dimensional (requiring a single subscript/index for referring to individual items) or 2-dimensional or 3-dimensional (requiring two or three subscripts/indexes, respectively). For example,

C (1, 3)

represents the item located in the first row and third column of a 2-dimensional table defined by the DATA DIVISION entries

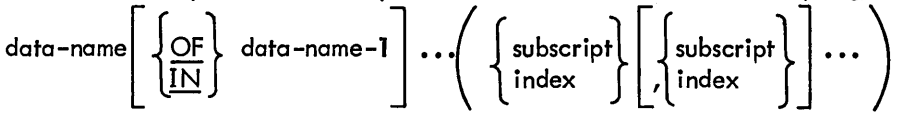
```
01  TABLEA
    02  LEVEL1 OCCURS 20 TIMES
        03  C  OCCURS 5 TIMES
```

Two forms of subscripting/indexing are permissible in DECsystem-10 COBOL - direct and relative. Direct subscripting/indexing means that the subscript/index refers directly to the desired element. The subscript/index must be enclosed in parentheses and must appear immediately after the terminal space that follows the data-name. When referring to elements in multi-dimensional tables, subscripts/indexes are written from left to right in the order of major (subscript/index varying least rapidly), intermediate, and in the case of a 3-dimensional table, minor (subscript/index varying most rapidly). Subscripts/indexes are separated by a comma followed by a space. No spaces can appear immediately following the left parenthesis or immediately preceding the right parenthesis. The form for direct subscripting/indexing is:

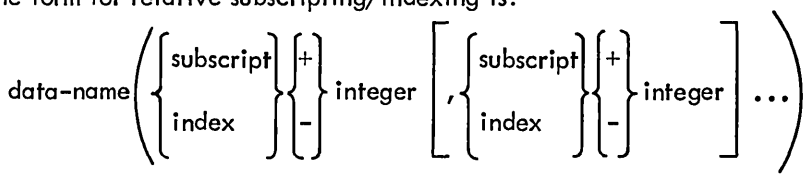


Thus in a table having a major element occuring 10 times, an intermediate element occuring 5 times within each occurence of the major element, and a minor element occurring 3 times within each intermediate element, the last major element of the table is referred to by the subscript form (10, 5, 3).

When a table element must be qualified for uniqueness, the format for direct subscripting/indexing is:



Relative subscripting/indexing means that the element of the table is referred to indirectly by a subscript/index to which an integer is added or subtracted. Relative subscripting/indexing is specified when the subscript/index is followed by the operator plus (+) or minus (-) followed by an unsigned integer numeric literal - all enclosed in the parentheses immediately following the terminal space of the data-name. The form for relative subscripting/indexing is:



When relative subscripting/indexing is used, the element of the table that is referred to is not the one to which the subscript/index refers, but the element to which the subscript/index plus or minus the integer refers. That is, if the item

AREA (IND + 2)

is specified, and IND is set at 3, the fifth occurrence of AREA is referred to, not the third. However, the value of the subscript/index is not changed by relative subscripting/indexing; the value of IND remains 3.

File Description (FD)

Function

The File Description (FD) 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-3 TO] integer-4 CHARACTERS ]
   [ LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED / record-name-1 [,record-name-2] ... } ]
   [ { REPORT IS / REPORTS ARE } report-name-1 [,report-name-2 ...] ]
   [ VALUE OF [ { ID / IDENTIFICATION } IS { data-name-1 / literal-1 } ]
                [ DATE-WRITTEN IS { data-name-2 / literal-2 } ]
                [ USER-NUMBER IS { data-name-3 / literal-3, literal-4 } ] ]
   [ DATA { RECORD IS / RECORDS ARE } record-name-3 [,record-name-4] ... ].
```

Technical Notes

a. An FD entry must be present for each file-name selected in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

b. All semicolons and commas are optional. The entire FD entry must terminate with a period.

c. The clauses may appear in any order within the File Description entry.

d. Each of the above clauses appears in alphabetical order on the following pages.

BLOCK CONTAINS

Function

The BLOCK CONTAINS clause specifies the size of a logical block.

General Format

```
[ BLOCK CONTAINS [integer-1 TO] integer-2 {RECORD(S)  } ]
                                          {CHARACTERS }
```

Technical Notes

a. If this clause is not present or if integer-2 is 0, the file is not organized into logical blocks. Rather, all records are placed in the file with no empty space. The file is then considered to be "unblocked" or "blocked zero".

b. If the CHARACTERS option is used, the logical block size is specified in terms of the number of character positions required to contain the record. If the recording mode is ASCII (that is, all records for the file are described, explicitly or implicitly, as USAGE DISPLAY-7), it is assumed that the size is specified in terms of DISPLAY-7 characters. If the recording mode is SIXBIT (that is, the records for the file are all described as other than USAGE DISPLAY-7), it is assumed that the size is specified in terms of DISPLAY-6 characters. See "USAGE" for a discussion of the USAGE of group items, including records.

c. Integer-1 and integer-2 must be positive integers. If only integer-2 is specified, it represents the exact size of the logical block. If both integer-1 and integer-2 are given, integer-1 is ignored and integer-2 is used as the blocking factor.

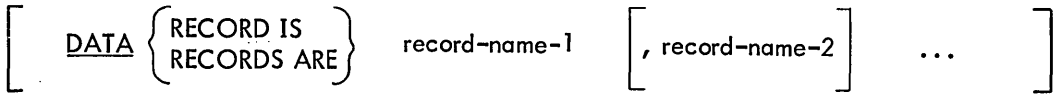
d. Files whose access modes are RANDOM or INDEXED must have a nonzero blocking factor. Also, RANDOM files with a recording mode of ASCII must have a blocking factor of 1.

DATA RECORD

Function

The DATA RECORD clause cross references the record descriptions with their associated file.

General Format



Technical Notes

a. This clause is optional because all records not associated with a LABEL RECORDS clause are assumed to be data records.

b. Both record-name-1 and record-name-2 must be the names given in 01-level data entries subordinate to this FD. The presence of more than one such record-name indicates that the file contains more than one type of data record. These records may have different descriptions. The order in which they are listed is not significant.

c. All records within a file share the same area.

FD file-name

Function

The FD file-name clause identifies the file to which this file description entry and the subsequent record descriptions relate.

General Format

FD file-name

Technical Notes

a. This entry must begin each file description.

b. The file-name must appear in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

LABEL RECORD

Function

The LABEL RECORD clause specifies whether or not labels are present on the file and, if so, identifies the format of the labels.

General Format

```
[ LABEL { RECORD IS  } { OMITTED                                      } ]
        { RECORDS ARE} { STANDARD                                     }
                       { record-name-1[, record-name-2] . . .         }
```

Technical Notes

a. If the clause is omitted, LABEL RECORDS ARE STANDARD is assumed.

b. The OMITTED option is used when the file has no header or trailer labels.

c. The STANDARD option is used when the file has header and trailer labels that conform to the DECsystem-10 standard format (see Chapter 8). If this clause is used for files on disk or DECtape, LABEL RECORDS ARE STANDARD must be specified. See VALUE OF IDENTIFICATION clause for the association between the label and the filename on disk or DECtape.

d. The record-name option is used when the file labels do not conform to the DECsystem-10 standard format. The record-names must appear as the names of record descriptions (level-01) subordinate to this FD; the record-names must not appear in a DATA RECORDS clause.

RECORD CONTAINS

Function

The RECORD CONTAINS clause specifies the size of the data records in this file.

General Format

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

Technical Notes

a. Because the size of each data record is completely defined by its record description entry, this clause is for documentation purposes only and is never required. However, if it is used, the following rules must be observed.

b. Integer-1 and integer-2 must be positive integers. Integer-2 may not be less than the size of the largest record but cannot exceed 4095, which is the limit on the size of a record. The limit on the size of ASCII (DISPLAY-7) records in a sort file is approximately 3400 characters.

c. The data record size is specified in terms of the number of character positions required to contain the record.

REPORT

Function

The REPORT clause specifies the name of each report that is associated with the file.

General Format

```
[ { REPORT IS  } report-name-1 [ ,report-name-2]... ]
  { REPORTS ARE}
```

Technical Notes

a. This clause is optional; it is used only when Report-Writer statements cause output to be written on the file.

b. Report-name-1 and report-name-2 must be the names of Report Descriptor items in the REPORT SECTION.

c. If this clause is used, the data record description can be omitted because the name of the data record is not referred to directly in the PROCEDURE DIVISION. When the data record description is omitted, the compiler automatically assumes a 132-character record.

SD file-name

Function

The SD file-name clause identifies the sort file to which this file description entry and the subsequent record descriptions relate.

General Format

```
SD File-name [ DATA { RECORD IS } record-name-1 [, record-name-2] ... ]
             { RECORDS ARE }
[ RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS ] .
```

Technical Notes

a. This entry must begin each sort file description.

b. The file-name must appear in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

c. The DATA RECORD and RECORD CONTAINS clauses are the only descriptive clauses allowed.

VALUE OF IDENTIFICATION/DATE-WRITTEN/USER-NUMBER

Function

The VALUE OF clause provides specific data for an item within the label records associated with a file.

General Format

```
[ VALUE OF [ {ID / IDENTIFICATION} IS {data-name-1 / literal-1} ]
           [ DATE-WRITTEN IS {data-name-2 / literal-2} ]
           [ USER-NUMBER IS {data-name-3 / literal-3, literal-4} ] ]
```

Technical Notes

a. The VALUE OF IDENTIFICATION clause is required only if label records are standard; it is ignored in all other cases. The VALUE OF DATE-WRITTEN is always optional.

b. The three clauses can be written in any order, but only one of each can be specified for a file.

c. IDENTIFICATION represents the file-name and file-name extension of a file with standard labels. If a data-name is specified, it must be associated with a DISPLAY-6 or DISPLAY-7 data item nine characters in length. If a literal is specified, it must be a nonnumeric literal nine characters in length. The first six characters are taken as the file-name, and last three characters are taken as the extension. The programmer must provide spaces as required to conform to this convention.

Examples:

(1) VALUE OF IDENTIFICATION IS "COST Δ Δ TST"

(2)
```
VALUE OF IDENTIFICATION IS FILE-1-NAME
        .
        .
(WORKING-STORAGE SECTION.)
        .
        .
77 FILE-1-NAME PICTURE IS X(9).
```

d. DATE-WRITTEN represents the date that a file (with STANDARD labels) was written. If a data-name is specified, it must be associated with a DISPLAY-6 or DISPLAY-7 data item six characters in length. If a literal is specified, it must be a nonnumeric or numeric literal six characters in length. The first two characters are taken as year, the next two as month, and the last two as day. The DATE-WRITTEN clause is ignored when the file is OPENed for output; instead, the current date is used.

Examples:

(1) VALUE OF IDENTIFICATION IS "RANDOMXYZ", DATE-WRITTEN IS 690612

(2) VALUE OF IDENTIFICATION IS "DATA ΔΔΔΔΔ", DATE-WRITTEN IS FILE-1-DATE

```
        .
        .
(WORKING-STORAGE SECTION.)
77 FILE-1-DATE PICTURE IS 9(6).
```

e. USER-NUMBER represents the project-programmer number of the owner of a disk file; it is ignored for all other devices. Data-name-3 must be a COMPUTATIONAL item of 10 or fewer digits in which the project-programmer numer is stored. Literal-3 and literal-4 are octal literals of six or fewer digits. Literal-3 is the project number and literal-4 is the programmer number.

f. For input files the VALUEs specified are checked against the file when it is opened. For output files, the VALUE OF IDENTIFICATION is written when the file is opened. Refer to Chapter 8, "Standard Label Procedures." If the specified values do not match a file on the selected medium, a run-time error message is issued.

g. If the access mode is INDEXED, only a literal can be specified for the VALUE OF IDENTIFICATION. It represents the index file, not the data file. The identification of the data file is stored in the index file.

h. If data-name-3 is used to represent the project-programmer number, the user must be aware that the value of data-name-3 is treated as decimal, even though the project-programmer number is octal. The data-name-3 value will be translated from decimal to binary by the COBOL conversion routine. Thus, the project-programmer number will not be accurate unless the user provides a conversion routine in his program to convert his octal project-programmer number to its decimal equivalent so that it will be converted to the correct binary number. The following example is a suggested method for performing the conversion.

```
77      ERR-FLAG,          PIC 9,          USAGE COMP.
77      HALF-NUM,          PIC S9(7),      USAGE COMP.
77      OCTAL-PPN,         PIC S9(10),     USAGE COMP.
77      DIGIT,             PIC 9,
01      PP-NUMBER,
        02 PROJ-NUMBER,    PIC 9(6).
        02 PROG-NUMBER,    PIC 9(6).
        02 EITHER-NUM,     PIC 9(6).
        02 X REDEFINES EITHER-NUM.
         03 PP-DIGIT,      PIC 9, OCCURS 6 TIMES, INDEXED BY I.

.
.
.
        ACCEPT PROJ-NUMBER, PROG-NUMBER.
        SET ERR-FLAG TO ZERO.
        MOVE PROJ-NUMBER TO EITHER-NUM.
```

(continued on next page)

```
                MOVE ZERO TO HALF-NUM.
                PERFORM CONVERT VARYING I FROM 1 BY 1 UNTIL I > 6.
                IF HALF-NUM > 32767, SET ERR-FLAG UP BY 1.
                COMPUTE OCTAL-PPN = HALF-NUM * 262144.
                MOVE PROG-NUMBER TO EITHER-NUM.
                MOVE ZERO TO HALF-NUM.
                PERFORM CONVERT VARYING I FROM 1 BY 1 UNTIL I > 6.
                COMPUTE OCTAL-PPN = OCTAL-PPN + HALF-NUM.
                IF ERR-FLAG IS NOT = 0, GO TO OCTAL-ERROR.
            .
            .
            .
        CONVERT.
                IF PP-DIGIT ( I ) = 8 OR 9, SET ERR-FLAG UP BY 1.
                COMPUTE HALF-NUM = 8 * HALF-NUM + PP-DIGIT ( I ) .
        *       THIS ROUTINE INVALID FOR PROJECT NUMBERS LARGER THAN 77777.
```

i. None of the VALUE clauses can appear in the LINKAGE SECTION.

RECORD DESCRIPTIONS

Following the FD for a file, a record description is given for each different record format in the file. A record description begins with a level-01 entry:

```
01 data-name
```

where the data-name is one of those listed in the DATA RECORDS clause of the FD.

A complete record description may be as simple as

```
01 data-name PICTURE picture-string.
```

or it may be more complex, where the 01-level is followed by a long series of data description entries of varying hierarchies that describe various portions and subportions of the record. A 01 level data-name in the FILE SECTION cannot be explicitly redefined (using the REDEFINES clause). However, because a file has only one record area, if more than one data-name is specified in the DATA RECORDS ARE clause in the FD, they implicitly redefine the first data-name.

Record Concepts

A record description consists of a set of data description entries which describe a particular logical record. Each data description entry consists of a level-number followed by a data-name (or FILLER) which is followed, as required, by a series of descriptive clauses.

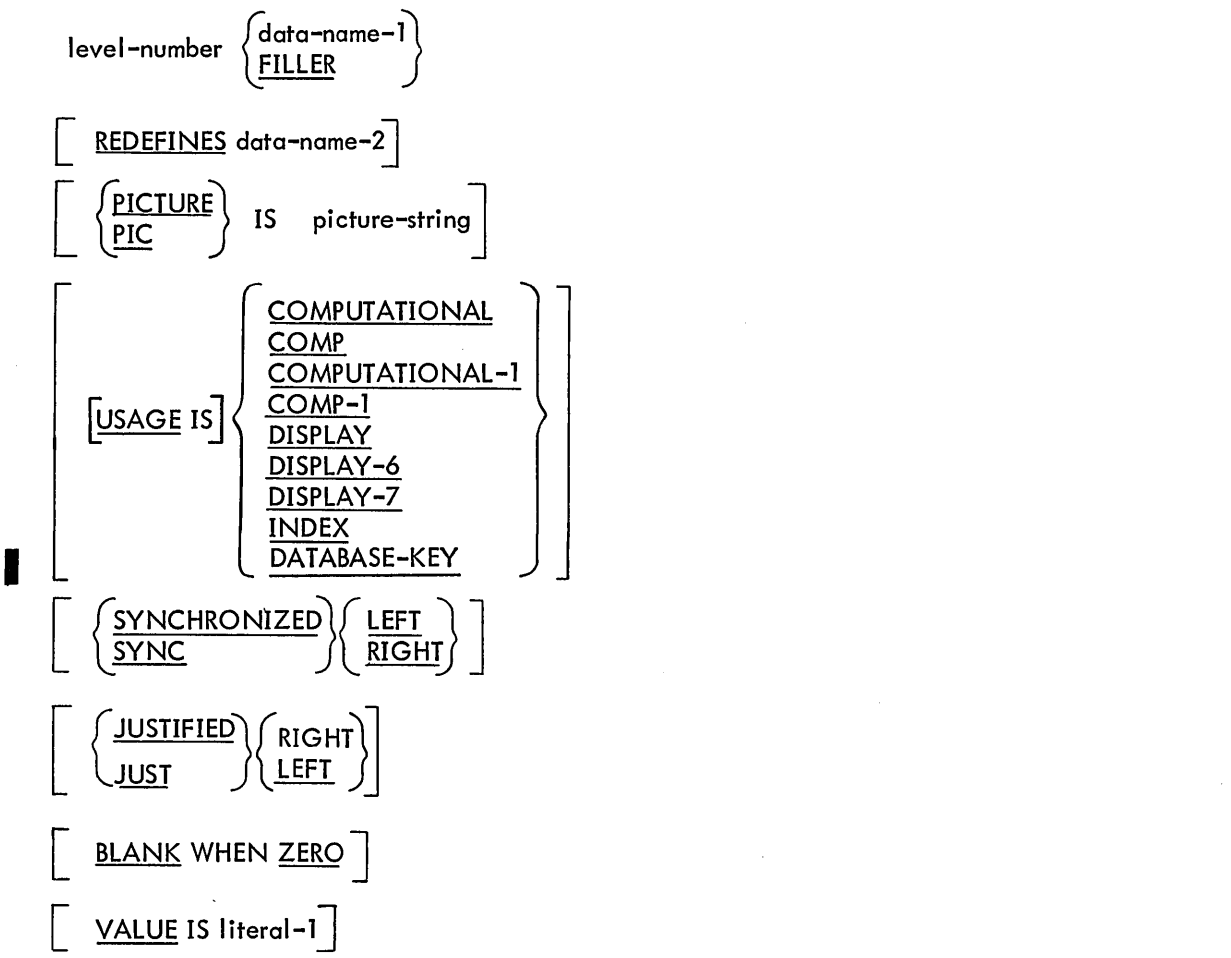
The general format of a data description entry follows.

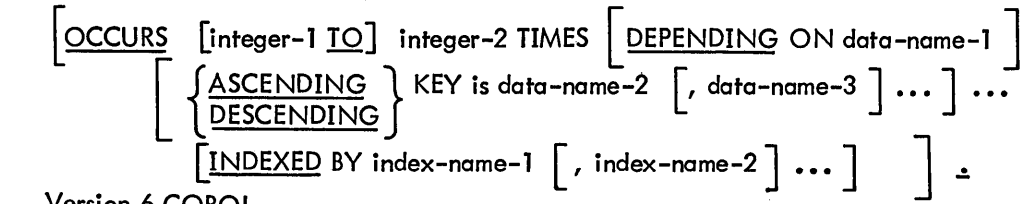
DATA DESCRIPTION ENTRY

Function

A data description entry describes a particular item of data.

General Format





RENAMES ENTRY

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

CONDITION-NAME ENTRY

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

   [, literal-3 [THRU literal-4]] ...  .
```

Technical Notes

a. Each data description entry must be terminated by a period. All semicolons and commas are optional.

b. The clauses may appear in any order, with one exception: the REDEFINES clause, when used, must immediately follow the data-name.

c. The VALUE clause must not appear in a data description entry which also contains an OCCURS clause, or in an entry which is subordinate to an entry containing an OCCURS clause. The latter part of this rule does not apply to condition-name (level-88) entries.

d. The PICTURE clause must be specified for every elementary item, except a USAGE INDEX or COMP-1 item.

e. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO can be specified only at the elementary level.

f. The clauses shown in the General Format appear in alphabetical order on the following pages.

BLANK WHEN ZERO

Function

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

General Format

```
[ BLANK WHEN ZERO ]
```

Technical Notes

a. When the BLANK WHEN ZERO option is used and the item is zero, the item is set to blanks.

b. BLANK WHEN ZERO can be specified only at the elementary level and only for numeric or numeric-edited items whose usage is DISPLAY-6 or DISPLAY-7.

c. More comprehensive editing features are available in the PICTURE clause. For example, if a PICTURE clause appears in the same data description entry and contains the zero suppression symbol* (zero suppress and replace with *), the field is replaced with * (see PICTURE).

condition-name (level-88)

Function

To assign a name to a value or range of values of the associated data item.

General Format

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

   [, literal-3 [THRU literal-4]] ... .
```

Technical Notes

a. Each condition-name requires a separate level-88 entry. This entry contains the name assigned to the condition, and the value or values associated with that condition. Condition-name entries must immediately follow the data description entry with which the condition-name is to be associated.

b. A condition-name entry can be associated with any elementary or group item except

(1) another condition-name entry, or

(2) a level-66 item.

c. Some examples of possible level-88 entries are given below.

(1)
```
05 B-FIELD PICTURE IS 99.
   88 B1 VALUE IS 3.
   88 B2 VALUES ARE 50 THRU 69.
   88 B3 VALUES ARE 20, 25, 28, 31 THRU 37.
   88 B4 VALUES ARE 70 THRU 75, 80 THRU 85, 90 THRU 95.
```

(2)
```
02 C-FIELD PICTURE IS XXX.
   88 C-YES VALUE IS "YES".
   88 C-NO VALUE IS "NOΔ".
```

d. The data item with which the condition-name is associated is called a conditional variable. A conditional variable may be used to qualify any of its condition-names. If references to a conditional variable require indexing, subscripting, or qualification, then references to its associated condition-names also require the same combination of indexing, subscripting, or qualification.

e. A condition-name is used in conditional expressions as an abbreviation for the related condition. Thus, if the above DATA DIVISION entries (Note c) are used, the statements in each pair below are functionally equivalent.

	Relational Expression	Condition-Name
(1)	IF B-FIELD IS EQUAL TO 3	IF B1 ....
(2)	IF B-FIELD IS GREATER THAN 49 AND LESS THAN 70	IF B2 ....
(3)	IF B-FIELD IS EQUAL TO 20 OR EQUAL TO 25 OR EQUAL TO 28 OR GREATER THAN 30 AND LESS THAN 38	IF B3 ....
(4)	IF B-FIELD IS GREATER THAN 69 AND LESS THAN 76 OR GREATER THAN 79 AND LESS THAN 86 OR GREATER THAN 89 AND LESS THAN 96	IF B4 ....
(5)	IF C-FIELD IS EQUAL TO "YES"	IF C-YES ....

f. Literal-1 must always be less than literal-2, and literal-3 less than literal-4. The values given must always be within the range allowed by the format given for the conditional variable. For example, any condition-name values given for a conditional variable with a PICTURE of PP999 must be in the range of .00000 to .00999. (See Note j under PICTURE in this chapter for the meaning of P in a picture-string.)

data-name/FILLER

Function

A data-name specifies the name of the data being described. The word FILLER specifies an unreferenced portion of the logical record.

General Format

level-number { data-name / FILLER }

Technical Notes

a. A data-name or the word FILLER must immediately follow the level-number in each data description entry.

b. A data-name must be composed of a combination of the characters A through Z, 0 through 9, and the hyphen. It must contain at least 1 alphabetic character and must not exceed 30 characters in length. It must not duplicate a COBOL reserved word. Refer to Section 2.2.3.2 for further information.

c. The key word FILLER is used to name an unreferenced item in a record (that is, an item to which the programmer has no reason for assigning a unique name). A FILLER item cannot, under any circumstances, be referenced directly in a PROCEDURE DIVISION statement. However, it may be indirectly referenced by referring to a group-level item of which the FILLER item is a part. FILLER can be used at any level, including the 01 level.

JUSTIFIED

Function

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format

Technical Notes

a. The JUSTIFIED clause cannot be specified at a group level or for numeric edited items. If neither RIGHT nor LEFT is specified, RIGHT is assumed for numeric DISPLAY items and LEFT is assumed for alphanumeric items.

b. An item subordinate to one containing a VALUE clause cannot be JUSTIFIED.

c. Only DISPLAY-6 and DISPLAY-7 items can be JUSTIFIED.

d. The standard rules for positioning data within an elementary data item are as follows:

(1) Receiving data item described as numeric or numeric-edited (see definitions in Notes f and i under PICTURE in this chapter). A numeric or numeric-edited item is justified according to the following rules, thus the JUSTIFIED clause cannot be used.

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.

If 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 the sending data is aligned according to this decimal point.

(2) Receiving data item described as alphanumeric (other than numeric edited) or alphabetic (see definitions in Notes e and g under PICTURE in this chapter).

The data is moved to the receiving character positions and aligned at the leftmost character position with space fill or truncation at the right end as required.

e. When a receiving item is described as JUSTIFIED LEFT, positioning occurs as in d (2) above.

f. When a receiving data item is described with the JUSTIFIED RIGHT clause and is larger than the sending data item, the data is aligned at the rightmost character position in the receiving item with space fill at the left end.

When a receiving data item is described with the JUSTIFIED RIGHT clause and is smaller than the sending data item, the data is aligned at the rightmost character position in the receiving item with truncation at the left end.

Examples are given below.

```
03  ITEM-A  PICTURE IS
    X(8) VALUE IS "ABCDEFGH".
03  ITEM-B  PICTURE IS
    X(4) VALUE IS "WXYZ".
03  ITEM-C  PICTURE IS X(6).
03  ITEM-D  PICTURE IS X(6)
    JUSTIFIED RIGHT.
```

	Contents of Receiving Field
MOVE ITEM-A TO ITEM-C.	A B C D E F
MOVE ITEM-A TO ITEM-D.	C D E F G H
MOVE ITEM-B TO ITEM-C.	W X Y Z Δ Δ
MOVE ITEM-B TO ITEM-D.	Δ Δ W X Y Z

level-number

Function

The level-number shows the hierarchy of data within a logical record. In addition, special level-numbers are used for condition-names (level-88), noncontiguous WORKING-STORAGE items (level-77), and the RENAMES clause (level-66).

General Format

level-number { data-name / FILLER }

Technical Notes

a. A level-number is required as the first element in each data description entry.

b. Level-numbers may be placed anywhere on the source line, at or after margin A.

c. Level-number 88 is described under "condition-name (level-88)", and level-number 66 is described under "RENAMES (level-66)", both in this section.

d. A further description of level-numbers and data hierarchy can be found in the introduction to this chapter.

OCCURS

Function

The OCCURS clause eliminates the need for separate entries for repeated data, and supplies information required for the application of subscripts and indexes.

General Format

```
[OCCURS [integer-1 TO] integer-2 TIMES [DEPENDING ON data-name-1]
[{ASCENDING}                                                    ]
[{DESCENDING} KEY IS data-name-2 [, data-name-3 ] ... ] ...     ]
[INDEXED BY index-name-1 [, index-name-2] ...]]
```

Technical Notes

a. This clause cannot be specified in a data description entry that has a 66 or 88 level-number, or in one that contains a VALUE clause.

b. The OCCURS clause is used to define tables or other homogeneous sets of repeated data. Whenever this clause is used, the associated data-name and any subordinate data-names must always be subscripted or indexed when used in all PROCEDURE DIVISION statements.

c. All clauses given in a data description that includes an OCCURS clause apply to each repetition of the item.

d. The integers must be positive. If integer-1 is specified, it must have a value less than integer-2. No value of a subscript can exceed integer-2; in addition, if the DEPENDING option is specified, no subscript can exceed the value of data-name-1 at the time of subscripting.

e. The value of data-name-1 is the count of the number of occurences of the item described by the OCCURS clause; its value must not exceed integer-2.

f. The DEPENDING option, when specified, must immediately follow TIMES. Data-name-1 must be USAGE INDEX or USAGE COMP of 10 digits or less with no scaling or decimal places. It cannot be subscripted or appear in the LINKAGE SECTION.

g. The KEY IS option is used to indicate that the repeated data is arranged in ascending or descending order according to the values associated with data-name-2, data-name-3, and so forth. The data-names are listed in descending order of significance.

h. Data-name-2 must be either the name of the entry containing the OCCURS clause, or the name of an entry subordinate to the entry containing the OCCURS clause. Data-name-3, etc., must be the name of an entry subordinate to the group item that is the subject of this entry.

i. An index-name defined in an OCCURS clause must not be defined elsewhere; its appearance in the INDEXED option is its only definition. There can be no items of the same name defined elsewhere. The USAGE of each index-name is assumed to be INDEX.

j. Subscripting and indexing are described in the introduction to this chapter.

PICTURE

Function

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

General Format

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

Technical Notes

a. A PICTURE clause may be used only at the elementary level. It may not be used with an item described as USAGE INDEX or COMP-1.

b. A picture-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. These symbols are as follows:

(1) Symbols representing data characters

9 represents a numeric character (0 through 9)

A represents an alphabetic character (A through Z, and the space)

X represents an alphanumeric character (any allowable character)

(2) Symbols representing arithmetic signs and assumed decimal point positioning

V represents the position of the assumed decimal point

P represents an assumed decimal point scaling position

S represents the presence of an arithmetic sign

(3) Symbols representing zero suppression operations

Z represents standard zero suppression (replacement of leading zeroes by spaces)

* represents check protection (replacement of leading zeroes by asterisks)

(4) Symbols representing insertion characters

$ represents a dollar sign (this sign floats from left to right and replaces rightmost leading zero when more than one $ appears)†

†If the CURRENCY SIGN IS clause appears in the SPECIAL-NAMES paragraph, the symbol specified by the literal must be used in all instances in place of the $.

, represents an insertion comma †

. represents an actual decimal point †

B represents an insertion blank

0 represents an insertion zero

(5) Symbols representing editing sign-control symbols

+ represents an editing plus sign — These float and replace rightmost leading zero

- represents an editing minus sign — when more than one + or - appear

CR represents an editing Credit symbol

DB represents an editing Debit symbol

(6) Consecutive repetitions of a picture-string symbol can be abbreviated to the symbol followed by (n), where n indicates the number of occurrences.

c. A maximum number of 30 symbols can appear in a picture-string. Note that the number of symbols in a picture-string and the size of the item represented are not necessarily the same. There are two reasons for this discrepancy. First, the abbreviated form for indicating consecutive repetitions of a symbol may result in fewer symbols in the picture-string than character positions in the item being described. For example, a data item having 40 alphanumeric character positions can be described by a picture-string of only 5 symbols:

PICTURE IS X(40)

The second reason is that some symbols are not counted when calculating the size of the data item being described. These symbols include the V (assumed decimal point), P (decimal point scaling position), and S (arithmetic sign); these symbols do not represent actual physical character positions within the data item. For example, the character-string

S999V99

represents a 5-position data item.

Other size restrictions for numeric and numeric edited items are given under the appropriate headings below.

d. There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. A description of each category is given in the notes below.

e. Definition of an Alphabetic Item

(1) Its picture-string may contain only the symbol A.

(2) It may contain only the 26 letters of the alphabet and the space.

f. Definition of a Numeric Item

(1) Its picture-string may contain only the symbols 9, P, S, and V. It must contain at least one 9.
The picture-string must have from 1 to 18 digit positions.

(2) It may contain only the digits 0 through 9 and an operational sign.

---

†If the DECIMAL-POINT IS COMMA clause appears in the SPECIAL-NAMES paragraph, the function of the comma and decimal point is reversed.

g. Definition of an Alphanumeric Item

(1) Its picture-string can consist of all Xs, or a combination of the symbols A, X, and 9 (except all 9s or all As). The item is treated as if the character-string contained all Xs.

(2) Its contents can be any combination of characters from the complete character set (see Section 1.2, Chapter 1).

h. Definition of an Alphanumeric Edited Item

(1) Its picture-string can consist of any combination of As, Xs, or 9s (it must contain at least one A or one X), plus at least one of the symbols B or 0.

(2) Its contents can be any combination of characters from the complete character set.

i. Definition of a Numeric Edited Item

(1) Its picture-string must contain at least one of the following editing symbols:

, . * + - Z 0 B CR DB $

It may also contain the symbols 9, V, or P.

The allowable sequences are determined by certain editing rules for each symbol and can be found in Note j.

The picture-string must have from 1 to 18 digit positions.

(2) The contents can be any combination of the digits 0 through 9 and the editing characters.

j. The symbols used to define the category of an elementary item and their functions are as follows.

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

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

Examples: (A-FLD contains the value 092469)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD.	99B99B99	0 9 Δ 2 4 Δ 6 9
MOVE A-FLD TO B-FLD.	9999BBBB	0 9 2 4 Δ Δ Δ Δ

Also see Note n, "Simple Insertion Editing".

P Each P in the picture-string indicates an assumed decimal point scaling position and is used to specify the location of an assumed decimal point when the point is outside the positions defined for the item. Ps are not counted in the size of the data item. They are counted in determining the maximum number of digit positions (18) allowed in numeric edited items or numeric items. Ps can appear only to the left or right of the picture-string and must appear together. The assumed decimal point is assumed to be to the left of the string of Ps if the Ps are at the left end of the picture-string and to

the right of the string of Ps if the Ps are at the right end of the picture-string. If the V symbol is used in this case, it must appear in either of those positions; it is redundant.

Examples:

PPP9999 (or VPPP9999) defines a data item of four character positions whose contents will be treated as .000nnnn during any decimal point alignment operation (such as in a MOVE or ADD).

9PPP (or 9PPPV) defines a data item of one character position whose contents will be treated as n000 during any decimal point alignment operation.

S An S in a picture-string indicates that the item has an operational sign and will retain the sign of any data stored in it. The S must be written as the leftmost character in the picture-string. If S is not included, all data will be stored in the item as an absolute value and will be treated as positive in all operations. The S symbol is not counted in the size of the data item.

V A V in a picture-string indicates the location of the assumed decimal point and may appear only once in a picture-string. The V does not represent a physical character position and is not counted in the size of the data item. If the assumed decimal point position is at the right of the rightmost character position of the item, the V is redundant (that is, 9999 is functionally equivalent to 9999V).

X Each X in a picture-string represents a character position which can contain any allowable character from the complete character set.

Z Each Z in a picture-string represents the leftmost leading numeric character positions in which leading zeroes are to be replaced by spaces. Each Z is counted in the size of the item.

* Each * in a picture-string represents the leftmost leading numeric character positions in which leading zeros are to be replaced by *. Each * is counted in the size of the item.

Examples: (A-FLD contains the value 00305)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	999999	0 0 0 3 0 5
MOVE A-FLD TO B-FLD	ZZ9999	Δ Δ 0 3 0 5
MOVE A-FLD TO B-FLD	ZZZZZZ	Δ Δ Δ 3 0 5
MOVE A-FLD TO B-FLD	ZZZZ.ZZ	Δ 3 0 5 . 0 0

Also see Note S, "Zero Suppression Editing".

9 Each 9 in a picture-string represents a character position which can contain a digit. Each 9 is counted in the size of the item.

0 Each 0 in a picture-string represents a character position into which a zero will be inserted. It is counted in the size of the item. The 0 symbol works in the same manner as the B symbol.

, Each, in a picture-string represents a character position into which a comma will be inserted.

Examples: (A-FLD contains 362577)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	9,999,999	0 , 3 6 2 , 5 7 7
MOVE A-FLD TO B-FLD	Z,ZZZ,ZZZ	Δ Δ 3 6 2 , 5 7 7

Also see Note n, "Simple Insertion Editing".

. A . (dot or period) in a picture-string is an editing symbol that represents an actual decimal point. It is used for decimal point alignment and also indicates where a point (.) is to be inserted. This symbol is counted in the size of the item. Only one . may appear in a picture-string.

Examples: (A-FLD contains 352699)†

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	99,999.99	0 3 , 5 2 6 . 9 9
MOVE A-FLD TO B-FLD	ZZ,ZZZ.ZZ	Δ 3 , 5 2 6 . 9 9
MOVE A-FLD TO B-FLD	99999.9999	0 3 5 2 6 . 9 9 0 0

See Noted under MOVE in Chapter 6 for a clarification of the rule governing the third example.

Also see Note O, "Special Editing".

+
-
CR
DB

Each of these symbols is used as an editing sign-control symbol. When used, they represent the character position(s) into which the editing sign-control symbol will be placed. Only one of these symbols can appear in a character-string.

The + and - symbols can appear either at the beginning or at the end of a picture-string. The CR and DB symbols can appear only at the end of a picture-string.

+ The character position containing this symbol will contain a + if the sending field either was unsigned (absolute) or had a positive operational sign; it will contain a - if the sending field had a negative operational sign.

- The character position containing this symbol will contain a space if the sending field either was unsigned (absolute) or had a positive operational sign; it will contain a - if the sending field had a negative operational sign.

CR
DB

Each of these symbols requires two character positions. The character positions containing either of these symbols will contain spaces if the sending field either was unsigned (absolute) or had a positive operational sign; they will contain the symbol specified if the sending field had a negative operational sign.

---

†The caret (^) symbol is used to indicate the location of the assumed decimal point.

Examples: (A-FLD contains 345625, B-FLD contains -345625)[†]

	C FLD picture-string	Result
MOVE A-FLD TO C-FLD	9999.99BCR	3 4 5 6 . 2 5 Δ Δ Δ
MOVE B-FLD TO C-FLD	9999.99BCR	3 4 5 6 . 2 5 Δ C R
MOVE A-FLD TO C-FLD	+9999.99	+ 3 4 5 6 . 2 5
MOVE B-FLD TO C-FLD	+9999.99	- 3 4 5 6 . 2 5
MOVE A-FLD TO C-FLD	-9999.99	Δ 3 4 5 6 . 2 5
MOVE B-FLD TO C-FLD	-9999.99	- 3 4 5 6 . 2 5
MOVE A-FLD TO C-FLD	9999.99DB	3 4 5 6 . 2 5 Δ Δ
MOVE B-FLD TO C-FLD	9999.99DB	3 4 5 6 . 2 5 D B
MOVE B-FLD TO C-FLD	$9999.99+	$ 3 4 5 6 . 2 5 -

Also see Note p, "Fixed Insertion Editing".

The + and - can also be used to perform floating insertion editing, a combination of zero suppression and symbol insertion. Floating insertion editing is indicated by the occurrence of two or more consecutive + or - symbols at the beginning of the picture-string. The total number of significant positions in the editing field must be at least one greater than the number of significant digits in the data to be edited. The floating + or - moves from left to right through any high-order zeros until a decimal point or the picture character 9 is encountered.

Examples: (A-FLD contains 005625; B-FLD contains -005625)

	C-FLD picture-string	Result
MOVE A-FLD TO C-FLD	++999.99	Δ + 0 5 6 . 2 5
MOVE B-FLD TO C-FLD	++++9.99	Δ Δ - 5 6 . 2 5
MOVE ZERO TO C-FLD	++999.99	Δ + 0 0 0 . 0 0
MOVE ZERO TO C-FLD	+++++.++	Δ Δ Δ Δ Δ Δ Δ Δ

(In order for floating to go past decimal point, all numeric positions of item must be represented by the floating insertion symbol)

MOVE A-FLD TO C-FLD	--999.99	Δ Δ 0 5 6 . 2 5
MOVE B-FLD TO C-FLD	--999.99	Δ - 0 5 6 . 2 5
MOVE ZERO TO C-FLD	---99.99	Δ Δ Δ 0 0 . 0 0
MOVE ZERO TO C-FLD	-----.--	Δ Δ Δ Δ Δ Δ Δ Δ

Also see Note o, "Floating Insertion Editing".

[†]The caret (Λ) symbol is used to indicate the location of the assumed decimal point.

Note that the + and - symbols are distinct from the S (operational sign) symbol. Normally, the + and - symbols are used to describe display items that are to appear on some printed report; they provide visual sign indication and cannot be used with items appearing as operands in arithmetic statements.

$ A $ (or the symbol specified by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph) represents the character position into which a $ (or the currency symbol) is to be placed. This symbol is counted in the size of the item.

Examples: (A-FLD contains 345675)
^

	B-FLD character-string	Result
MOVE A-FLD TO B-FLD	$9,999.99	$ 3 , 4 5 6 . 7 5
MOVE A-FLD TO B-FLD	$999,999.99	$ 0 0 3 , 4 5 6 . 7 5

Also see Note p, "Fixed Insertion Editing".

The $ symbol can also be used to perform floating insertion editing. Floating insertion editing is indicated by the occurrence of two or more consecutive $ symbols at the beginning of the character string. The total number of significant positions in the editing field must be at least one greater than the number of significant digits in the data to be edited. The floating $ symbol floats from left to right through any high-order zeros until a decimal point or the picture character 9 is encountered.

Examples: (A-FLD contains 005625)
^

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	$$9,999.99	Δ $ 0 , 0 5 6 . 2 5
MOVE A-FLD TO B-FLD	$$$,$$$.99	Δ Δ Δ Δ $ 5 6 . 2 5
MOVE ZERO TO B-FLD	$$$,999.99	Δ Δ Δ $ 0 0 0 . 0 0
MOVE ZERO TO B-FLD	$$$,$$$.$$	Δ Δ Δ Δ Δ Δ Δ Δ Δ Δ

Also see Note q, "Floating Insertion Editing".

k. There are two general methods of performing editing in the PICTURE clause:

(1) insertion, or

(2) suppression and replacement.

There are four types of insertion editing available:

(1) Simple insertion

(2) Special insertion

(3) Fixed insertion

(4) Floating insertion

There are two types of suppression and replacement editing:

(1) Zero suppression and replacement with spaces

(2) Zero suppression and replacement with asterisks

l. The type of editing that may be performed upon an item depends on the category to which the item belongs.

Category	Type of Editing Allowed
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion: 0 and B
Numeric Edited	All (except for the restrictions given in Note m)

m. Floating insertion editing and zero suppression/replacement editing are mutually exclusive in a PICTURE clause. Only one type of replacement can be used with zero suppression in a PICTURE clause.

n. Simple Insertion Editing (, B 0)

The , (comma), B (space), and 0 (zero) constitute those editing symbols used in simple insertion editing. These insertion characters represent the character position in the item into which the character will be inserted. These symbols are counted in the size of the item.

o. Special Insertion Editing (.)

The . (decimal point) symbol is used in special insertion editing. In addition to its use as an insertion character, it also represents the position of the decimal point for decimal point alignment. This symbol is counted in the size of the item. The symbols . and V (assumed decimal point) are mutually exclusive in a PICTURE clause. If the . is the last symbol in the character-string, it must be immediately followed by one of the punctuation characters (semi-colon or period) followed by a space.

p. Fixed Insertion Editing ($ + - CR DB)

The currency symbol ($) and the editing sign control characters (+ - CR DB) constitute the characters used in fixed insertion editing. Only one $ and one of the editing sign control characters can be used in a PICTURE character-string. When the symbols CR or DB are used, they represent two character positions in determining the size of the item. The symbols + or - when used must be the leftmost or rightmost character positions to be counted in the size of the item. The $ when used must be the leftmost character position to be counted in the size of the item, except that it can be preceded by a + or - symbol. A fixed insertion editing character appears in 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 on the value of the data being edited.

Editing Symbol in PICTURE character-string	Result	
	Data Positive	Data Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

q. Floating Insertion Editing ($$ ++ --)

The $ and the editing sign control symbols + and - are the floating insertion editing characters and are mutually exclusive in a given PICTURE string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the allowable insertion characters to represent the leftmost numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters embedded in the string of floating insertion characters or to the immediate right of this string are part of the floating string.

In a PICTURE character-string, there are only two ways of representing floating insertion editing:

(1) Represent any two or more of the leading numeric character positions on the left of the decimal point by the insertion character. The result is that a single insertion character will be placed in the character position immediately preceding the leftmost nonzero digit of the data being edited or in the character position immediately preceding the decimal point, or in the character position represented by the rightmost insertion character, whichever is encountered first.

(2) Represent all numeric character positions in the character-string by the insertion character. If the value is not zero, the result is the same as when the insertion character appears only to the left of the decimal point. If the value is zero, the entire item is set two spaces.

A picture-string containing floating inserting characters must contain at least one more floating insertion character than the maximum number of significant digits in the item to be edited. For example, a data field containing five significant digit positions requires an editing field of at least six significant positions.

All floating insertion characters are counted in the size of the item.

r. Zero Suppression Editing (Z *)

The suppression of leading zeroes and commas in a data field is indicated by the use of the Z or the * symbol in a picture-string. These symbols are mutually exclusive in a given picture-string. Each suppression symbol is counted in the size of the item. If a Z is used, the replacement character is a space. If an * is used, the replacement character is an *.

Zero suppression and replacement is indicated by a string of one or more Zs or *s to represent the leading numeric-character positions which are to be replaced when the associated character position in the data contains a leading zero. Any of the simple insertion characters embedded in this string of zero suppression symbols or to the immediate right of this string are part of the string.

If the zero suppression symbols appear only to the left of the decimal point, any leading zero in the data that corresponds to a zero suppression 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 in the string or at the decimal point, whichever is encountered first.

If all numeric character positions in the picture-string are represented by the suppression symbol and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire item will be set to the replacement character (with the exception of the decimal point if the suppression symbol is an *).

When the * is used and the clause BLANK WHEN ZERO appears in the same entry and zeros are moved to the field, all character positions with the exception of the decimal point are replaced by *.

s. The symbols + - * Z and $ when used as floating replacement characters are mutually exclusive within a given picture-string.

t. The following chart shows the order of precedence of the various picture-string symbols. Each "Y" on the chart indicates that the symbol in the top row directly above can precede the symbol at the left of the row in which the "Y" appears.

{ } indicate that the symbols are mutually exclusive.

The P and the fixed insertion + and - appear twice.

P9, +9, and -9 represent the case where these symbols appear to the left of any numeric positions in the string.

9P, 9+, and 9- represent the case where these symbols appear to the right of any numeric positions in the string.

The Z, *, and the floating ++, --, and $$ also appear twice.

Z., *., $$., ++., and --. represent the case where these symbols appear before the decimal point position.

.Z, .*, .$$, .++, and .-- represent the case where these symbols appear following the decimal point position.

		FIXED INSERTION								OTHER											
		B	0	,	.	{+9 -9}	{9+ 9-}	{CR DB}	$	A X	P9	9P	S	V	{Z. *.}	{.Z .*}	9	{++. --.}	{.++ .--}	$$.	.$$
FIXED INSERTION	B	Y	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y
	0	Y	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y
	,	Y	Y	Y	Y	Y			Y		Y			Y	Y	Y	Y	Y	Y	Y	Y
	.	Y	Y	Y		Y			Y		Y				Y		Y	Y		Y	
	{+9 -9}										Y			Y							
	{9+ 9-}	Y	Y	Y	Y				Y		Y			Y	Y	Y	Y			Y	Y
	{CR DB}	Y	Y	Y	Y				Y		Y			Y	Y	Y	Y			Y	Y
	$					Y					Y			Y							
OTHER	A X	Y	Y							Y							Y				
	P9										Y		Y	Y							
	9P	Y	Y	Y		Y	Y	Y	Y			Y	Y		Y		Y	Y		Y	
	S																				
	V	Y	Y	Y		Y	Y	Y	Y			Y	Y		Y		Y	Y		Y	
	{Z. *.}	Y	Y	Y		Y			Y						Y						
	{.Z .*}	Y	Y	Y	Y	Y			Y		Y			Y	Y	Y					
	9	Y	Y	Y	Y	Y			Y	Y	Y		Y	Y	Y		Y	Y		Y	
	{++. --.}	Y	Y	Y					Y									Y			
	{.++ .--}	Y	Y	Y	Y				Y		Y			Y				Y	Y		
	$$.	Y	Y	Y		Y														Y	
	.$$	Y	Y	Y	Y	Y					Y			Y						Y	Y

REDEFINES

Function

The REDEFINES clause allows the same core memory area to be allocated to two or more data items.

General Format

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

Technical Notes

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

b. The level-numbers of data-name-1 and data-name-2 must be identical.

c. This clause must not be used for level-number 66 or 88 items. Also, it must not be used for level-01 entries in the FILE SECTION; implicit redefinition is provided by specifying more than one data-name in the DATA RECORDS ARE clause in the FD.

d. When the level-number of data-name-2 is other than level-01, it should specify a storage area of the same size as data-name-1. FILLER items may be used to comply with this rule.

e. The REDEFINES entry must immediately follow the entries describing data-name-2.

f. The data description entry for data-name-2 cannot contain an OCCURS clause or be subordinate to an entry that contains an OCCURS clause. Also, the redefinition entries cannot contain VALUE clauses, except in condition-name (level-88) entries.

g. Data-name-2 must not be qualified.

h. The following example illustrates the use of the REDEFINES entry. The entries shown cause AREA-A and AREA-B to occupy the same area in memory.

```
03 AREA-A.
    04 FIELD-1 PICTURE IS X(7).
    04 FIELD-2 PICTURE IS A(13).
    04 FIELD-3.
```

```
        05  SUBFIELD-1  PICTURE IS
            S999V99 USAGE IS COMP.
        05  SUBFIELD-2  PICTURE IS
            S999V99 USAGE IS COMP.
03  AREA-B REDEFINES AREA-A.
    04  FIELD-A  PICTURE IS X(22).
    04  FIELD-B  PICTURE IS X(5).
    04  FILLER PICTURE IS X (9).
```

Note how the length of each area is calculated so that AREA-B can be defined so that its size is equal to that of AREA-A.

AREA-A:	FIELD-1	7	6-bit characters (DISPLAY-6 assumed)
	FIELD-2	13	6-bit characters (DISPLAY-6 assumed)
		4	6-bit characters (not used because COMP items must start at a new word boundary)
	SUBFIELD-1	6	6-bit characters (COMP items occupy one word, or six 6-bit character positions)
	SUBFIELD-2	6	6-bit characters (COMP items occupy one word, or six 6-bit character positions)
Total 6-bit characters		36	
AREA-B:	FIELD-A	22	6-bit characters (DISPLAY-6 assumed)
	FIELD-B	5	6-bit characters (DISPLAY-6 assumed)
	FILLER	9	6-bit characters (needed to make AREA-B size equal to AREA-A)
Total 6-bit characters		36	

RENAMES (level-66)

Function

The RENAMES clause permits alternate, possibly overlapping, groupings of elementary items.

General Format

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

Technical Notes

a. All RENAMES entries associated with items in a given record must immediately follow the last data description entry for that record.

```
01  data-name-a
      .
    (data description entries)
      .
      .
    (level-66 entries associated with this logical record)
01  data-name-b.
      .
      .
```

b. Data-name-1 cannot be used as a qualifier, and can be qualified only by the names of the level-01 or FD entries associated with it.

c. Data-name-2 and data-name-3 must be the names of items in the associated logical record and cannot be the same data-name.

Neither data-name-2 nor data-name-3 can have a level-number of 01, 66, 77, or 88. Neither of these data-names can have an OCCURS clause in its data description entry, nor be subordinate to an item that has an OCCURS clause in its data description entry.

Data-name-2 must precede data-name-3 in the record description, and data-name-3 cannot be subordinate to data-name-2. If there is any associated redefinition (REDEFINES), the ending point of data-name-3 must logically follow the beginning point of data-name-2.

When data-name-3 is specified, data-name-1 is a group item that includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item) and concluding with data-name-3 (or the last elementary item in data-name-3).

If data-name-3 is not specified, data-name-2 can be either a group or elementary item. If it is a group item, data-name-1 is treated as a group item and includes all elementary items in data-name-2; if data-name-2 is an elementary item, data-name-1 is treated as an elementary item with the same descriptive clauses.

d. The following examples illustrate the use of the RENAMES entry.

```
01  RECORD-NAME.
    02  FIRST-PART.
        03  PART-A.
            04  FIELD-1 PICTURE IS ...
            04  FIELD-2 PICTURE IS ...
            04  FIELD-3 PICTURE IS ...
        03  PART-B.
            04  FIELD-4 PICTURE IS ...
            04  FIELD-5.
                05  FIELD-5A PICTURE IS ...
                05  FIELD-5B PICTURE IS ...
    02  SECOND-PART.
        03  PART-C.
            04  FIELD-6 PICTURE IS ...
            04  FIELD-7 PICTURE IS ...
    66  SUBPART RENAMES PART-B THRU PART-C.
    66  SUBPART1 RENAMES FIELD-3 THRU SECOND-PART.
    66  SUBPART2 RENAMES FIELD-5B THRU FIELD-7.
    66  AMOUNT RENAMES FIELD-7.
```

SYNCHRONIZED

Function

The SYNCHRONIZED clause specifies the positioning of an elementary item within a computer word (or words).

General Format

Technical Notes

a. This clause can appear only in the data description of an elementary item.

b. This clause specifies that the item being defined is to be placed in an integral number of computer words and that it is to begin or end at a computer word boundary. No other adjacent fields are to occupy these words. The unused positions, however, must be counted when calculating (1) the size of any group to which this elementary item belongs, and (2) the computer core allocation when the item appears as the object of a REDEFINES clause. However, when a SYNCHRONIZED item is referenced, the original size of the item (as indicated by the PICTURE clause) is used in determining such things as truncation, justification, and overflow.

c. SYNCHRONIZED LEFT or SYNC LEFT specifies that the item is to be positioned in such a way that it will begin at the left boundary of a computer word.

SYNCHRONIZED RIGHT or SYNC RIGHT specifies that the item is to be positioned in such a way that it will terminate at the right boundary of a computer word.

d. When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is SYNCHRONIZED.

e. Any FILLER required to position the item as specified will be automatically generated by the compiler. The content of this FILLER is indeterminate.

f. COMP(UTATIONAL) and COMP(UTATIONAL)-1 items are always implicitly SYNCHRONIZED RIGHT, and therefore cannot be SYNCHRONIZED LEFT.

g. An item subordinate to one containing a VALUE clause cannot be SYNCHRONIZED.

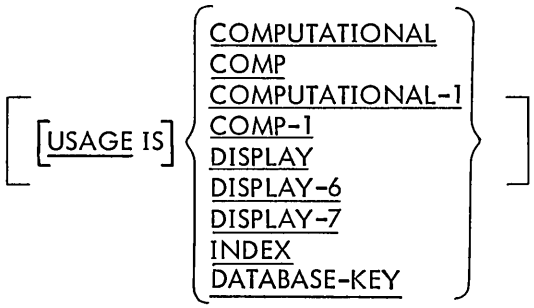
h. Only DISPLAY-6 or DISPLAY-7 items can be SYNCHRONIZED.

USAGE

Function

The USAGE clause specifies the format of a data item in computer storage.

General Format



Technical Notes

a. The USAGE clause can be written at any level. If it 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.

The implied USAGE of a group item is DISPLAY-7 if all items subordinate to it are defined as DISPLAY-7; otherwise, its USAGE is DISPLAY-6 (DISPLAY).

b. This clause specifies the manner in which a data item is represented within computer memory.

c. COMPUTATIONAL (COMP)

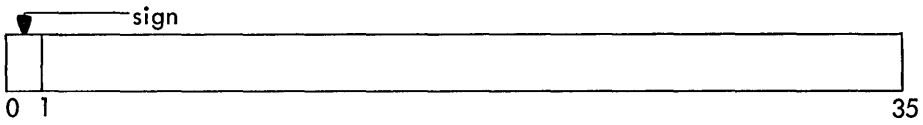
(1) COMP is equivalent to COMPUTATIONAL.

(2) A COMPUTATIONAL item represents a value to be used in computations and must be numeric. Its picture-string can contain only the symbols: 9 S V P
Its value is represented as a binary number with an assumed decimal point.

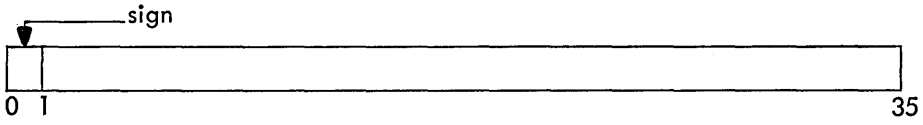
(3) If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. However, the group item itself is not COMPUTATIONAL and cannot be used as an operand in arithmetic computations.

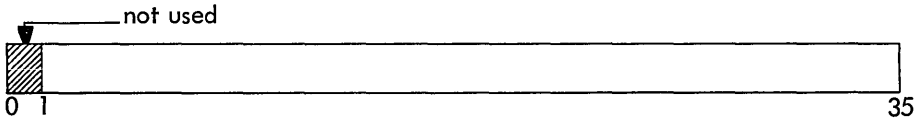
(4) COMPUTATIONAL items of 10 or fewer decimal positions will be SYNCHRONIZED RIGHT in one computer word. COMPUTATIONAL items of more than 10 decimal positions will be SYNCHRONIZED RIGHT in two full computer words.

(5) The following illustrations give the format of a COMPUTATIONAL item.



1-WORD COMPUTATIONAL ITEM





2-WORD COMPUTATIONAL ITEM

d. COMPUTATIONAL-1 (COMP-1)

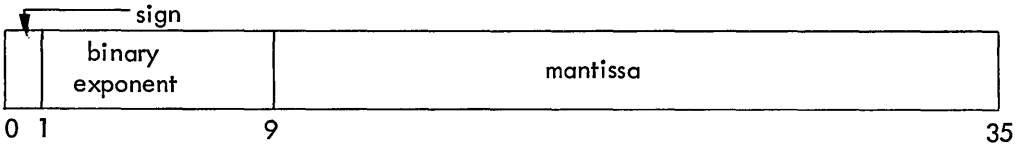
(1) COMP-1 is equivalent to COMPUTATIONAL-1.

(2) A COMPUTATIONAL-1 item can contain a value, in floating point format, to be used in computations and must be numeric. A COMP-1 item must not have a PICTURE.

(3) If a group item is described as COMPUTATIONAL-1, the elementary items within the group are COMPUTATIONAL-1. However, the group item itself is not COMPUTATIONAL-1 and cannot be used as an operand in arithmetic computations.

(4) COMPUTATIONAL-1 items will be SYNCHRONIZED in one full computer word.

(5) The following illustration gives the format of a COMPUTATIONAL-1 item.



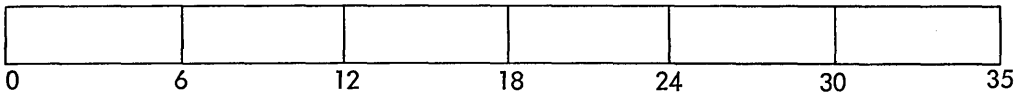
e. DISPLAY-6 (DISPLAY)

(1) DISPLAY is equivalent to DISPLAY-6.

(2) A DISPLAY-6 item represents a string of 6-bit characters. Its picture-string may contain any picture symbols. Refer to Appendix B for the SIXBIT collating sequence.

(3) DISPLAY-6 items may be SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT, as desired. Otherwise, they may share a computer word with other DISPLAY-6 items.

(4) The illustration below gives the format of a DISPLAY-6 word.



(5) If the USAGE clause is omitted for an elementary item, its USAGE is assumed to be DISPLAY-6 (DISPLAY).

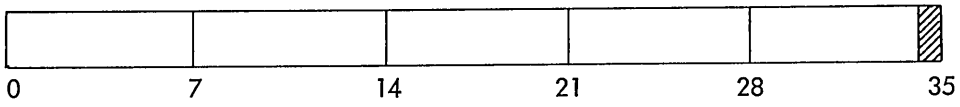
f. DISPLAY-7

(1) A DISPLAY-7 item represents a string of 7-bit ASCII characters. Its picture-string may contain any picture symbols.

(2) If any item in a record is DISPLAY-7, all items in that record must be DISPLAY-7.

(3) DISPLAY-7 items can be SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT, as desired; otherwise, they may share a computer word with other items. If the item is SYNCHRONIZED RIGHT, the last character of the item will end in bit 34 of a computer word.

(4) The illustration below gives the format of a DISPLAY-7 word.



g. INDEX

(1) An elementary item described as USAGE INDEX is called an index data-item. It is treated as a COMP item with PICTURE S9(5) and can be used as a COMP item.

(2) An index data-item must not have a PICTURE.

(3) If a group item is described as INDEX, the elementary items within the group are treated as INDEX. However, the group item itself is not INDEX and cannot be used as an operand in arithmetic statements.

(4) Index data items and index-names (defined in the OCCURS clause by the INDEXED BY option) are equivalent.

(5) If an index-name is defined in an OCCURS clause, it cannot be defined elsewhere.

h. DATABASE-KEY

(1) An item described as USAGE DATABASE-KEY is treated as a COMP item with PICTURE S9(10) and can be used as a COMP item.

(2) The item with USAGE DATABASE-KEY must not have a PICTURE.

(3) An item with USAGE DATABASE-KEY is primarily used in programs accessing data bases through the DECsystem-10 Data Base Management System (DBMS-10). This item can be used to store the value of a data base key. All data base keys are assigned by DBMS-10 and cannot be changed by the user. Refer to the DBMS-10 Programmer's Procedures Manual (DEC-10-APPMA-A-D) for more information about DBMS-10.

VALUE

Function

The VALUE clause defines the initial value of WORKING-STORAGE items, and the values associated with condition-names (level-88).

General Format

```
FORMAT 1:  [ VALUE IS literal ]

FORMAT 2:  [ { VALUE IS   }  literal-1 [ THRU literal-2 ]
           [ { VALUES ARE }
                                 [ , literal-3 [ THRU literal-4 ] ] ... ]
```

Technical Notes

a. Format 2 can be specified only for level-88 items.

b. In the FILE SECTION, the VALUE clause can be used only with level-88 items. In the WORKING-STORAGE SECTION, it can be used at all levels, except level-66. It must not be stated in a data description entry that contains an OCCURS clause or that is subordinate to an entry containing an OCCURS clause. Also, it must not be stated in an entry that contains a REDEFINES clause or that is subordinate to an entry that contains a REDEFINES clause (unless the VALUE clause is part of a level-88 entry).

c. If the VALUE clause is used at a group level, the literal must be a figurative constant or a nonnumeric literal. The group item is initialized to this value without consideration for the individual elementary or group items contained within this group. No VALUE clauses can appear at subordinate levels within the group.

d. If no VALUE clause appears for a WORKING-STORAGE item, the initial value of the item is unpredictable.

e. More information concerning Format 2 can be found under "condition-name (level-88)" in this chapter.

f. The VALUE clause must not conflict with other clauses in the data description entry or in the data description entries within the hierarchy of the item. The following rules apply:

(1) If the category of an item is numeric, all literals in the VALUE clause must be numeric. All literals in a VALUE clause must have a value within the range of values indicated by the PICTURE clause; for example, an item with PICTURE PPP9 may have only the values in the range .0000 through .0009.

(2) If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal will be aligned according to the normal alignment rules (see "JUSTIFIED") except that the number of characters in the literal must not exceed the size of the item.

(3) If the category of an item is numeric-edited or alphanumeric-edited, no editing of the value is performed in the VALUE clause.

(4) The USAGE of the literal agrees with the USAGE of the item. Thus, if the item has USAGE DISPLAY-6, the literal also has USAGE DISPLAY-6 and its value must contain legal SIXBIT characters.

g. The figurative constants SPACE(S), ZERO(E)(S), QUOTE(S), LOW-VALUE(S), and HIGH-VALUE(S) may be substituted for a literal. If the item is numeric, only ZERO(E)(S), LOW-VALUE(S), and HIGH-VALUE(S) are allowed.

REPORT SECTION

The REPORT SECTION contains the descriptions of one or more reports and the report groups that make up each report.

Report groups are the basic elements of a report. Each report group is divided into report lines, which are in turn divided into fields. The report groups that can appear in a report are:

REPORT HEADING	printed once at the beginning
REPORT FOOTING	printed once at the end
PAGE HEADING	printed at the beginning of each page
PAGE FOOTING	printed at the end of each page
DETAIL	printed for each set of report data
CONTROL HEADING	printed at the beginning of each detail report group when a control break occurs
CONTROL FOOTING	printed at the end of each detail report group when a control break occurs

The detail report groups contain the data items that constitute the report. Data items within a detail group can be designated by the programmer as controls. These control items are in descending order of rank from FINAL, through major, intermediate, to minor. Each time a control item changes, a control break is said to occur; the control footings for that detail group are printed, and control headings for the next detail group are printed before the next detail group is printed. A FINAL control break occurs twice during the generation of a report, before the first detail line is printed and after the last detail line is printed. The most major control breaks least often and the most minor control breaks most often. If the most minor control field breaks, the control footing for that control field is generated, and the control heading for the next detail group for that control is generated. If a more major control field breaks, the control footings for all fields more minor than that which broke are generated, starting with the most minor and continuing up to the control footing for the control that broke. The control headings are then printed starting with the control field that broke and continuing through the most minor control field. An example of a skeleton report follows.

```
REPORT HEADING
PAGE HEADING
CONTROL HEADING (FINAL)
CONTROL HEADING (MAJOR)
CONTROL HEADING (MINOR)
DETAIL GROUP
          .
          .
          .
CONTROL FOOTING (MINOR) (control break occurred)
CONTROL HEADING (MINOR)
DETAIL GROUP
          .
          .
          .
CONTROL FOOTING (MINOR)
CONTROL FOOTING (MAJOR) (control break occurred)
CONTROL HEADING (MAJOR)
CONTROL HEADING (MINOR)
DETAIL GROUP
          .
          .
          .
CONTROL FOOTING (MINOR)
CONTROL FOOTING (MAJOR)
CONTROL FOOTING (FINAL) (control break occurred)
PAGE FOOTING
REPORT FOOTING
```

Within a report file, more than one report can be written. If more than one report is written in a file, the names of all the reports must be specified in the REPORTS clause of the file description entry, and a unique code must be specified for each report by means of the CODE clause in the Report Description of each report. The code must also be identified in the SPECIAL-NAMES section of the ENVIRONMENT DIVISION.

To print one of the reports within a report file, the user enters the filename and the code of the desired report into the queue for the line-printer spooler, LPTSPL. LPTSPL copies the report lines with the designated code to the line printer, but does not erase the lines from the file. The file is entered into the line-printer queue by means of the QUEUE monitor command. The code is specified by the /REPORT switch in the QUEUE command string.

```
.QUEUE = filename.ext/REPORT: code
```

Only the first 12 characters of the code will be accepted in the QUEUE command string.

Included in the description of a report are the number of lines on a report page, where headings should begin on the page, where footings should end, the column on the page where each item in a report group is to be placed, and the number of lines that are left between report groups.

To cause a report to be printed, in addition to specifying its format and data in the DATA DIVISION, the user must include certain verbs in the PROCEDURE DIVISION. These verbs are: INITIATE, which initializes the report and sets sum counters to zero; GENERATE, which causes report groups to be generated on specified control breaks; and TERMINATE, which ends the report. An additional statement, USE BEFORE REPORTING, causes programmer-specified procedures to be performed before a report group is produced.

Report Description (RD)

Function

The Report Description furnishes information concerning the physical structure for a report.

General Format

```
RD report-name
    [CODE mnemonic-name]
[ { CONTROL IS  } { FINAL                                          } ]
[ { CONTROLS ARE} { identifier-1 [,identifier-2] ...               } ]
[                 { FINAL, identifier-1 [,identifier-2] ...        } ]
  [ PAGE { LIMIT IS  } integer-1 { LINE  }
         { LIMITS ARE}           { LINES }
       [HEADING integer-2] [FIRST DETAIL integer-3]
       [LAST DETAIL integer-4] [FOOTING integer-5] ].
```

Technical Notes

a. The order of appearance of the optional clauses is immaterial.

b. The fixed data-name PAGE-COUNTER is automatically generated for each RD entry. Its function is to contain the current page number of a report. It is a computational item; its size is equal to the size of the largest field that refers to it in a SOURCE clause. The contents of the PAGE-COUNTER are set to 1 by the INITIATE statement.

c. The fixed data-name LINE-COUNTER is automatically generated for each RD entry. Its function is to contain the current line number within a report page. It is a computational item; its size is based on the number of lines specified in the PAGE-LIMIT clause.

d. PAGE-COUNTER or LINE-COUNTER may be referenced as if it were any data-name. It must be qualified by the report-name if more than one RD entry is present in the program.

e. Each of the above clauses appears on the following pages.

CODE

Function

The CODE clause defines a unique string of one or more characters that is affixed to each line of the report.

General Format

CODE mnemonic-name

Technical Notes

a. This clause is necessary only if more than one report is to be written in a single file.

b. Mnemonic-name is defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

c. The character string represented by mnemonic-name is affixed to the beginning of each report line, and is used to uniquely define the lines of separate reports written in one file.

d. The number of characters represented by mnemonic-name must be the same for the codes of all reports in the same file.

Function

The CONTROL clause indicates the identifiers that control the printing of totals in the report.

General Format

```
{CONTROL IS  }  {FINAL                                        }
{CONTROLS ARE}  {identifier-1 [,identifier-2] ...             }
                {FINAL, identifier-1 [, identifier-2] ...     }
```

Technical Notes

a. The CONTROL clause is required when CONTROL HEADING or CONTROL FOOTING report groups are specified.

b. The identifiers specify the control hierarchy for this report. They are listed in order from major to minor; FINAL is the highest level of control, identifier-1 is the major control, identifier-2 is the intermediate control, etc. The last identifier specified is the minor control.

c. Identifiers must be defined in the FILE or WORKING-STORAGE SECTION of the DATA DIVISION. They cannot be subscripted or indexed.

PAGE LIMIT

Function

The PAGE LIMIT clause indicates the specific line control to be maintained within the presentation of a report page.

General Format

PAGE { LIMIT IS / LIMITS ARE } integer-1 { LINE / LINES }

[HEADING integer-2] [FIRST DETAIL integer-3]

[LAST DETAIL integer-4] [FOOTING integer-5]

Technical Notes

a. The PAGE LIMIT clause is required when page format must be controlled by the Report Writer.

b. All integers must have a positive value less than 512. Integer-2 through integer-5 must not be greater than integer-1.

c. If absolute line spacing is indicated for all report groups (see the LINE and NEXT GROUP clauses described later in this section), integer-2 through integer-5 need not be specified.

d. The integers specify line numbers relative to the beginning of a page.

e. The HEADING clause specifies the first line of a page to be used; no line will precede integer-2.

f. The FIRST DETAIL clause specifies the first line of the first DETAIL or CONTROL print group; no DETAIL or CONTROL group will precede integer-3.

g. The LAST DETAIL clause specifies the last line of a DETAIL or CONTROL HEADING report group; no such group will extend beyond integer-4.

h. The FOOTING clause specifies the last line number of the last CONTROL FOOTING report group; no CONTROL FOOTING group will extend beyond integer-5.

i. If any optional clause is omitted, a value is assumed for its integer. The default values are:

integer-2:	default is 1
integer-3:	default is the value of integer-2
integer-4:	default is the value of integer 5 if specified; if integer-5 is also omitted, the default is the value of integer-1
integer-5:	default is the value of integer-4 if specified; if integer-4 is omitted, the default is the value of integer-1.

Report Group Description

Function

The Report Group Description entry specifies the characteristics and format of a particular report group.

General Format

Option 1

```
01 [data-name-1]
[ LINE NUMBER IS { integer-1
                 { PLUS integer-2
                 { NEXT PAGE                } ]
[ NEXT GROUP IS  { integer-3
                 { PLUS integer-4
                 { NEXT PAGE                } ]
                 { REPORT HEADING
                 { RH
                 { PAGE HEADING
                 { PH
                 { { CONTROL HEADING } { identifier-1 }
                 { { CH              } { FINAL        }
                 { DETAIL
TYPE IS          { DE
                 { { CONTROL FOOTING } { identifier-2 }
                 { { CF              } { FINAL        }
                 { PAGE FOOTING
                 { PF
                 { REPORT FOOTING
                 { RF
[ [USAGE IS] { DISPLAY   } ] .
             { DISPLAY-6 }
             { DISPLAY-7 }
```

(continued on next page)

Option 2

```
level-number [data-name-1]

    [BLANK WHEN ZERO]
    [COLUMN NUMBER IS integer-1]
    [GROUP INDICATE]
      [{JUSTIFIED} RIGHT ]
       {JUST     }

    [ LINE NUMBER IS {integer-2       } ]
                     {PLUS integer-3  }
                     {NEXT PAGE       }

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

    [RESET ON {identifier-1} ]
              {FINAL       }

    {SOURCE IS identifier-2                                                  }
    {SUM identifier-3 [,identifier-4]... [UPON data-name-2]                  }
    {VALUE IS literal-1                                                      }

    [[USAGE IS] {DISPLAY   } ] .
                {DISPLAY-6 }
                {DISPLAY-7 }
```

Technical Notes

a. Except for the data-name, which when present must immediately follow the level-number, the clauses may be written in any order.

b. In order for a report group to be referred to by a PROCEDURE DIVISION statement, it must have a data-name.

c. All elementary items must have a PICTURE clause and one of the clauses SOURCE, SUM, or VALUE.

d. For a detailed description of the BLANK WHEN ZERO, JUSTIFIED, PICTURE, VALUE, and USAGE clauses, see the pages following the Data Description Entry.

e. The data-name need not appear in an entry unless it is referred to by a GENERATE or USE statement, or reference is made to the SUM counter.

f. If the 01-level item is elementary, the clauses in Format 2 may be used in addition to the clauses in Format 1.

g. The remaining clauses are described in detail on the following pages.

COLUMN

Function

The COLUMN NUMBER clause indicates the column on the printed page in which the high-order (leftmost) character of an item will be printed.

General Format

COLUMN NUMBER IS integer

Technical Notes

a. Integer must have a positive value less than 512.

b. This clause is valid only for an elementary item.

c. Within a report group and a particular LINE NUMBER specification, COLUMN NUMBER entries must be indicated from left to right.

d. If the COLUMN NUMBER clause is omitted, the elementary item, though included in the description, is suppressed when the report group is produced at object time.

GROUP INDICATE

Function

The GROUP INDICATE clause indicates that this elementary item is to be produced only on the first occurrence of the item after any CONTROL or PAGE breaks.

General Format

GROUP INDICATE

Technical Notes

a. This clause can only be used at the elementary level within a TYPE DETAIL report group.

b. A GROUP INDICATEd item is presented in the first detail line of a report after any control breaks and after any page breaks; it is suppressed at all other times.

LINE NUMBER

Function

The LINE NUMBER clause indicates the absolute or relative line number entry in reference to the page or the previous entry.

General Format

```
LINE NUMBER IS { integer-1          }
               { PLUS integer-2     }
               { NEXT PAGE          }
```

Technical Notes

a. Integer-1 and integer-2 must be positive integers with values less than 512. Integer-1 must be within the range specified by the PAGE LIMITS clause in the RD entry.

b. The LINE NUMBER clause must be given for each report line of a report group, and must be specified at or before the first elementary item of each report line.

c. If a LINE NUMBER clause is specified for an item, all entries following that item, up to but not including the next item with a LINE NUMBER clause, are presented on the same line.

d. A LINE NUMBER at a subordinate level may not contradict a LINE NUMBER at a group level.

e. Integer-1 indicates that the current line is to be presented at that line number.

f. PLUS integer-2 indicates that the LINE-COUNTER is to be incremented by the value of integer-2, and that the current line is to be presented on the line specified by the new value of the LINE-COUNTER.

g. NEXT PAGE is used to indicate an automatic skip to the next page before the current line is presented.

NEXT GROUP

Function

The NEXT GROUP clause specifies the spacing condition following the last line of the report group.

General Format

```
NEXT GROUP IS { integer-1      }
              { PLUS integer-2 }
              { NEXT PAGE      }
```

Technical Notes

a. The NEXT GROUP clause may appear only at the 01 level of a report group.

b. Integer-1 and integer-2 must be positive integers with values less than 512. Integer-1 cannot exceed the number of lines specified by the PAGE LIMIT clause.

c. Integer-1 indicates a line number to which the LINE-COUNTER is set after the group is presented.

d. PLUS integer-2 indicates a relative line number that increments the LINE-COUNTER by the value of integer-2 after the group is presented. Integer-2 is the number of lines skipped following the last line of the report group.

e. NEXT PAGE indicates an automatic skip to the next page after the group is presented.

RESET

Function

The RESET clause indicates the CONTROL data-item that causes the SUM counter to be reset to zero on a control break.

General Format

$$\underline{\text{RESET}}\ \text{ON}\ \left\{\begin{array}{l}\text{identifier-1}\\ \underline{\text{FINAL}}\end{array}\right\}$$

Technical Notes

a. Identifier-1 must be one of the identifiers associated with the CONTROL clause in the RD entry.

b. The RESET clause may be used only in conjunction with a SUM clause at a CONTROL FOOTING elementary level.

c. Identifier-1 must be a higher level (more major) control identifier than the control identifier associated with this report group.

d. After a TYPE CONTROL FOOTING report group is presented, the sum counters associated with that group are automatically set to zero, unless an explicit RESET clause directs that the counter be cleared at a higher level.

Function

The SOURCE clause indicates the source of the data for a report item.

General Format

SOURCE IS identifier

Technical Notes

a. The SOURCE clause can only be given at the elementary level.

b. Identifier must indicate an item that appears in the FILE or WORKING-STORAGE SECTION.

c. The identifier cannot be subscripted or indexed.

d. When the report group is presented, the contents of this report item are replaced by the contents of identifier.

SUM

Function

The SUM clause indicates the items to be summed to produce the source of data for a report item.

General Format

SUM identifier-1 [, identifier-2] ... [UPON data-name-1]

Technical Notes

a. A SUM clause may appear only in a TYPE CONTROL FOOTING report group.

b. Each identifier must indicate a SOURCE item in a TYPE DETAIL report group, or a SUM counter in a TYPE CONTROL FOOTING report group.

c. If the SUM counter is referred to by a PROCEDURE DIVISION or REPORT SECTION statement, a data-name must be specified for the item. The data-name then represents the summation counter automatically generated by the Report Writer; that data-name does not represent the report group item itself.

d. A summation counter is incremented just before the presentation of the identifiers. Any editing of the SUM counters is done only when the sum item is presented; at all other times it is treated as a numeric item.

e. If higher-level report groups are indicated in the control hierarchy, each lower level that is figured into the sum is summed into the higher level before each lower level is reset, i.e., counters are rolled forward prior to the reset operation.

f. The UPON option is required to obtain selective summation for a particular data item that is named as a SOURCE item in two or more TYPE DETAIL report groups. Identifier-1 and identifier-2 must be SOURCE data items in data-name-1; data-name-1 must be the name of a TYPE DETAIL report group.

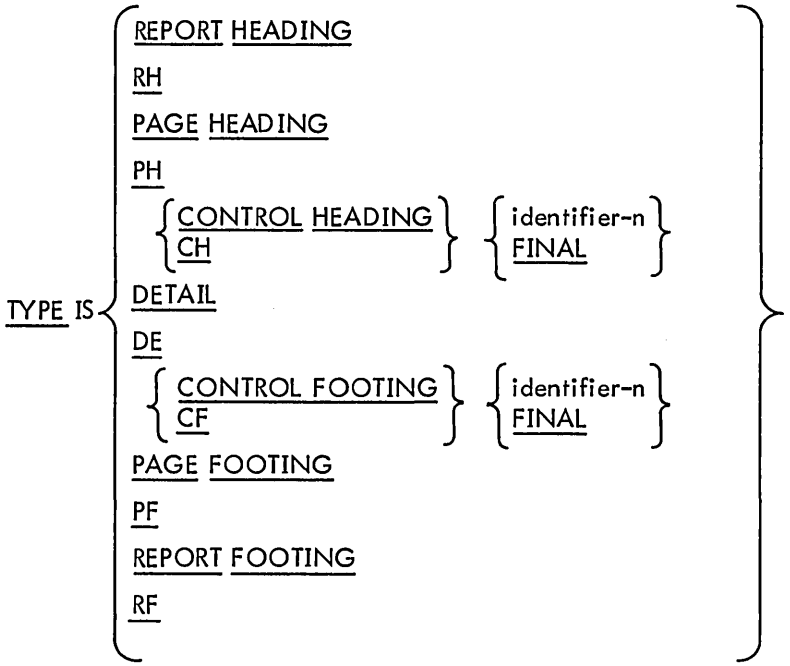
g. When the UPON option is used, summation occurs only when a GENERATE statement references data-name-1. It does not occur during summary reporting (refer to the GENERATE statement in the PROCEDURE DIVISION).

h. The identifiers cannot be subscripted or indexed.

Function

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time when the report group is generated.

General Format



Technical Notes

a. RH is an abbreviation for REPORT HEADING;
PH is an abbreviation for PAGE HEADING;
CH is an abbreviation for CONTROL HEADING;
DE is an abbreviation for DETAIL;
CF is an abbreviation for CONTROL FOOTING;
PF is an abbreviation for PAGE FOOTING;
RF is an abbreviation for REPORT FOOTING.

b. If the report group is described as TYPE DETAIL, the GENERATE statement in the PROCEDURE DIVISION directs the Report Writer to produce the named report group.

c. The REPORT HEADING entry indicates a report group that is produced only once at the beginning of a report, during the execution of the first GENERATE statement. There may be only one report group of this type in a report.

d. The PAGE HEADING entry indicates a report group that is automatically produced at the beginning of each page of the report. There may be only one report group of this type in a report.

e. The CONTROL HEADING entry indicates a report group that is produced at the beginning of a control group for a designated identifier. In the case of FINAL, it is produced once before the first control group during the execution of the first GENERATE statement. There may be only one report group of this type for each identifier and for FINAL.

f. The CONTROL FOOTING entry indicates a report group that is produced at the end of a control group for a designated identifier, or that is produced only once at the termination of a report in the case of FINAL. There may be only one report group of this type for each identifier and for FINAL. In order to produce any CONTROL FOOTING report groups, a control break must occur.

g. The PAGE FOOTING entry indicates a report group that is automatically produced at the bottom of each page of the report. There may be only one report group of this type in a report.

h. The REPORT FOOTING entry indicates a report group that is produced only once, at the termination of a report. There may be only one report group of this type in a report.

i. Each identifier, as well as FINAL, must be one of the identifiers associated with the CONTROL clause in the RD entry.

5.9 REPORT EXAMPLE

The following is an example of a program that generates two reports - a summary report and a detail report. Comments within the program (indicated by an asterisk in the continuation area) are used to point out report statements and their use in the program. The data for the report and the two reports follow the program.

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. EXAMPL.
   REMARKS.
       THIS PROGRAM IS AN EXAMPLE OF THE USE OF THE REPORT-WRITER.

       IT GENERATES TWO REPORTS: ONE IS A LIST OF CUSTOMERS BY
           CITY AND STATE, THE SECOND IS A LIST OF TOTAL NUMBER OF
           CUSTOMERS IN EACH STATE.

       BOTH REPORTS ARE GENERATED AT ONE TIME ONTO ONE FILE, TO BE
           SEPARATED BY THE LINE-PRINTER SPOOLER.

   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   OBJECT-COMPUTER. PDP-10; MEMORY SIZE 6 MODULES.
   SPECIAL-NAMES.
*      THESE MNEMONIC NAMES DEFINE CODES OF THE TWO REPORTS.
*      THE APPROPRIATE CODES WILL BE APPENDED TO THE BEGINNING OF EACH
*      LINE.
*      NOTE THAT THESE ARE NEEDED ONLY BECAUSE TWO REPORTS ARE WRITTEN
*      ONTO ONE FILE; IN THE NORMAL CASE OF ONE REPORT PER FILE, THEY
*      NEED NOT (AND PROBABLY WOULD NOT) BE SPECIFIED.

       'A' IS BY-CITY-CODE;  'B' IS STATE-TOTALS-CODE.

   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
       SELECT CUSTOMER-FILE;  ASSIGN TO DSK;
           RECORDING MODE IS ASCII.
       SELECT PRINTER-FILE; ASSIGN TO DSK.
       SELECT SORT-FILE; ASSIGN TO DSK, DSK, DSK, DSK, DSK.

   DATA DIVISION.
   FILE SECTION.

   SD  SORT-FILE.
   01  SORT-RECORD.
   02  SORT-NAME              PIC X(24).
   02  SORT-CITY              PIC X(20).
   02  SORT-STATE             PIC XX.
   02  SORT-STREET            PIC X(20).
   02  SORT-SALES             PIC S9(10); COMP.

   FD  CUSTOMER-FILE;
       VALUE OF IDENTIFICATION IS 'CUSTMRDAT'.
   01  CUSTMR-RECORD.
       02  CUSTMR-NAME        PIC X(24).
       02  CUSTMR-STREET      PIC X(20).
       02  CUSTMR-CITY        PIC X(20).
       02  CUSTMR-STATE       PIC XX.
       02  CUSTMR-SALES       PIC S9(10)V99.
       02  FILLER             PIC X(302).
```

```
*       THE 'REPORTS ARE' CLAUSE SAYS THAT THOSE RD-ENTRIES ARE DEFINED IN
*       THE REPORT SECTION, AND THE REPORTS WILL BE WRITTEN ONTO THIS FILE.

    FD  PRINTER-FILE;
        VALUE OF IDENTIFICATION IS 'CUSTMRLPT';
        REPORTS ARE STATE-TOTALS-ONLY, BY-CITY.
    01  PRINTER-RECORD          PIC X (89).
*       NOTE THAT THE SIZE OF THE PRINTER RECORD IS LARGE ENOUGH TO CONTAIN
*       THE LARGEST LINE WRITTEN, PLUS THE 1-CHARACTER CODE.
*       IT COULD BE LARGER WITH NO CHANGE IN OUTPUT, BUT IT CANNOT
*       BE SHORTER.

    WORKING-STORAGE SECTION.

    01  THIS-DATE                PIC X(8).
        01  TD-REDEFINED          REDEFINES THIS-DATE.
        02  TD-MONTH              PIC Z9.
        02  TD-HYF-1              PIC X.
        02  TD-DAY                PIC 99.
        02  TD-HYF-2              PIC X.
        02  TD-YEAR               PIC 99.

    01  UNEDITED-DATE.
        02  UE-YEAR               PIC 99.
        02  UE-MONTH              PIC 99.
        02  UE-DAY                PIC 99.
        02  FILLER                PIC X(6).

    77  ONE-COUNT                PIC S9; COMP; VALUE 1.
    77  CURRENT-STATE            PIC XX.

*       THE FOLLOWING IS THE GUTS OF THE PROGRAM.  IT DEFINES HOW THE
*       REPORTS ARE TO APPEAR ON THE PRINTED PAGES.

*       SEVERAL CLAUSES ARE USED AGAIN AND AGAIN.  THEY ARE:

*       TYPE -      THIS SPECIFIES WHERE THE RECORD WILL APPEAR IN THE REPORT.
*                   IT MUST BE SPECIFIED AT THE 01-LEVEL FOR EACH RECORD.
*                   THERE ARE SEVEN TYPES:
*                   REPORT HEADING (RH) - THIS APPEARS ONLY ONCE, AT THE
*                       BEGINNING OF THE REPORT.
*                   REPORT FOOTING (RF) - THIS APPEARS ONLY ONCE, AT THE
*                       END OF THE REPORT.
*                   PAGE HEADING (PH) - THIS APPEARS AT THE TOP OF EACH
*                       PAGE OF THE REPORT.
*                   PAGE FOOTING (PF) - THIS APPEARS AT THE BOTTOM OF EACH
*                       PAGE OF THE REPORT.
*                   CONTROL HEADING (CH) - THIS APPEARS JUST BEFORE ANY
*                       DETAIL LINES WHENEVER THE SPECIFIED CONTROL
*                       FIELD BREAKS (I.E. CHANGES FROM THE PREVIOUS
*                       RECORD), AND AT THE BEGINNING OF THE FIRST PAGE
*                       AFTER THE PAGE HEADING.
*                       THIS CLAUSE IS WRITTEN 'TYPE CH BREAK-NAME', WHERE
*                       'BREAK-NAME' IS THE FIELD TO TEST FOR A BREAK.
*                   CONTROL FOOTING (CF) - THIS IS PRODUCED WHENEVER THE
*                       SPECIFIED CONTROL FIELD BREAKS (SEE CONTROL HEADING)
*                       AFTER THE LAST DETAIL LINE BEFORE THE BREAK.
```

(Continued next page)

```
*                DETAIL (DE) - THIS IS THE DETAIL LINE, USUALLY PRINTED
*                    FOR EACH INPUT RECORD.  IT IS PRINTED WHENEVER
*                    THE 'GENERATE DETAIL-LINE-NAME' STATEMENT IS
*                    EXECUTED IN THE PROCEDURE DIVISION.
*      LINE -    THIS CLAUSE SPECIFIES WHERE ON THE PAGE THE RECORD IS TO
*                BE PRINTED.  IT APPLIES FROM THE ITEM WHICH CONTAINS THE
*                LINE CLAUSE THROUGH THE END OF THE RECORD, OR UNTIL
*                ANOTHER CLAUSE IS SPECIFIED.
*                THIS CLAUSE COMES IN THREE FLAVORS.
*                LINE N - 'N' IS AN INTEGER; THE RECORD WILL BE
*                    PRINTED ON THE NTH LINE OF THE PAGE.  IF THAT
*                    LINE HAPPENS TO BE BEFORE THE CURRENT LINE, A
*                    NEW PAGE WILL BE STARTED.
*                LINE PLUS N - AGAIN, 'N' IS AN INTEGER.  THIS ITEM WILL
*                    BE PRINTED AFTER SKIPPING N-1 LINES.
*                LINE NEXT PAGE - THE RECORD WILL BE PRINTED ON THE
*                    NEXT PAGE.  IF THIS IS A PAGE HEADING,
*                    IT WILL APPEAR ON LINE 1; OTHERWISE IT WILL APPEAR
*                    AFTER THE PAGE HEADING.
*      COLUMN -  THIS CLAUSE SPECIFIES WHERE ON A LINE THE ITEM WILL
*                APPEAR.  IF THIS CLAUSE IS NOT PRESENT FOR AN ITEM, THE
*                ITEM WILL NOT BE PRINTED.
*                COLUMNS MUST BE SPECIFIED IN INCREASING ORDER WITHIN
*                EACH LINE.
*      VALUE -   SAME AS IN FILE AND WORKING-STORAGE SECTIONS.
*      SOURCE -  THIS CLAUSE SPECIFIES WHAT IS TO BE PRINTED.  IN
*                EFFECT, THE DATA-NAME SPECIFIED IN THE SOURCE CLAUSE
*                IS MOVED TO THE ITEM BEFORE THAT ITEM IS PRINTED.
*      SUM -     THIS CLAUSE SPECIFIES THAT THE DATA-ITEM IS TO BE SUMMED
*                OR ACCUMULATED.  THE CLAUSE IS WRITTEN
*                'SUM DATA-NAME'; DATA-NAME WILL BE ADDED TO A COMPILER-
*                GENERATED ACCUMULATOR, AND THAT ACCUMULATOR IS MOVED
*                TO THE ITEM BEFORE IT IS PRINTED.
*                SUM CLAUSES MAY APPEAR ONLY IN 'TYPE CONTROL FOOTING'
*                RECORDS.  THE ACCUMULATOR IS CLEARED TO ZERO AFTER BEING
*                MOVED TO THE ITEM.

  REPORT SECTION.

*     THE FOLLOWING DESCRIBES THE SUMMARY LISTING.
*     THE CODE CLAUSE SAYS TO PUT A CODE (SEE SPECIAL-NAMES PARAGRAPH)
*     AT THE BEGINNING OF EACH LINE OF THE REPORT.
*     THE CONTROL CLAUSE SPECIFIES THE BREAK FIELDS, IN ORDER OF
*     MOST MAJOR TO MOST MINOR. 'FINAL' IS THE SPECIAL CASE, AND
*     MEANS 'AT THE END OF THE REPORT'.

  RD  STATE-TOTALS-ONLY;
      CODE STATE-TOTALS-CODE;
      CONTROLS ARE FINAL, SORT-STATE.
```

```
*       'NEXT GROUP PLUS 2' MEANS THAT WHATEVER RECORD FOLLOWS THIS ONE
*       WILL BE PRINTED 2 LINES AFTER THIS ONE.

  01  TYPE PH; NEXT GROUP PLUS 2.
      02  LINE 1 COLUMN 1    PIC X(25); VALUE 'STATE TOTALS OF CUSTOMERS'.
      02  LINE 2 COLUMN 1;   PIC X(8); SOURCE THIS-DATE.

  01  TYPE CF SORT-STATE; LINE PLUS 1.
      02  COLUMN 10          PIC ZZ, ZZ9; SUM ONE-COUNT.
      02  COLUMN 16          PIC X(5); VALUE ' FOR '.
      02  COLUMN 21          PIC XX; SOURCE CURRENT-STATE.
      02  COLUMN 23          PIC X; VALUE ','.
      02  COLUMN 33          PIC X (6); VALUE 'SALES:'.
      02  COLUMN 40          PIC $$,$$$,$$$,$$9; SUM SORT-SALES.

  01  TYPE CF FINAL; LINE PLUS 2.
      02  COLUMN 10          PIC ZZ,ZZ9; SUM ONE-COUNT.
      02  COLUMN 17          PIC X(14); VALUE 'FOR ALL STATES'.
      02  COLUMN 31          PIC X(8); VALUE ', SALES:'.
      02  COLUMN 40          PIC $$,$$$,$$$,$$9; SUM SORT-SALES.

*       THERE ARE SUM CLAUSES IN THE CONTROL FOOTING RECORDS WHICH
*       MENTION THESE ITEMS.  THERE IS A RULE ABOUT SUMMING:  ONE MAY
*       SUM ONLY THOSE ITEMS WHICH APPEAR IN A 'SOURCE' CLAUSE OF A
*       DETAIL ITEM, OR IN ANOTHER SUM CLAUSE.
*       NOTE THAT SINCE NO COLUMNS ARE SPECIFIED, NOTHING WOULD BE PRINTED
*       EVEN IF WE TRIED TO GENERATE THIS RECORD.

  01  TYPE DETAIL.
      02                      PIC S9(5); SOURCE ONE-COUNT.
      02                      PIC S9(10); SOURCE SORT-SALES.

*       THE FOLLOWING DESCRIBES THE DETAILED REPORT.
*       THE 'PAGE LIMIT' CLAUSE MEANS THAT THERE ARE NEVER
*       MORE THAN 58 LINES PUT ON A PAGE; GENERATION OF WHAT WOULD
*       BE THE 59TH LINE CAUSES AN AUTOMATIC 'TOP OF NEXT PAGE'
*       GENERATION, COMPLETE WITH ANY PAGE FOOTING AND/OR PAGE HEADING
*       LINES.
*       'FOOTING 58' MEANS THAT NO CONTROL FOOTING LINE WILL APPEAR AFTER
*       LINE 58, BUT INSTEAD WILL CAUSE AN AUTOMATIC SKIP TO TOP OF PAGE.
*       'FIRST DETAIL 3' MEANS THAT NO DETAIL, CONTROL HEADING, OR CONTROL
*       FOOTING LINE WILL APPEAR BEFORE LINE 3.
*       'LAST DETAIL 55' MEANS THAT NO DETAIL LINE WILL APPEAR AFTER
*       LINE 55.

  RD  BY-CITY
      CODE BY-CITY-CODE
      CONTROLS ARE FINAL, SORT-STATE, SORT-CITY;
      PAGE LIMIT IS 58 LINES;
          HEADING 1, FOOTING 58, FIRST DETAIL 3, LAST DETAIL 55.

  01  REPORT-HEADER; TYPE REPORT HEADING; LINE 25.
      02  COLUMN 27; PIC X(27); VALUE 'CUSTOMERS BY CITY AND STATE'.
      02  LINE 29 COLUMN 36; PIC X(8); SOURCE THIS-DATE.

  01  REPORT-FOOTER; TYPE REPORT FOOTING; LINE PLUS 2.
      02  COLUMN 30; PIC X(19); VALUE '** END OF REPORT **'.
```

```
*       'PAGE-COUNTER' IS A DATA ITEM GENERATED BY THE COMPILER FOR EACH
*       'RD' ITEM.  IT IS, AS ITS NAME IMPLIES, THE PAGE NUMBER.  IT IS
*       SET TO 1 BY AN 'INITIATE' STATEMENT (SEE PROCEDURE DIVISION),
*       AND BUMPED BY 1 FOR EACH NEW PAGE.
*       NOTE THAT SINCE WE HAVE TWO 'RD' ITEMS IN THIS PROGRAM, WE
*       MUST QUALIFY THE PAGE-COUNTER.

*       THERE IS A SIMILAR COMPILER-GENERATED ITEM, LINE-COUNTER, WHICH
*       SPECIFIES THE CURRENT LINE NUMBER WITHIN A PAGE.

    01  PAGE-HEADING; TYPE PAGE HEADING; LINE 1.
        02  COLUMN 1          PIC X(33); VALUE 'CUSTOMERS BY CITY AND STATE'.
        02  COLUMN 45         PIC X(8); SOURCE THIS-DATE.
        02  COLUMN 60         PIC X(4); VALUE 'PAGE'.
        02  COLUMN 65         PIC ZZZ9; SOURCE PAGE-COUNTER OF BY-CITY.

    01  TYPE CONTROL HEADING SORT-CITY; LINE PLUS 2; NEXT GROUP IS PLUS 1.
        02  COLUMN 1          PIC X(13); VALUE 'CUSTOMER NAME'.
        02  COLUMN 30         PIC X(6); VALUE 'STREET'.
        02  COLUMN 59         PIC X(5); VALUE 'SALES'.
        02  COLUMN 65         PIC X(5); VALUE 'STATE'.
        02  COLUMN 71         PIC X(4); VALUE 'CITY'.

*       'GROUP INDICATE' MEANS PRINT THIS ITEM ONLY ON THE FIRST DETAIL
*       LINE AFTER A CONTROL BREAK, OR ON THE FIRST DETAIL LINE OF A PAGE.

    01  DETAIL-LINE; TYPE DETAIL; LINE PLUS 1.
        02  COLUMN 1          PIC X(24); SOURCE SORT-NAME.
        02  COLUMN 30         PIC X(20); SOURCE SORT-STREET.
        02  COLUMN 51         PIC Z,ZZZ,ZZZ,ZZ9; SOURCE SORT-SALES.
        02  COLUMN 66         PIC XX; SOURCE SORT-STATE; GROUP INDICATE.
        02  COLUMN 69         PIC X(20); SOURCE SORT-CITY; GROUP INDICATE.
        02                     PIC S9(5); SOURCE ONE-COUNT.

    01  CITY-FOOTING; TYPE CONTROL  FOOTING SORT-CITY; LINE PLUS 2.
        02                COLUMN 10      PIC X(10); VALUE 'CITY TOTAL'.
        02  CITY-COUNT   COLUMN 22      PIC ZZ,ZZ9; SUM ONE-COUNT.
        02  CITY-SALES   COLUMN 29      PIC $$,$$$,$$$,$$9; SUM SORT-SALES.

    01  STATE-FOOTING; TYPE CF SORT-STATE; LINE PLUS 1; NEXT GROUP NEXT PAGE.
        02                 COLUMN 7      PIC XX; SOURCE CURRENT-STATE.
        02                 COLUMN 10     PIC X(11); VALUE 'STATE TOTAL'.
        02  STATE-COUNT   COLUMN 22      PIC ZZ,ZZ9; SUM CITY-COUNT.
        02  STATE-SALES   COLUMN 29      PIC $$,$$$,$$$,$$9; SUM CITY-SALES.

    01  FINAL-FOOTING; TYPE CF FINAL; LINE PLUS 2.
        02  COLUMN 10         PIC X(11); VALUE 'FINAL TOTAL'.
        02  COLUMN 22         PIC ZZ,ZZ9; SUM STATE-COUNT.
        02  COLUMN 29         PIC $$,$$$,$$$,$$9; SUM STATE-SALES.

    PROCEDURE DIVISION.

*       THE 'USE' PROCEDURE FOR A REPORT-RECORD IS EXECUTED JUST BEFORE
*       THAT RECORD IS PUT ONTO THE LISTING FILE.

    DECLARATIVES.
    EOR SECTION.  USE BEFORE REPORTING REPORT-FOOTER.
    EOR-A.  DISPLAY 'END OF REPORTS'.
    END DECLARATIVES.
```

```
MAIN SECTION.

START.
    SORT SORT-FILE ON ASCENDING KEYS SORT-STATE, SORT-CITY, SORT-NAME;
                INPUT PROCEDURE IS IN-PROCEDURE;
                OUTPUT PROCEDURE IS OUT-PROCEDURE.
    STOP RUN.

IN-PROCEDURE SECTION.

START.
    OPEN INPUT CUSTOMER-FILE.
LOOP.
    READ CUSTOMER-FILE; AT END GO TO DONE-INPUT.
    COMPUTE SORT-SALES ROUNDED = CUSTMR-SALES.
    MOVE CUSTMR-NAME TO SORT-NAME.
    MOVE CUSTMR-STATE TO SORT-STATE.
    MOVE CUSTMR-STREET TO SORT-STREET.
    MOVE CUSTMR-CITY TO SORT-CITY.
    RELEASE SORT-RECORD.
    GO TO LOOP.

DONE-INPUT.  CLOSE CUSTOMER-FILE.


OUT-PROCEDURE SECTION.

START.
    OPEN OUTPUT PRINTER-FILE.
    MOVE TODAY TO UNEDITED-DATE.
    MOVE UE-DAY TO TD-DAY; MOVE UE-MONTH TO TD-MONTH;
                MOVE UE-YEAR TO TD-YEAR; MOVE '-' TO TD-HYF-1, TD-HYF-2.

*   THE 'INITIATE' STATEMENT INITIALIZES A REPORT.  THE LISTING
*   FILE MUST BE OPEN, AND THE REPORT ITSELF MUST NOT ALREADY BE
*   INITIATED.

    INITIATE BY-CITY.
    INITIATE STATE-TOTALS-ONLY.

LOOP.
    RETURN SORT-FILE; AT END GO TO DONE-REPORTS.

*   THE 'GENERATE' STATEMENT CAUSES THE REPORT WRITER TO TEST THE
*   BREAK CONTROL FIELDS, AND PRODUCE ANY NECESSARY CONTROL
*   FOOTING AND/OR HEADING RECORDS.
*   IF THE OPERAND FOR THE GENERATE IS A DETAIL RECORD, THAT DETAIL
*   RECORD WILL BE PRINTED AFTER ANY BREAK PRINTING (THIS IS CALLED
*   'DETAIL REPORTING'); IF THE OPERAND IS AN 'RD' ITEM, ANY
*   DETAIL LINES FOR THAT 'RD' WILL BE SET UP, BUT NONE WILL BE
*   PRINTED (THIS IS CALLED 'SUMMARY REPORTING').
*   THE REPORT MUST BE INITIATED BEFORE GENERATES ARE ALLOWED.

    GENERATE DETAIL-LINE.
    GENERATE STATE-TOTALS-ONLY.

    MOVE SORT-STATE TO CURRENT-STATE.
    GO TO LOOP.
```

```
    DONE-REPORTS.
*       THE 'TERMINATE' STATEMENT COMPLETES THE PROCESSING FOR THE REPORT.
*       BREAKS ARE FORCED ON EACH CONTROL FIELD, SO THAT CONTROL FOOTING
*       RECORDS WILL BE PRINTED (CONTROL HEADINGS ARE NOT, HOWEVER). IN
*       ADDITION, ANY REPORT FOOTING RECORDS WILL BE PUT OUT.
*       THE REPORT MUST BE INITIATED BEFORE TERMINATION IS ALLOWED.
        TERMINATE BY-CITY, STATE-TOTALS-ONLY.
        CLOSE PRINTER-FILE.
```

The data that is input to the above program for the reports is shown below.

```
WORCESTER POLYTECHNIC     PINE POINT          WORCESTER   MA     8798858
DIGITAL EQUIPMENT CORP    MAIN STREET         MAYNARD     MA    76549381
MASSACHUSETTS BAY FISH    100 THE FENWAY      BOSTON      MA    12654700
ABERDEEN WIDGET           10400 PROXMIRE      ABERDEEN    KA       12378
PRUFROCK CEMENT           10 WASHINGTON ST    BOSTON      MA      100045
```

To have the individual reports printed from the file CUSTMR.LPT, the user must issue the QUEUE monitor command, which will add the reports to the line-printer queue. The command is

```
.QUEUE = CUSTMR.LPT /REPORT:A, /REPORT:B
```

A is the code for the first report (the detail report), and B is the code for the second report (the summary report). Both reports are shown on the following pages.

CUSTOMERS BY CITY AND STATE

10-12-71

CUSTOMERS BY CITY AND STATE 10-12-71 PAGE 1

CUSTOMER	NAME	STREET	SALES	STATE	CITY
ABERDEEN	WIDGET	10400 PROXMIRE	124	KA	ABERDEEN
	CITY TOTAL 1	$124			
KA	STATE TOTAL 1	$124			

CUSTOMER NAME	STREET	SALES	STATE	CITY
MASSACHUSETTS BAY FISH	100 THE FENWAY	126,547	MA	BOSTON
PRUFROCK CEMENT	10 WASHINGTON ST	1,000		
CITY TOTAL 2	$127,547			
CUSTOMER NAME	STREET	SALES	STATE	CITY
DIGITAL EQUIPMENT CORP	MAIN STREET	765,494	MA	MAYNARD
CITY TOTAL 1	$765,494			
CUSTOMER NAME	STREET	SALES	STATE	CITY
WORCESTER POLYTECHNIC	PINE POINT	87,989	MA	WORCESTER
CITY TOTAL 1	$87,889			
MA STATE TOTAL 4	$981,030			

FINAL TOTAL 5 $981,154

** END OF REPORT **

STATE TOTALS OF CUSTOMERS
10-12-71

1 FOR KA,	SALES:	$124
4 FOR MA,	SALES:	$981,030
5 FOR ALL STATES,	SALES:	$981,154

Chapter 6
The PROCEDURE DIVISION

The PROCEDURE DIVISION specifies the processing to be performed on the files and file data described in the ENVIRONMENT and DATA DIVISIONS. This processing is described by a series of COBOL procedure statements. Statements, sentences, paragraphs, and sections are described in Section 6.1. The PROCEDURE DIVISION must contain at least one paragraph, and each paragraph must contain at least one sentence. Sections are optional and permit a group of consecutive paragraphs to be referenced by a single procedure-name; sections can also be used for segmentation purposes (see "Segmentation"). If any section appears in the PROCEDURE DIVISION, then all paragraphs must appear within a section.

The first entry in the PROCEDURE DIVISION of a source program must be the division-header.

PROCEDURE DIVISION [USING identifier-1 [, identifier-2] ...] .

The next entry must be either the DECLARATIVES header (see "USE"), or a paragraph-name or section-name.

Only in a subprogram can USING clauses appear in the PROCEDURE DIVISION header. Refer to Section 8.11 for more information on subprograms.

When a program-name is specified in a CALL statement in a calling program, control is transferred to the beginning of the executable code in the subprogram (i.e., the PROCEDURE DIVISION).

The identifiers in the USING clause indicate those data items in the called program that may reference data items in the calling program. The order of identifiers in the CALL statement of the calling program and in the PROCEDURE DIVISION header of the called program is critical. The items in the USING clauses are related by their corresponding positions, not by name. Corresponding identifiers refer to a single set of data that is available to both the calling and the called programs.

The number of identifiers in the USING clause in the PROCEDURE DIVISION header must be less than or equal to the number of identifiers in the USING clause in the CALL statement in the calling program.

6.1 SYNTACTIC FORMAT OF THE PROCEDURE DIVISION

The PROCEDURE DIVISION consists of a series of procedure statements grouped into sentences, paragraphs, and sections. By grouping the statements in this manner, reference can be made to them via a procedure-name (i.e., a paragraph-name or a section-name). The order in which procedure-statements are executed can be controlled by using the sequence-control verbs ALTER, GO TO, and PERFORM.

6.1.1 Statements and Sentences

Statements fall into three categories: imperative, conditional, and compiler-directing, depending upon the verb used. Verbs, in turn, are also classified into certain categories. These categories and their relationship to the three statement categories are given in Table 6-1.

Table 6-1
Procedure Verb and Statement Categories

Verb	Verb Category	Statement Category
ADD COMPUTE DIVIDE MULTIPLY SUBTRACT	ARITHMETIC	IMPERATIVE
ALTER CALL ENTER ENTRY EXIT PROGRAM GOBACK GO TO PERFORM STOP	SEQUENCE-CONTROL	IMPERATIVE
EXAMINE MOVE SET STRING UNSTRING	DATA MOVEMENT	IMPERATIVE
CANCEL EXAMINE RELEASE RETURN SEARCH SORT TRACE	MISCELLANEOUS	IMPERATIVE

Table 6-1 (Cont)
Procedure Verb and Statement Categories

Verb	Verb Category	Statement Category
GENERATE INITIATE TERMINATE	REPORT	IMPERATIVE
ACCEPT CLOSE DELETE DISPLAY OPEN READ REWRITE SEEK WRITE	I-O	IMPERATIVE
IF	CONDITIONAL	CONDITIONAL
COPY EXIT NOTE USE	COMPILER-DIRECTING	COMPILER-DIRECTING
ACCEPT COUNT DISABLE ENABLE IF MESSAGE RECEIVE SEND	* COMMUNICATIONS	IMPERATIVE
CLOSE DELETE FIND GET IF INSERT INVOKE MODIFY MOVE OPEN REMOVE STORE USE	** DATA BASE MANAGEMENT	IMPERATIVE

*These verbs are used when writing programs for use with the DECsystem-10 Message Control System (MCS-10). Refer to the MCS-10 Programmer's Procedures Manual for more information.

**These verbs are used when writing programs for use with the DECsystem-10 Data Base Management System (DBMS-10). Refer to the DBMS-10 Programmer's Procedures Manual for more information.

A statement or sequence of statements terminated by a period forms a sentence. Sentences are classified into the same three categories as statements.

An imperative sentence consists solely of one or more imperative statements. Except for imperative sentences containing one of the sequence control verbs, control passes to the next procedural sentence following execution of the imperative sentence. If a GO TO or STOP RUN statement is present in an imperative sentence, it must be the last statement in the sentence.

A conditional sentence performs some test and, on the basis of the results of that test, determines whether a "true" or a "false" path should be taken. A conditional sentence is one that contains the conditional verb (IF) or one of the option clauses ON SIZE ERROR (used with arithmetic verbs), AT END (used with the READ verb), or INVALID KEY (used with the READ verb for mass storage devices).

A compiler-directing sentence consists of a single compiler-directing statement. Compiler-directing sentences are used to indicate the end point of a PERFORM loop (EXIT), insert comments in the PROCEDURE DIVISION (NOTE), and specify procedures for input-output errors and label handling (USE). Generally, compiler-directing sentences generate no object program coding.

6.1.2 Paragraphs

A single sentence or a group of sequential sentences can be assigned a paragraph-name for reference. The paragraph-name must begin in Area A (see Chapter 2) and terminate with a period. The first sentence of the paragraph can begin after the space following this period or it can begin on the next line, beginning in Area B.

A paragraph-name must be unique within its section, but need not be unique within the program. A non-unique paragraph-name must be qualified by its section-name except when it is referenced from within its own section.

6.1.3 Sections

A single paragraph or a group of sequential paragraphs can be assigned a section-name for reference. The section-name must begin in Area A and be followed by the word SECTION followed by a priority number, if desired, followed by a terminating period.

```
section-name SECTION nn.
```

If the section-name is in the DECLARATIVES portion, it may not have a priority number. A USE statement may appear following the terminating space after the period.

The section-name applies to all paragraphs following it until another section-header is encountered.

All section-names must be unique within a program. Sections are optional within the PROCEDURE DIVISION, but if a DECLARATIVES portion is used there must be a named section immediately following the END DECLARATIVES statement.

When a section-name is referenced, the word SECTION is not allowed in the reference.

6.2 SEQUENCE OF EXECUTION

In the absence of sequence-control verbs, sentences are executed consecutively within paragraphs, paragraphs are executed consecutively within sections, and sections are executed consecutively within the PROCEDURE DIVISION (with the exception of sections within the DECLARATIVES portion, which are executed individually when the related condition occurs).

6.3 SEGMENTATION AND SECTION-NAME PRIORITY NUMBERS

COBOL source programs can be written to enable certain portions of the PROCEDURE DIVISION code to share the same core memory area at object run time, thus decreasing the amount of core required to run the object program. The method used to achieve this reduction is called segmentation.

Segmentation consists of dividing the PROCEDURE DIVISION sections into logically related groupings called segments. The programmer defines a segment by assigning the same priority-number (a priority-number follows the word SECTION in the section-header, and can be in the range 00 through 99) to all the sections he wants included in that segment; these sections need not appear consecutively in the source program.

Segments are classified into three groups, depending upon their priority-number. These three groups are described in Table 6-2.

Table 6-2
Types of Segments

Priority-Number	Type	Description
None, or 00 up to SEGMENT-LIMIT minus 1	Resident Segment	This segment is always resident in core and is never overlaid.
SEGMENT-LIMIT up to 49	Nonresident; ALTERed GO TOs retained	These segments are nonresident and are brought into core when needed. Any ALTERed GO TOs retain their most recently set values.

Table 6-2 (Cont)
Types of Segments

Priority-Number	Type	Description
50 through 99	Nonresident; ALTERed GO TOs reset	These segments are also nonresident and are brought into core when needed. Any ALTERed GO TOs do not retain their latest values, but are reset to their original setting each time the segment is entered from another segment.

In addition to the resident segment, all data areas described in the DATA DIVISION are resident at all times. Thus, memory can be thought of as being divided into two parts:

a. A resident area, in which reside all data areas and the resident segment, and

b. A nonresident area, equal to the size of the largest nonresident segment, into which each nonresident segment is read when needed. Since each nonresident segment reads into the same memory area, any previous nonresident segment in that area is overlaid and must be brought in again when it is to be executed again.

The resident segment should consist of those sections that constitute the main portion of the processing. Infrequently used sections can be allocated to the nonresident segments.

6.4 ARITHMETIC EXPRESSIONS

An arithmetic expression is an identifier of a numeric elementary item, or a numeric literal, or such identifiers and literals separated by arithmetic operators.

Algebraic negation can be indicated by a unary minus symbol.

6.4.1 Arithmetic Operators

There are five arithmetic operators that may be used in arithmetic expressions. They are represented by specific character symbols that must be preceded by a space and followed by a space.

Arithmetic Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
**	Exponentiation
↑	Exponentiation

6.4.2 Formation and Evaluation Rules

The following rules for formation and evaluation apply to arithmetic expressions.

a. Parentheses specify the order in which elements within an arithmetic expression are to be evaluated. Expressions within parentheses are evaluated first. Within a nest of parentheses, the evaluation proceeds from the elements within the innermost pair of parentheses to the outermost pair of parentheses. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchal order of operations is implied:

First:	unary +, unary -	
then	** and ↑	(exponentiation)
then	* and /	(multiplication and division)
and then	+ and -	(addition and subtraction)

b. When the order of a sequence of operations on the same hierarchal level (e.g., a sequence of + and - operations) is not completely specified by use of parentheses, the order of operations is from left to right.

c. An arithmetic expression may begin only with one of the following:

(- + variable

and may end only with one of the following:

) variable

d. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression; each left parenthesis must precede its corresponding right parenthesis.

6.5 CONDITIONAL EXPRESSIONS

A conditional expression causes the object program to select between alternate paths (called the true and false paths) of control depending upon the truth value of a test. Conditional expressions can be used in conditional (IF) statements and in PERFORM statements (options 3 and 4). A conditional expression can be one of the following types:

Relation condition	(greater than, equal to, less than)
Class condition	(numeric or alphabetic)
Condition-name condition	(level-88 condition-names)
Switch-status condition	(SPECIAL-NAMES paragraph)
Sign condition	(positive, negative, zero)

Each of these types is discussed below.

6.5.1 Relation Condition

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, a figurative constant, or an arithmetic expression. Comparison of two numeric operands is permitted regardless of their formats as described by their respective USAGE clauses. Comparison of two operands is permitted if each is either DISPLAY-6 or DISPLAY-7. However, for all other comparisons, the operands must be described as having the same USAGE.

A numeric-edited operand may not be compared to a numeric operand. An alphanumeric operand may not be compared to a numeric operand unless the alphanumeric operand contains no characters other than numeric digits. For example, the statement:

```
IF NUM < "2".
```

is permissible but the statement:

```
IF NUM < "2.0".
```

is not.

6.5.1.1 Format of a Relation Condition

The general format for a relation condition is

```
{ identifier-1            }                          { identifier-2            }
{ literal-1               }  relational-operator     { literal-2               }
{ arithmetic-expression-1 }                          { arithmetic-expression-2 }
{ figurative-constant-1   }                          { figurative-constant-2   }
```

The first operand is called the subject of the condition; the second operand is called the object of the condition. Either the subject or the object must be an identifier or an arithmetic expression.

6.5.1.2 Relational Operators

Relational operators specify the type of comparison to be made in the relation condition. Relational operators must be preceded by a space and followed by a space.

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] > THAN	Greater than, not greater than
IS [NOT] LESS THAN IS [NOT] < THAN	Less than, not less than
IS [NOT] EQUAL (EQUALS) TO IS [NOT] = TO	Equal to, not equal to

6.5.1.3 Comparison of Numeric Items

For operands with a numeric category, a comparison results in the determination that the algebraic value of one of the operands is less than, equal to, or greater than the other operand. The number of digits contained in the operands is not significant. Zero is considered

a unique value regardless of the sign (i.e., +0 and -0 are considered equal). Unsigned numeric operands are considered positive for purposes of comparison.

6.5.1.4 Comparison of Nonnumeric Items - For operands whose category is nonnumeric (or where one operand is numeric and the other is nonnumeric), a comparison results in the determination that one of the operands is less than, equal to, or greater than the other operand with respect to a specified collating sequence of characters (see Appendix B). The size of an operand is the total number of characters in the operand.

There are three cases to consider: operands of equal size, operands of unequal size, and operands with differing justification.

a. Operands of equal size - If the operands are of equal size, characters in corresponding character positions of the two operands are compared, starting at the higher-order (leftmost) end and continuing through the low-order end. If all pairs of characters compare equally through the last pair, the operands are considered to be equal. If they do not all compare equally, the first pair of unequal characters encountered is compared to determine their relative position in the collating sequence. The operand containing the character that is positioned higher in the collating sequence is considered to be the greater operand.

b. Operands of unequal size - If the operands are of unequal size, the comparison of characters proceeds from the high-order end to the low-order end until either

(1) A pair of unequal characters is encountered, or

(2) One of the operands has no more characters to compare.

If a pair of unequal characters is encountered, the comparison is determined in the manner described for equal-sized operands.

If the end of one of the operands is encountered before unequal characters are encountered, this shorter operand is considered to be less than the longer operand unless the remaining characters in the longer operand are spaces, in which case the two operands are considered equal.

c. If one operand is right-justified and the other is left-justified, they are compared just as they appear in the record. That is, PICTURE XXX, VALUE "B" and PICTURE XXX, VALUE "B", JUSTIFIED RIGHT are not equal because the first appears in the record as B△△ and the second as △△B.

6.5.2 Class Condition

The class condition determines whether the operand is numeric (i.e., whether it consists entirely of the digits 0 through 9, with or without an operational sign) or alphabetic (i.e., whether it consists entirely of the characters A through Z and the space).

6.5.2.1 Format of a Class Condition - The general format of a class condition is

identifier IS [NOT] { NUMERIC / ALPHABETIC }

The identifier must be described, implicitly or explicitly, as DISPLAY, DISPLAY-6, or DISPLAY-7.

6.5.2.2 The NUMERIC Test - The NUMERIC test cannot be used with an item that is described as alphabetic. If the item being tested is not described as containing an operational sign, it will be considered numeric only if the contents are numeric and an operational sign is not present.

6.5.2.3 The ALPHABETIC Test - The ALPHABETIC test cannot be used with an item that is described as numeric. The item being tested is determined to be alphabetic only if the contents consist entirely of any combination of the alphabetic characters A through Z and the space.

6.5.3 Condition-Name Condition

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 (level-88).

6.5.3.1 Format of a Condition-Name Condition - The general format for a condition-name condition is

[NOT] condition-name

If the condition-name is associated with a range of values, then the conditional variable is tested to determine whether or not its value falls within this range, including the end values.

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

The result of the test is true if one of the values associated with the condition-name equals the value of its associated conditional variable.

6.5.4 Switch-Status Condition

A switch-status condition determines the on or off status of a hardware switch.

6.5.4.1 Format of a Switch-Status Condition - The general formats for a switch-status condition are

Format 1: [NOT] condition-name

Format 2: mnemonic-name IS [NOT] $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

Format 3: SWITCH (integer) IS [NOT] $\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\}$

In format 1, condition-name is associated with a SWITCH IS ON or OFF STATUS clause in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

In format 2, mnemonic-name is associated with a SWITCH (not an ON or OFF STATUS) in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

In format 3, integer must be in the range from 0 through 35.

In format 1, the result of the test is true if the switch is [NOT] set to the position associated with the condition-name.

In formats 2 and 3, the result of the test is true if the switch is [NOT] set to the position specified in the condition.

6.5.5 Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

6.5.5.1 Format of a Sign Condition - The general format for a sign condition is

$$\left\{\begin{matrix}\text{identifier}\\ \text{arithmetic-expression}\end{matrix}\right\} \text{ IS } [\underline{\text{NOT}}] \left\{\begin{matrix}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{matrix}\right\}$$

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 sign is ignored if the value is zero).

6.5.6 Logical Operators

The interpretation of any of the above conditions is reversed by preceding the condition with the logical operator NOT. Any of the above types of conditions can be combined by either of two logical operators. A logical operator must be preceded by a space and followed by a space.

Logical Operator	Meaning
OR	Entire condition is true if either or both of the simple conditions are true.
AND	Entire condition is true if both of the simple conditions are true.

6.5.7 Formation and Evaluation Rules

A conditional expression can be composed of either a simple-condition or a compound-condition. A simple-condition is one that performs a single test. A compound-condition is one that contains a string of simple-conditions connected by the logical operators AND, OR. A compound-condition can contain any combination of types of conditional expressions (relational, class, condition-name, switch-status, and sign).

The evaluation rules for conditions are analogous to those given for arithmetic expressions, except that the following hierarchy applies:

arithmetic-expressions
all relational operators
NOT
AND
OR

Parentheses may be used either to improve readability or to override the effects of the hierarchy given above. Each set of conditions within a pair of parentheses is reduced to a single condition. When this is accomplished, reductions which cross parentheses are done.

Examples:

a. Using parentheses for ease of reading.

The following expression

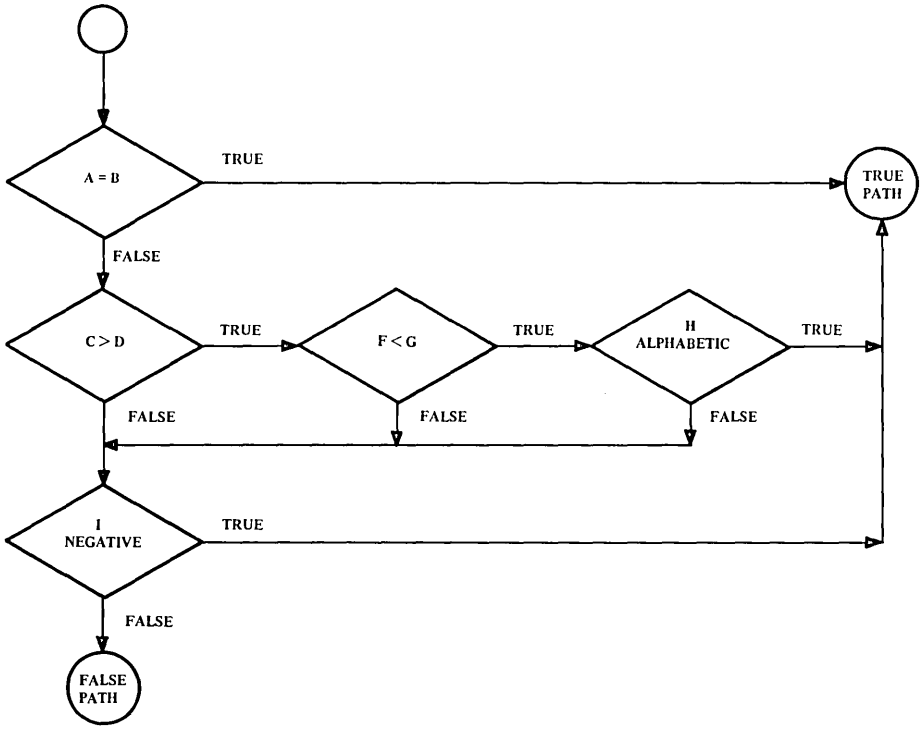
A = B OR C > D AND F < G AND H IS ALPHABETIC OR I IS NEGATIVE

can be parenthesized for readability without changing its effect as shown below.

(A = B) OR (C > D AND F < G AND H IS ALPHABETIC) OR (I IS NEGATIVE)

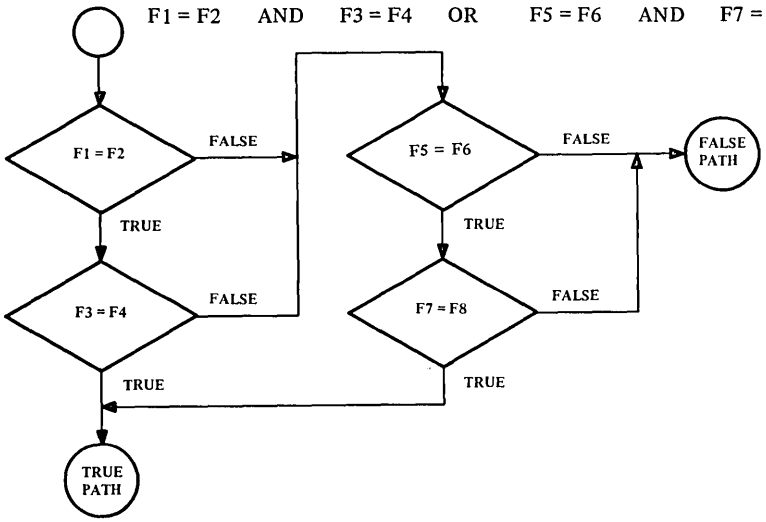
If all the conditions within any of the three sets of parentheses are true, then the entire conditional expression is true.

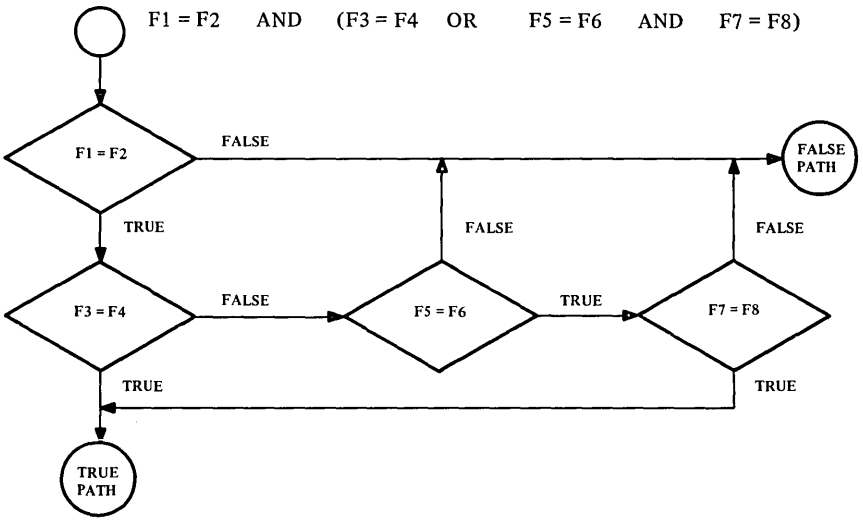
The diagram below illustrates the effect of this statement and the order of evaluation.

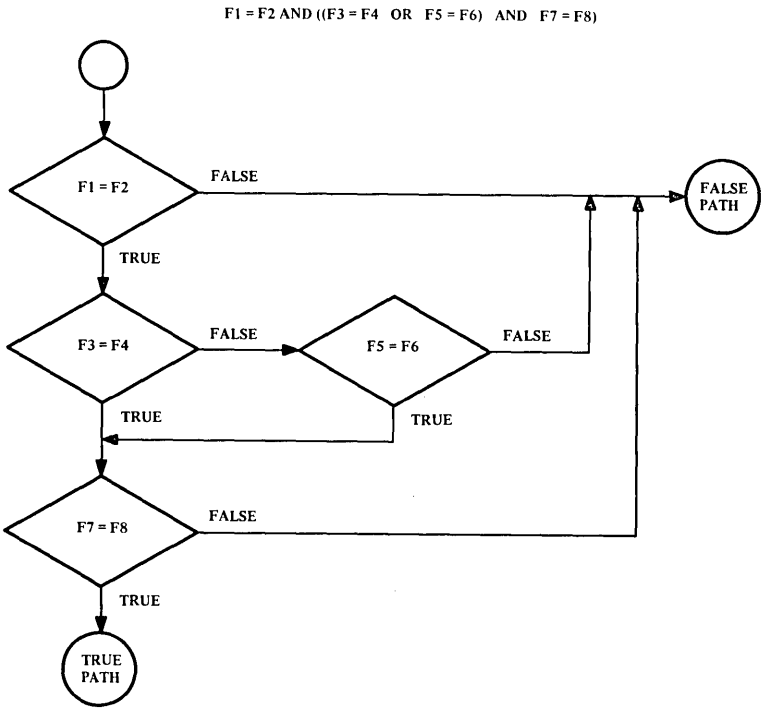


b. Using parentheses to override normal order of evaluation.

To illustrate this usage, a compound-conditional is shown in three forms, each accompanied by a flow diagram showing the result of each.




F1 = F2 AND (F3 = F4 OR F5 = F6 AND F7 = F8)
F1 = F2
FALSE
FALSE PATH
TRUE
FALSE
FALSE
F3 = F4
FALSE
F5 = F6
TRUE
F7 = F8
TRUE
TRUE
TRUE PATH


F1 = F2 AND ((F3 = F4 OR F5 = F6) AND F7 = F8)
F1 = F2
FALSE
FALSE PATH
TRUE
F3 = F4
FALSE
F5 = F6
FALSE
TRUE
TRUE
F7 = F8
FALSE
TRUE
TRUE PATH

6.5.8 Abbreviations in Relation Conditions

When a string of consecutive relation conditions appears in a statement, abbreviations can be used, in certain cases, for any relation condition other than the first. The subject, or the subject and relational operator, or the subject, relational operator and logical connective may be omitted. In each of these cases, the effect of the abbreviated relation condition is as if the omitted parts were the same as those in the nearest preceding complete relation condition within the same sentence. There are two valid forms of abbreviation.

a. Abbreviation 1

If the subject is identical in a series of relational conditions, it can be omitted in all the relational conditions except the first.

Example: A = B OR A < C AND A = D OR A = E

can be abbreviated to

A = B OR < C AND = D OR = E

b. Abbreviation 2

If subjects and relational operators are identical in a series of relational conditions, they can be omitted in all the relational conditions except the first.

Example: A = B OR A = C AND A = D OR A = E

can be abbreviated to

A = B OR C AND D OR E

6.6 COMMON OPTIONS ASSOCIATED WITH THE ARITHMETIC VERBS

Associated with the five arithmetic verbs (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT) are two options: the ROUNDED option, and the ON SIZE ERROR option. These two options are described here to avoid the necessity of including their descriptions with each of the arithmetic verbs.

If the ROUNDED option is specified, the absolute value of the item is increased by 1 if the leftmost truncated digit is greater than 4.

Example:	result:	$567_{\wedge}8756$
	resultant-identifier picture:	999V99
	stored result without ROUNDED option:	$567_{\wedge}87$
	stored result with ROUNDED option:	$567_{\wedge}88$

When the low-order positions in a resultant-identifier are represented by the symbol P in the PICTURE associated with the resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

Example:	result:	5388
	resultant-identifier picture:	99PP
	stored result without ROUNDED option:	53
	stored result with ROUNDED option:	54

6.6.1 The SIZE ERROR Option

If, after decimal point alignment, the number of significant digits in the result of an arithmetic operation is greater than the number of integer positions provided in the result-identifier, a size error condition occurs. Division by zero always causes a size error condition. The size error condition applies to both the intermediate results and the final result of an arithmetic operation. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error does occur, the subsequent action depends upon whether or not the SIZE ERROR option is specified.

If the SIZE ERROR is not specified and a size error condition occurs, the value of the resultant-identifier is unpredictable, and no additional action is taken.

If SIZE ERROR is specified, and a size error condition occurs, then the values of the resultant-identifier(s) affected by the size errors are not altered. Values for resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s). After completion of the execution of the arithmetic operation, the statement(s) after SIZE ERROR is executed.

Example:	ADD A TO B ON SIZE ERROR GO TO OVERFLW	
	A:	954
	B:	PICTURE IS 999; VALUE 954.
	Result:	The contents of B are left unchanged and control is transferred to the paragraph or section named OVERFLW

6.7 THE CORRESPONDING OPTION

The CORRESPONDING option is used in the formats of two of the arithmetic verbs (ADD and SUBTRACT) and in the format of the MOVE verb.

For the purpose of this discussion, d_1 and d_2 represent identifiers that refer to group items. A pair of data items, one from d_1 and one from d_2, correspond if the following conditions exist:

a. A data item in d_1 and a data item in d_2 have the same data-name and the same qualification up to, but not including, d_1 and d_2.

b. Both of the data items are elementary numeric data items in the case of an ADD or SUBTRACT statement with the CORRESPONDING option.

Neither d_1 nor d_2 may be data items with level-number 66, 77, or 88.

Each data item subordinate to d_1 or d_2 that contains a RENAMES, a REDEFINES or an OCCURS clause is ignored. However, d_1 and d_2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

See "ADD," "MOVE," and "SUBTRACT" for information on the specific formats and results of the use of the CORRESPONDING option.

6.8 DETERMINATION OF USAGE IN ARITHMETIC COMPUTATIONS

If a programmer describes a numeric field as having USAGE DISPLAY-6 or DISPLAY-7, the compiler converts this data to fixed-point binary when performing arithmetic computations with it. If the field contains 10 or fewer digits, it is converted to single-precision fixed-point binary. Conversion to double-precision fixed-point binary is performed if the field contains more than 10 digits. A field described as COMPUTATIONAL (or INDEX) is fixed-point binary; single-precision for 10 or fewer digits, double-precision for more than 10 digits. A field described as COMPUTATIONAL-1 is floating-point binary.

When any arithmetic computation is performed, the arithmetic usage (single-precision fixed-point, double-precision fixed-point, or floating-point) used for each operation is determined from the usages of the two operands of the computation. If either operand is floating-point, the operation is performed in floating-point arithmetic. If neither operand is floating-point, but one operand is double-precision fixed-point, the operation is performed in double-precision fixed-point arithmetic. Otherwise, the operation is performed in single-precision fixed-point arithmetic. If both operands are constants, the operation is performed in single- or double-precision fixed-point arithmetic, as appropriate.

If any nonnumeric characters appear in the DISPLAY-6 or DISPLAY-7 field that is to be converted, the compiler attempts to convert them to binary; however, in many cases, undefined results can occur. When DISPLAY-6 and DISPLAY-7 characters are converted to binary, the following rules apply.

0 through 9	are converted to 0 through 9.
A through I	are converted to 1 through 9.
?	is converted to 0.
J through R	are converted to 1 through 9, and the field is made negative if it is found in the low-order digit, unless an explicit sign is present.
:	is converted to 0, and the field is made negative if it is found in the low-order digit, unless an explicit sign is present.
Nulls	are ignored.
Leading spaces and tabs	are ignored.
+ and -	are treated as sign characters.

Scanning of a field proceeds from left to right, it stops when one of the following conditions is met:

a. The entire field has been scanned.
b. A trailing space, tab, plus, or minus is seen.

If both leading and trailing signs appear in the field, the trailing sign will be ignored.

6.9 PROCEDURE DIVISION VERB FORMATS

The format of each PROCEDURE DIVISION verb is given on the following pages. The verbs are presented in alphabetical order.

The word "identifier" is a data-name followed, as required, by any qualification, subscripts, and/or indexes to make the data-name unique.

ACCEPT

Function

The ACCEPT statement causes low-volume data to be read from the user's terminal.

General Format

$$\underline{\text{ACCEPT}}\ \text{identifier-1}\ \left[, \text{identifier-2}\right]\ \ldots\left[\text{FROM mnemonic-name}\right]$$

Technical Notes

a. The ACCEPT statement causes the next set of data available from the terminal to replace the contents of the item named by identifier-1, identifier-2, ...

b. If the FROM option is specified, the mnemonic-name must appear in the CONSOLE IS clause of the SPECIAL-NAMES paragraph.

c. When the data to be read for one or more ACCEPT statements is numeric, comma (,), space, or tab is used as a delimiter separating the data items.

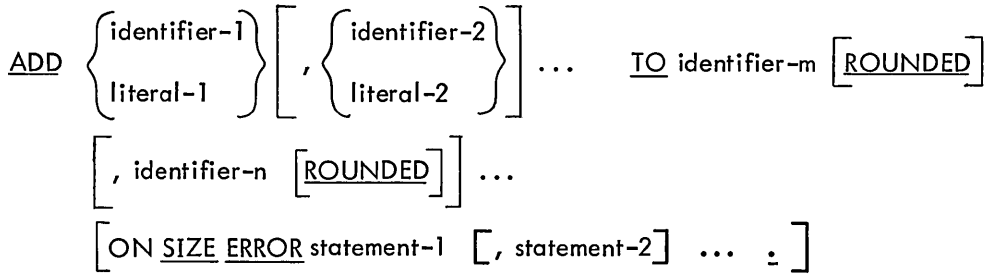
ADD

Function

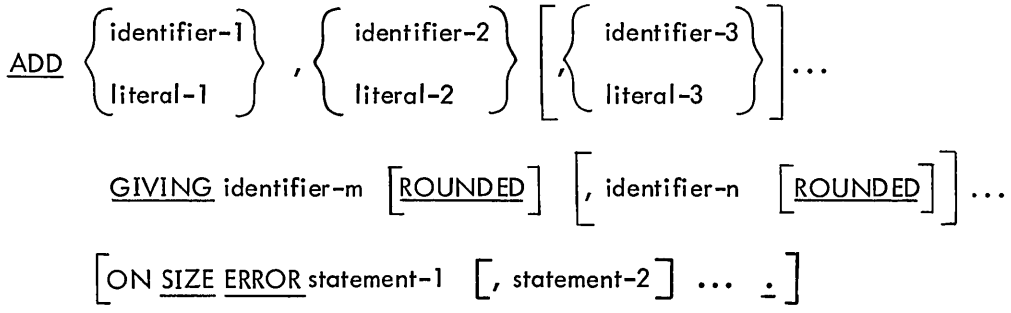
The ADD statement computes the sum of two or more numeric operands and stores the result.

General Format

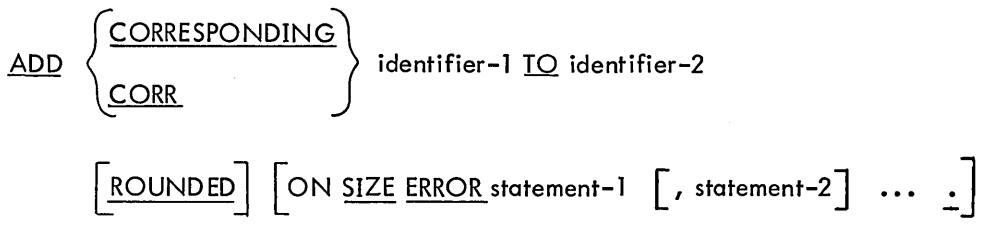
Option 1



Option 2



Option 3



Technical Notes

a. Each ADD statement must contain at least two operands (i.e., an addend and an augend).

In options 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers appearing to the right of the word GIVING may refer to numeric edited items. In option 3, each identifier must refer to a group item.

Each literal must be a numeric literal; the figurative constant ZERO is permitted.

b. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

c. Option 1 causes the values of the operands preceding the word TO to be algebraically summed. The resultant sum is then added to the current value of identifier-m and this result replaces the current value in identifier-m. If other identifiers follow, the same process is repeated for each of them.

d. Option 2 causes the values of the operands preceding the word GIVING to be algebraically summed. The resultant sum then replaces the current contents of identifier-m. If other identifiers follow, their contents are also replaced by this resultant sum. The current values of identifier-m, identifier-n, ... do not enter into the arithmetic computation.

e. Option 3 causes the data items in the group item associated with identifier-1 to be added to the current value of the corresponding data items associated with identifier-2, and each result replaces the value of the corresponding data-items associated with identifier-2. The criteria used to determine whether two items are corresponding are described under "The CORRESPONDING Option" at the beginning of this chapter.

f. The ROUNDED and ON SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

ALTER

Function

The ALTER statement changes the object of one or more GO TO statements.

General Format

```
ALTER   procedure-name-1 TO PROCEED TO procedure-name 2

        [, procedure-name-3 TO PROCEED TO procedure-name-4] ...
```

Technical Notes

a. During execution of the object program, the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ... replacing the object of the GO TO by procedure-name-2, procedure-name-4, ..., respectively.

b. Each procedure-name-1, procedure-name-3, ... must be the name of a paragraph that contains only a single GO TO statement without the DEPENDING option.

c. Each procedure-name-2, procedure-name-4, ... must be the name of a paragraph or section within the PROCEDURE DIVISION.

d. A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority.

e. An ALTER statement in a procedure not in the DECLARATIVES portion of the program may not reference a procedure name within the DECLARATIVES; conversely, an ALTER statement within the DECLARATIVES may not reference a procedure-name not in the DECLARATIVES.

f. Restrictions similar to those in Note e also apply to the INPUT PROCEDURES and to the OUTPUT PROCEDURES associated with sort verbs.

Function

The CALL statement is used to transfer control to a subprogram.

General Format

```
CALL { program-name } [USING identifier-1 [, identifier-2] ...] .
     { entry-name   }
```

Technical Notes

a. Program-name is a 1-to-6 character name (PROGRAM-ID) of the subprogram to be called. Entry-name is a 1-to-6 character name of an entry point in the subprogram. Either name can be enclosed in quotation marks, but can contain only letters and digits.

b. If the program-name is used, the entry point will be at the beginning of the executable code in the subprogram.

c. Called programs can call other subprograms, but a called program cannot call, either directly or indirectly, any part of itself or the program that called it.

d. The number of operands in each USING clause of the CALL statement must be greater than or equal to the number of operands in the ENTRY statement or PROCEDURE DIVISION header in the subprogram.

e. Each of the operands in the USING clause may be any item defined in the FILE, WORKING-STORAGE, or LINKAGE SECTION of the calling program. However, these items must be word-aligned, i.e., they must begin on a word boundary. 01- and 77-level items are always word-aligned. Any other item can be word-aligned by means of the SYNCHRONIZED LEFT clause.

f. The identifiers in the USING clause indicate those data items in the calling program that may be referenced (or whose subordinate parts may be referenced) in the called program. The order of the identifiers in the CALL statement in the calling program and in the PROCEDURE DIVISION header or ENTRY statement of the called program is critical. The items in the USING clause are related by their corresponding positions, not by name. Corresponding identifiers refer to a single set of data that is available to both the calling and called programs.

(continued on next page)

g. The first time a called program is entered, its state is that of a fresh copy. Subsequently, unless the program has been cancelled, its state when entered is exactly as it was left after the last exit from it. That is, all internal variables, altered GO TO's, and the like are exactly as they were left. However, external data (i.e., data described in the LINKAGE SECTION) may have been changed since the last exit.

h. Refer to Section 8.11 for more information on subprograms.

Function

The CLOSE statement terminates the processing of input and output files, reels, or units.

General Format

```
CLOSE file-name [ { REEL } ]  [ WITH { NO REWIND } ]
                { UNIT }              { LOCK      }
                                      { DELETE    }

[ , file-name-1 [ { REEL } ]  [ WITH { NO REWIND } ] ] ...
                { UNIT }             { LOCK      }
                                     { DELETE    }
```

Technical Notes

a. Each file-name must appear as the subject of an FD entry in the FILE SECTION of the DATA DIVISION.

b. The REEL, UNIT, and NO REWIND options apply only to magnetic tape files. UNIT is synonymous with REEL.

c. The DELETE option applies only to disk and DECtape files. If this option is included, the file will be deleted from the device.

d. For the purpose of showing the effect of various CLOSE options as applied to the various storage media, all input, output, and input-output files are divided into the following three mutually exclusive categories:

(1)	NON-REEL	A file whose device is such that the concepts of REWIND, REEL, or UNIT have no meaning. This category includes files residing on disk, punched cards, paper tape, line printer, and Teletype.
(2)	SINGLE-REEL	A file that is entirely contained on one reel or unit.
(3)	MULTI-REEL	A file that may be contained on more than one reel or unit.

The results of each CLOSE option for each of the above types of files are summarized in Table 6-3. The definitions for the symbols used in this table are given below. Where the definition depends upon whether the file is an input or output file, alternate definitions are given; otherwise, the single definition given applies to both input and output files.

A Any subsequent reels of this file will not be processed.

B The current reel is not rewound.

C Standard CLOSE File Procedure

INPUT and I-O Files (see "OPEN")

If the file is positioned at its end, the user's ENDING FILE LABEL PROCEDUREs are performed, if the user has specified any via a USE statement. An input file is considered to be at the end-of-file if the imperative-statement in the AT END clause of a READ for the file has been executed, and no CLOSE statement for the file has been executed.

OUTPUT Files

If LABEL RECORDS are STANDARD, an ending label is created and written on the output medium. Then, any user ENDING FILE LABEL PROCEDUREs are performed.

D The current reel is rewound and unloaded.

E Any attempt to subsequently OPEN this file will result in an error message being typed and the run terminated.

F Standard CLOSE REEL Procedure

INPUT Files

(1) If the file is assigned to more than one device, the next device specified in the ASSIGN clause becomes the current device. If no other device is specified, the first device mentioned becomes the current device.

(2) The standard beginning reel label procedure and the user's BEGINNING REEL LABEL PROCEDURE (specified in a USE statement) are performed for the new reel.

OUTPUT and I-O Files

(1) The standard ending reel label procedure and any user's ENDING REEL LABEL PROCEDURE are performed.

(2) If the file is assigned to more than one device, the devices are swapped. A halt occurs to allow the user to mount an available reel.

(3) The standard beginning reel label procedure and any user's BEGINNING REEL LABEL PROCEDURE are performed.

G The tape is rewound.

H The file is deleted from the device.

X Illegal. This is an illegal combination of a CLOSE option and a file type.

Table 6-3
CLOSE Options and File Types

CLOSE Option	File Type: NON-REEL	File Type: SINGLE REEL/UNIT	File Type: MULTI-REEL
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL	X	X	F,G
CLOSE REEL WITH LOCK	X	X	F,D
CLOSE REEL WITH NO REWIND	X	X	F,B
CLOSE WITH DELETE	C,H	X	X

e. If a file is OPENed but not CLOSEd before the STOP RUN statement is executed, the file will be automatically CLOSEd.

f. If the file has been specified with an OPTIONAL clause in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION and the file was not present for this run, the CLOSE has no effect.

g. If a CLOSE statement without the REEL or UNIT option has been executed for a file, a READ, WRITE, or CLOSE statement for that file must not be executed until another OPEN for that file has been executed.

COMPUTE

Function

The COMPUTE statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression.

General Format

```
COMPUTE identifier-1 [ROUNDED] { EQUALS  } { identifier-2           }
                               { EQUAL TO } { literal-1              }
                               { =        } { arithmetic-expression  }

[ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

Technical Notes

a. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. If the composite operand exceeds 19 decimal digits, the composite is converted to COMP-1 format.

b. Identifier-1 must be an elementary numeric or numeric edited item.

c. Identifier-2 must be an elementary numeric item. Literal-1 must be a numeric literal.

The identifier-2 and literal-1 options provide a method for setting the value of identifier-1 equal to identifier-2 or literal-1.

d. The rules for forming arithmetic expressions and the order of evaluation are given earlier in this chapter under "Arithmetic Expressions."

e. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with the Arithmetic Verbs".

DELETE

Function

The DELETE statement removes a specified record from a file whose access mode is INDEXED.

General Format

DELETE record-name INVALID KEY statement-1 [, statement-2] ... .

Technical Notes

a. Record-name must be a record associated with a file whose access mode is INDEXED.

b. When the DELETE statement is executed, the record in the file that has a key equal in value to the SYMBOLIC key for the file is removed from the file. If no such record exists, the statement(s) associated with the INVALID KEY clause is executed.

c. At the time that the DELETE statement is executed, the file must be open for OUTPUT or INPUT-OUTPUT.

DISPLAY

Function

The DISPLAY statement causes low-volume data to be written on the user's Teletype console.

General Format

```
DISPLAY { literal-1    } [ , { literal-2    } ] ...
        { identifier-1 }     { identifier-2 }

[ UPON mnemonic-name ]  [ WITH NO ADVANCING ]
```

Technical Notes

a. The contents of each operand are written on the user's Teletype console in the order listed.

b. Each of the literals can be numeric or nonnumeric, or one of the figurative constants. If a figurative constant is specified as one of the operands, only a single occurrence of that constant is written on the device.

c. The mnemonic-name must appear in the CONSOLE clause in the SPECIAL-NAMES paragraph of the Environment Division.

d. If WITH NO ADVANCING is specified, the Teletype does not advance to the next line. Thus, printing or type-in can continue on the same line.

DIVIDE

Function

The DIVIDE statement divides one numeric item into another and sets the value of a data item equal to the result.

General Format

Option 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED] [REMAINDER identifier-4]
       { literal-1    }

    [ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

Option 2

```
DIVIDE { identifier-2 } BY identifier-1 [ROUNDED] [REMAINDER identifier-4]
       { literal-2    }

    [ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

Option 3

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3
       { literal-1    }     { literal-2    }

    [ROUNDED] [REMAINDER identifier-4]
    [ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

Option 4

```
DIVIDE { identifier-2 } BY { identifier-1 } GIVING identifier-3
       { literal-2    }   { literal-1    }

    [ROUNDED] [REMAINDER identifier-4]
    [ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

a. The value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2.

In option 1, the resulting quotient replaces the value of identifier-2. In option 2, the resulting quotient replaces the value of identifier-1. In options 3 and 4, the resulting quotient replaces the value of identifier-3.

b. Each DIVIDE statement must contain two operands (i.e., a dividend and a divisor). Both of these operands (identifier-1 and identifier-2) must refer to elementary numeric items. Identifier-3 may be an elementary numeric or numeric edited item. Each literal-1 or literal-2 must be a numeric literal.

c. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

d. If the REMAINDER clause is used, the resulting remainder replaces the value of identifier-4.

e. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

Function

The ENTER statement allows the execution of MACRO, FORTRAN-10, and FORTRAN IV subroutines in conjunction with the COBOL program.

General Format

```
ENTER { MACRO      }
      { FORTRAN-IV } program-name
      { FORTRAN    }
      { COBOL      }

[ USING { identifier-1   } [ , [ { identifier-2   } ] ] ... ]
        { literal-1      }       { literal-2      }
        { procedure-name-1 }     { procedure-name-2 }
```

Technical Notes

a. MACRO refers to MACRO assembly language, FORTRAN-IV to the old DECsystem-10 FORTRAN language, and FORTRAN to the new DECsystem-10 FORTRAN-10 language.

b. The program-name can be enclosed in quotation marks.

c. The ENTER statement generates a subroutine call followed by the address in which the items associated with the USING clause are located. The ENTER statement is discussed further in Appendix C.

d. ENTER COBOL is equivalent to CALL.

ENTRY

Function

The ENTRY statement establishes an entry point in a subprogram.

General Format

```
ENTRY entry-name [ USING identifier-1 [ , identifier-2 ] ... ] .
```

Technical Notes

a. The ENTRY statement can only be used in a subprogram.

b. Control is passed to the entry point by a CALL statement in a calling program.

c. Entry-name is a 1-to-6 character name that can contain only letters and digits. It can, however, be enclosed in quotation marks. This name must not be the same as any other entry-name or PROGRAM-ID in any program with which the subprogram containing it is loaded.

d. The identifiers listed in the USING clause must be defined as 01- or 77-level items in the LINKAGE SECTION of the subprogram containing the ENTRY statement.

e. The number of operands in the USING clause of an ENTRY statement must be less than or equal to the number of operands in any CALL statement referencing that ENTRY statement.

f. The identifiers in the USING clause indicate those data items in the called program that may reference data items in the calling program. The order of identifiers in the CALL statement in the calling program and in the ENTRY statement in the called program is critical. The items in the USING clauses are related by their corresponding positions, not by name. Corresponding identifiers refer to a single set of data that is available to both the calling and called programs.

g. At run-time, ENTRY statements are ignored unless there are specific calls to them.

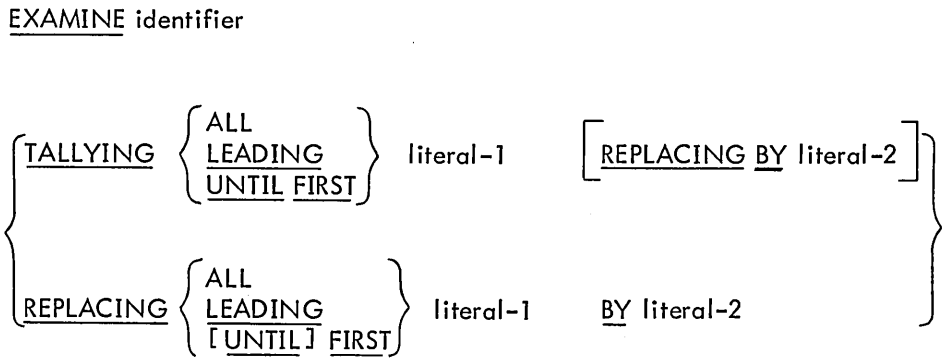
h. Refer to Section 8.11 for more information on subprograms.

EXAMINE

Function

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

General Format



Technical Notes

a. The USAGE of identifier must be DISPLAY or DISPLAY-7, implicitly or explicitly.

b. Each literal must consist of a single character belonging to a class consistent with that of the identifier. A literal may be any figurative constant.

c. Examination starts at the leftmost character of the identifier and proceeds to the right.

d. When the TALLYING option is used, a count is kept of

(1) Occurrences of literal-1 when the ALL option is used.

(2) Occurrences of literal-1 prior to a character other than literal-1 when the LEADING option is used.

(3) Characters prior to the first occurrence of literal-1 when the UNTIL FIRST option is used.

This count replaces the contents of the special register called TALLY (see "Special Registers," Chapter 1). TALLY has a PICTURE of S99999, and can be referenced in any statement where an identifier referring to an elementary numeric data item is valid.

If the REPLACING BY clause is used with the TALLYING option, replacement is performed according to the rules below.

e. When either of the REPLACING BY options are used, replacement rules are

(1) If the ALL option is used, literal-2 is substituted for each occurrence of literal-1.

(2) If the LEADING option is used, the substitution of literal-2 for literal-1 terminates as soon as a character other than literal-1 is encountered.

(3) If the UNTIL FIRST option is used, literal-2 is substituted for each character prior to the first occurrence of literal-1.

(4) If the FIRST option is used, literal-2 is substituted for only the first occurrence of literal-1.

f. If the identifier is classified as numeric, it must consist solely of numeric characters. It may possess an operational sign, but this sign is ignored by the EXAMINE process.

EXIT

Function

The EXIT statement provides a common end point for a series of routines executed by a PERFORM or USE statement.

General Format

```
paragraph-name. EXIT.
```

Technical Notes

a. EXIT must be the first sentence in a paragraph. Only NOTE may follow.

b. The EXIT statement may be used to provide an end point for a series of paragraphs that are PERFORMed, or at the end of a section in the DECLARATIVES. By using EXIT at the end of the range of a PERFORM or USE, a variety of exits from the performed procedure can be accomplished by making each point at which an exit is required a transfer to the EXIT paragraph. However, unless EXIT is specified as the end of the range of a PERFORM or USE or is placed as the last paragraph in the range of a PERFORM or USE, it is ignored.

Example:

```
PERFORM TAX-ROUTINE THROUGH EXIT-RTE.
.
.
.
TAX-ROUTINE.
    IF TOTAL-TAX IS EQUAL TO OR GREATER THAN TAX-LIMIT
    GO TO EXIT-RTE.
    MULTIPLY .....
    .
    .
DEDUCTION-RTE.
    IF NO-OF-DEPENDENTS IS EQUAL TO ZERO
    GO TO EXIT-RTE.
    MULTIPLY NO-OF-DEPENDENTS BY DEP-DEDUCT....
    .
    .
EXIT-RTE. EXIT.
```

c. If control reaches an EXIT statement and no associated PERFORM or USE statement is active or if EXIT is not the last paragraph in the range of a PERFORM or USE statement even if the PERFORM or USE statement is active, control passes through the EXIT paragraph to the first statement of the next paragraph.

EXIT PROGRAM

Function

The EXIT PROGRAM statement is used to return control from a subprogram to its calling program.

General Format

EXIT PROGRAM.

Technical Notes

a. EXIT PROGRAM can only appear in a subprogram.

b. When an EXIT PROGRAM statement is executed, control is returned to the calling program at the statement immediately following the CALL statement.

c. If an EXIT PROGRAM statement is encountered in a subprogram that is operating as a main program, it is ignored.

d. Refer to Section 8.11 for more information on subprograms.

Function

The GENERATE statement causes the Report-Writer to execute all automatic report operations, and, if required, produce one or more report group.

General Format

GENERATE identifier

Technical Notes

a. If identifier is the name of a TYPE DETAIL report group, the GENERATE statement performs all the automatic report operations, and produces an output detail report group on the output file. This is called detailed reporting.

b. If the identifier is the name of an RD entry, the GENERATE statement performs all the automatic report operations, but does not produce an output detail report group. This is called summary reporting.

c. A GENERATE statement performs the following automatic operations:

(1) Steps and tests the LINE-COUNTER and/or PAGE-COUNTER to produce, if necessary, any PAGE FOOTING and PAGE HEADING report groups.

(2) Recognizes any specified control breaks to produce appropriate CONTROL FOOTING and CONTROL HEADING report groups, and resets appropriate summation counters.

(3) Accumulates into the summation counters all specified identifiers.

(4) Executes any routines defined by a USE statement.

(5) If this is detailed reporting, produces the detailed report group.

d. During the execution of the first GENERATE statement for a report, the following groups, if specified, are produced:

(1) Report Heading
(2) Page Heading
(3) All Control Headings, in the order major to minor.
(4) The detail report group, if this is detailed reporting.

e. Data is moved to the data item in the Report Group Description Entry according to the same rules for movement described for the MOVE statement.

f. A GENERATE statement for a particular report may not be executed until an INITIATE statement has been executed for that report. In addition, if a TERMINATE statement has been executed for that report, a GENERATE statement may not be executed until an intervening INITIATE statement is executed for the report.

g. Refer to "REPORT EXAMPLE" in Section 5.7 for an example of a report-writing program.

GO

Function

The GO TO statement causes control to be transferred from one part of the PROCEDURE DIVISION to another.

General Format

Option 1

```
GO TO [ procedure-name-1 ]
```

Option 2

```
GO TO procedure-name-1, procedure-name-2 [ , procedure-name-3 ] ...
    DEPENDING ON identifier
```

Technical Notes

a. Each procedure-name is the name of a paragraph or section in the PROCEDURE DIVISION of the program.

b. Option 1 causes transfer of control to the specified procedure-name, or to some other procedure-name if the GO TO has been previously ALTERed.

In order to be alterable, Option 1 must appear as the first sentence in a paragraph; only NOTE may follow.

If procedure-name-1 is not specified, the GO TO must be alterable and an associated ALTER statement must be executed prior to executing this GO TO.

When this form of GO TO appears in an imperative sentence, it must appear as the last or only statement in the sentence, except for NOTE.

c. Option 2 causes transfer of control to procedure-name-1, procedure-name-2, ... or procedure-name-n depending on whether the value of the identifier is 1, 2, ... or n, respectively.

The identifier must refer to an elementary numeric item having no positions to the right of the decimal point. The item may not be USAGE COMPUTATIONAL-1.

If the value of the identifier is other than the positive integers 1, 2, ... or n, the GO TO statement is by-passed.

GOBACK

Function

The GOBACK statement is used in a subprogram to return control to the calling program.

General Format

GOBACK .

Technical Notes

a. The GOBACK statement can only be used in subprograms.

b. When control reaches a GOBACK statement, control is returned to the calling program at the statement immediately following the CALL statement.

c. If a GOBACK statement is encountered in a subprogram that is operating as a main program, it is treated as a STOP RUN statement.

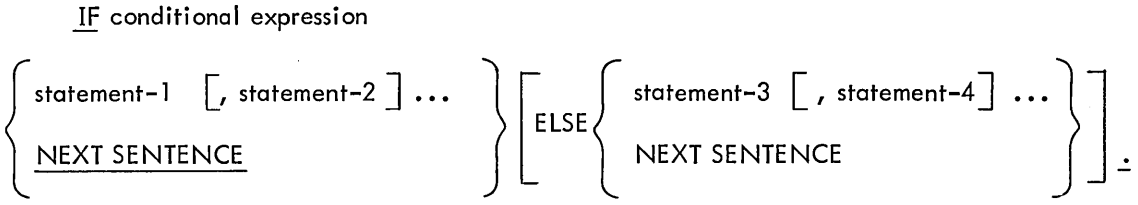
d. Refer to Section 8.11 for more information on subprograms.

IF

Function

The IF statement causes a conditional expression to be evaluated and the subsequent operations to be performed to be determined as a result of this evaluation.

General Format



a. Conditional expressions are discussed in Paragraph 6.5 in this chapter.

b. The subsequent action of the program is determined by whether the conditional expression is true or false.

(1) If the conditional expression is true and statement-1 and any following statements are given, statement-1 and any following statements are executed and, provided that they do not contain a GO TO or STOP RUN, control passes to the next sentence.

If the conditional expression is true and NEXT SENTENCE is given, control passes to the next sentence.

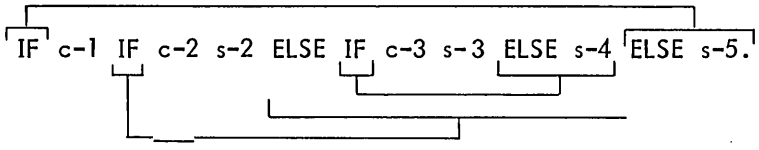
(2) If the conditional expression is false and statement-3 and any following statements are given, statement-3 and any following statements are executed and, provided that they do not contain a GO TO or STOP RUN, control passes to the next sentence.

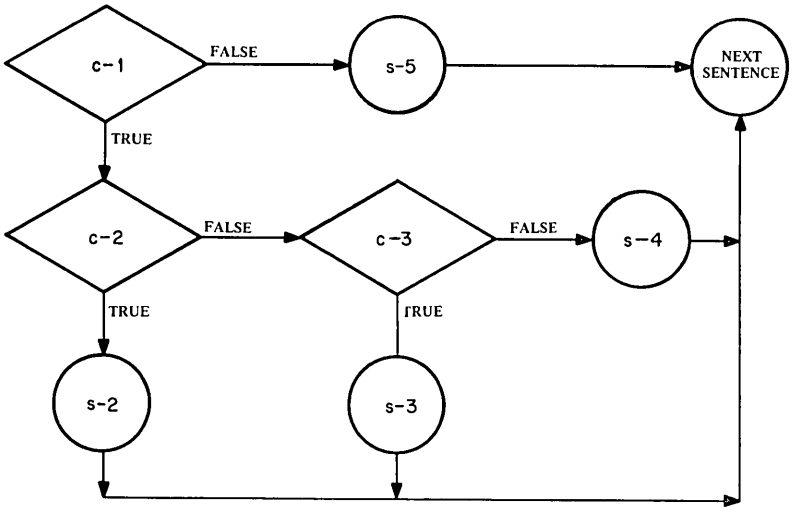
If the conditional expression is false and either ELSE NEXT SENTENCE is given or the entire ELSE clause is omitted, control passes to the next sentence.

c. Statement-1, statement-2, statement-3, and statement-4 may be any statement or sequence of statements.

Any statement may contain another IF statement; in this case, this second IF statement is said to be nested. Nested IF statements may be considered paired IF and ELSE combinations. Each subsequent ELSE encountered is considered to apply to the nearest preceding IF that has not already been paired with an ELSE. In other words, the pairing process begins with the innermost nested IF...ELSE pair and works outward.

Example: (c = condition; s = statement)





INITIATE

Function

The INITIATE statement is used to initialize all counters before a report is produced.

General Format

```
INITIATE report-name-1 [,report-name-2] ...
```

Technical Notes

a. Each report-name must be defined by an RD entry in the REPORT SECTION of the DATA DIVISION.

b. The INITIATE statement resets all data-name entries that contain SUM clauses associated with a report.

c. The PAGE-COUNTER is set to 1 during the execution of an INITIATE statement. If a different starting value for the PAGE-COUNTER is desired, it may be reset following the INITIATE statement before the execution of the first GENERATE statement.

d. The LINE-COUNTER is set to 0 during execution of the INITIATE statement.

e. The INITIATE statement does not open the file with which the report is associated. An OPEN statement must be executed prior to the execution of the INITIATE statement.

f. A second INITIATE statement for a particular report-name may not be executed until a TERMINATE statement for that report-name is executed.

g. An example of a report-writing program is shown in Section 5.7 in Chapter 5.

MOVE

Function

The MOVE statement transfers data in accordance with the rules of editing, from one data area to one or more data areas.

General Format

Option 1

```
MOVE { identifier-1 } TO identifier-2 [, identifier-3] ...
     { literal-1    }
```

Option 2

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

Technical Notes

a. Identifier-1 (or literal-1) represents the data to be moved and is called the sending item. Identifier-2, identifier-3, ... represent the receiving data items.

b. In option 1, the data contained in identifier-1 or literal-1 is moved first to identifier-2, then identifier-3, etc.

In option 2, data items within the group item associated with identifier-1 are moved to corresponding data items within the group item associated with identifier-2. The results are the same as if the user had referred to each pair of corresponding identifiers in separate MOVE statements. The criteria used to determine whether two items are corresponding are described under "The CORRESPONDING Option" at the beginning of this chapter.

c. The following rules apply to both group and elementary items; a group item is treated as a single field.

(1) A numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

(2) A numeric or numeric edited item must not be moved to an alphabetic data item.

(3) A numeric item whose implicit decimal point is not immediately to the right of the least significant digit must not be moved to an alphanumeric or alphanumeric edited item.

All other moves are legal.

d. The following rules apply to legal moves.

(1) When an alphanumeric, alphanumeric edited, or alphabetic item is the receiving item,

(a) If the size of the sending field is greater than the size of the receiving field, the least significant (rightmost) characters are truncated if the receiving field is not described by a JUSTIFIED RIGHT clause; the most significant (leftmost) characters are truncated if the receiving field is described as JUSTIFIED RIGHT.

(b) If the size of the sending field is less than the size of the receiving field, spaces are placed in the remaining rightmost characters of the receiving field if the receiving field is not described by a JUSTIFIED RIGHT clause; spaces are placed in the remaining leftmost characters of the receiving field if the receiving field is described by a JUSTIFIED RIGHT clause.

(c) If the sizes of the sending and receiving fields are equal, no truncation or filling with spaces takes place.

(2) When a numeric or numeric edited item is the receiving item, the sending and receiving fields are aligned by decimal point. If the sending field is not numeric, the decimal point is assumed to be on the right. Any necessary zero filling takes place before editing. If the receiving item has no operational sign, the absolute value of the sending item is stored. If the receiving item has fewer digits to the left or right of the decimal point than does the sending item, the excess digits are truncated. If the sending item contains any nonnumeric characters, the result is unpredictable.

(3) Any necessary conversion of data from one form of internal representation to another is performed automatically during the move, along with any editing specified by the PICTURE of the receiving item.

e. Any move that is not an elementary move (that is, both the sending and receiving items are not elementary items) is called a group move. A group move is treated 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 other words, the individual data descriptions of the items within the sending group item and the receiving group item are completely ignored and both items are treated as though they were described by a PICTURE IS X(n) clause, where n is the number of character positions in the particular item.

MULTIPLY

Function

The MULTIPLY statement causes one data item to be multiplied by another data item and the resulting product to be stored in a data item.

General Format

Option 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }

    [ ON SIZE ERROR statement-1 [, statement-2 ] ... . ]
```

Option 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3
         { literal-1    }    { literal-2    }

    [ROUNDED] [ ON SIZE ERROR statement-1 [, statement-2] ... . ]
```

Technical Notes

a. Each MULTIPLY statement must contain at least two operands (a multiplicand and a multiplier). Each identifier must refer to an elementary numeric item, except that identifier-3 may refer to either a numeric or a numeric edited item. Each literal must be a numeric literal; the figurative constants ZERO and TALLY are permitted.

b. Option 1 causes the value of identifier-1 or literal-1 to be multiplied by the value of identifier-2. The resultant product replaces the value of identifier-2.

c. Option 2 causes the value of identifier-1 or literal-1 to be multiplied by the value of identifier-2 or literal-2. The resultant product replaces the value of identifier-3.

d. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

e. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

NOTE

Function

The NOTE statement allows the programmer to insert comments in the PROCEDURE DIVISION of his program.

General Format

NOTE character-string .

Technical Notes

a. Any combination of characters from the ASCII character set may be included in the character-string.

b. If the NOTE sentence appears as the first sentence in a paragraph, the entire paragraph is considered to be part of the character-string. The paragraph is ended when a new paragraph is begun. A new paragraph has its first word starting in Area A.

c. If the NOTE statement appears as other than the first sentence in a paragraph, the character-string ends at the first period followed by a space or carriage return.

d. The contents of the character-string appear on the compilation listing, but are not compiled.

OPEN

Function

The OPEN statement initiates the processing of files and, where necessary, performs the checking and writing of labels.

General Format

```
OPEN { { INPUT  } file-name-1 [WITH NO REWIND] [, file-name-2 [WITH NO REWIND]] ... }...
     { { OUTPUT }                                                                   }
     { { I-O          } file-name-3 [, file-name-4] ...                             }
     { { INPUT-OUTPUT }                                                             }
```

Technical Notes

a. The OPEN statement must be executed for a file prior to the execution of any SEEK, READ, WRITE, or CLOSE for that file.

b. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

c. An OPEN statement does not obtain or release the first record of a file. A READ statement must be executed to obtain the first record (or a WRITE statement must be executed to release the first record).

d. Up to 16 files can be opened at a time. If the program is segmented, one less file can be open; similarly, if the RERUN option is being used, one less file can be open. When indexed sequential files are being used, each indexed sequential file is treated as two files: the index file and the data file. The key word INPUT, OUTPUT, INPUT-OUTPUT, or I-O applies to each subsequent file-name until another such key word is encountered or until the end of the OPEN statement is reached.

e. The NO REWIND option has meaning only for magtape files and is ignored for all other devices. If the NO REWIND clause is not specified for a tape file, the tape is rewound to the beginning of tape.

f. If labels exist, the label is read into the record area to make it available to the USE routines. The record area is then filled with spaces. If a file has been described as LABEL RECORDS ARE STANDARD, standard label checking or label writing is performed; the user's BEGINNING LABEL (USE) routines are executed if provided. If a file has been described as LABEL RECORDS ARE data-name-1, the user's BEGINNING LABEL (USE) routines are executed. If a file has been described as LABEL RECORDS ARE OMITTED, no label checking or writing is performed.

g. If an INPUT file is described as OPTIONAL (in the FILE-CONTROL paragraph), the operating system will type the message

IS file-name PRESENT?

and wait for the user to type "YES" or "NO". If the user types "NO", the first READ statement for this file causes the imperative-statement at the AT END or INVALID KEY clause to be executed.

h. The I-O or INPUT-OUTPUT options permit the opening of a file on a random-access device for both input and output processing. When the I-O option is specified, the execution of the OPEN statement causes the standard beginning label procedures (see Chapter 8) and the user's BEGINNING LABEL routines, if specified by a USE statement, to be executed. If the file does not exist when it is opened for INPUT-OUTPUT, an empty file is created.

PERFORM

Function

The PERFORM statement is used to depart from the normal sequence of execution in order to execute one or more procedures and then return control to the normal sequence.

General Format

Option 1

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

Option 2

```
PERFORM procedure-name-1 [THRU procedure-name-2]
        {identifier-1} TIMES
        {integer-1   }
```

Option 3

```
PERFORM procedure-name-1 [THRU procedure-name-2]
        UNTIL condition-1
```

Option 4

```
PERFORM procedure-name-1 [THRU procedure-name-2]

        VARYING identifier-1 FROM {literal-1   }
                                  {identifier-2}

        BY {literal-2   } UNTIL condition-1
           {identifier-3}

      [ AFTER VARYING identifier-4 FROM {literal-3   }
                                        {identifier-5}
```

BY {literal-4 / identifier-6} UNTIL condition-2

[AFTER VARYING identifier-7 FROM {literal-5 / identifier-8}

BY {literal-6 / identifier-9} UNTIL condition-3]]

Technical Notes

a. Each procedure-name is the name of a section or paragraph in the PROCEDURE DIVISION. Each identifier must refer to a numeric elementary item described in the DATA DIVISION. Each literal must be a numeric literal or the figurative constants ZERO and TALLY.

b. When the PERFORM statement is executed, control is transferred to the first statement of procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows. The procedures executed constitute the range of the PERFORM.

(1) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, the return is after the last statement of procedure-name-1.

(2) If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is after the last statement in the last paragraph in procedure-name-1.

(3) If procedure-name-2 is a paragraph-name, the return is after the last statement in that paragraph.

(4) If procedure-name-2 is a section-name, the return is after the last statement in the last paragraph of that section.

c. There is no relationship between procedure-name-1 and procedure-name-2, except that the sequence of operations beginning at procedure-name-1 must eventually end with the execution of procedure-name-2 in order to effect the return at the end of procedure-name-2. Any number of GO TO and/or PERFORM statements may occur between procedure-name-1 and procedure-name-2.

d. If control passes to these procedures by means other than a PERFORM statement, control passes through the return point to the following statement as though no return mechanism were present.

e. No perform statement may terminate until all PERFORM statements that it has executed have terminated. No PERFORM statement may be executed which terminates at the same procedure-name as another active PERFORM.

f. Option 1 causes the PERFORM range to be executed once, followed by a return to the statement immediately following the PERFORM.

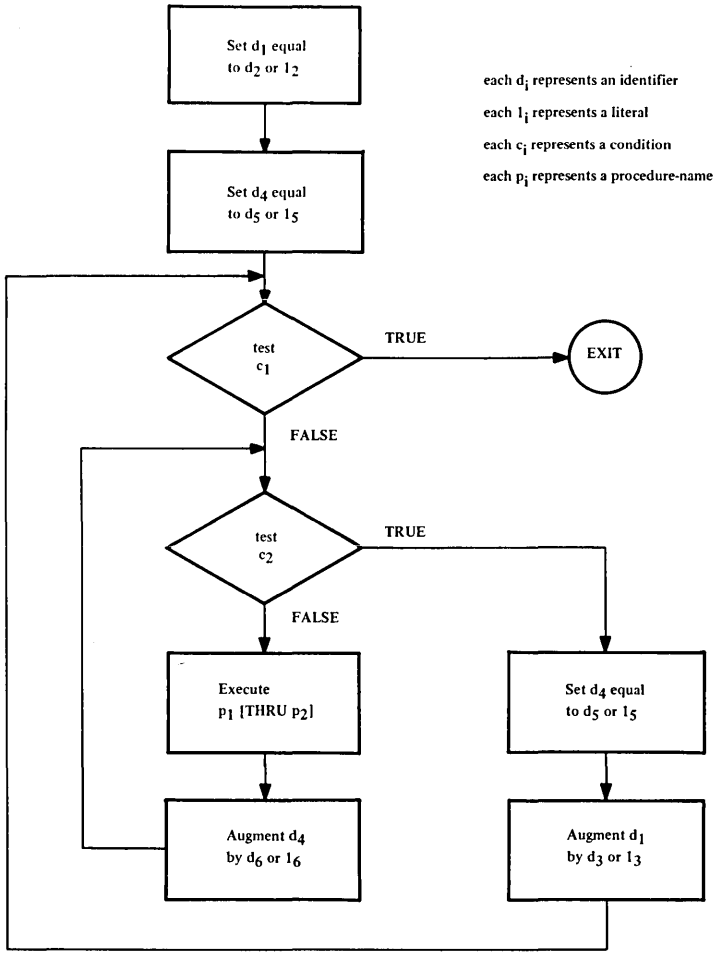
g. Option 2 causes the PERFORM range to be executed the number of times specified by identifier-1 or integer-1. The value of identifier-1 or integer-1 must not be negative; it may be zero. Once the PERFORM statement has been initialized, any modification to the contents of identifier-1 has no effect on the number of times the range is executed.

h. Option 3 causes the PERFORM range to be executed until the condition specified in the UNTIL clause is true. If this condition is true at the time the PERFORM statement is initialized, the range is not executed. Conditions are explained under "Conditional Expressions" earlier in this chapter.

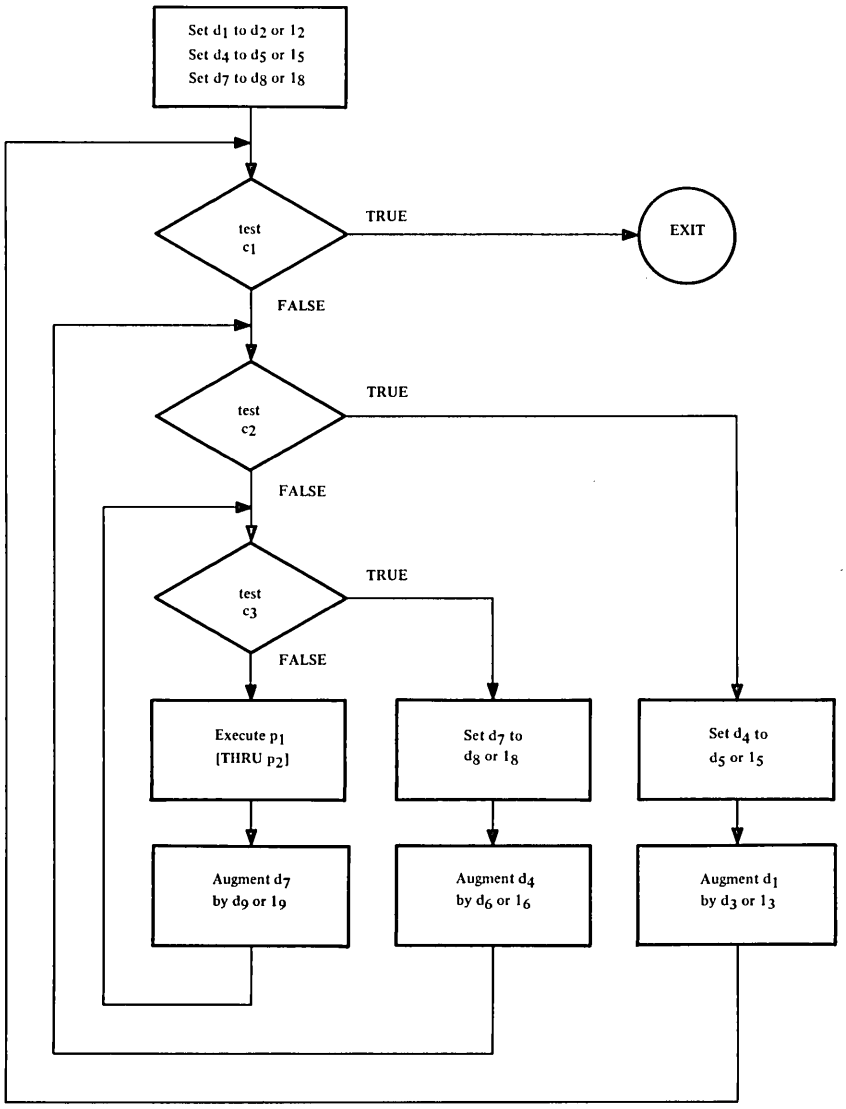
i. Option 4 is used to augment the value of one or more identifiers during the execution of a PERFORM statement.

In option 4, when only one identifier is varied, identifier-1 is set equal to identifier-2 or literal-2 when the PERFORM statement is initialized. If the condition specified is determined to be false at this point, the PERFORM range is executed once. Then the value of identifier-1 is augmented by identifier-3 or literal-3 and the test of the condition is done again. This cycle continues until condition-1 is true; at this point, control passes to the statement following the PERFORM statement. If condition-1 is true at the beginning of the execution of the PERFORM, control immediately passes to the statement following the PERFORM.

The flow chart below illustrates the logic of the PERFORM cycle when two identifiers are varied.


Set d_1 equal to d_2 or l_2
Set d_4 equal to d_5 or l_5
test c_1
TRUE
EXIT
FALSE
test c_2
TRUE
FALSE
Execute p_1 [THRU p_2]
Set d_4 equal to d_5 or l_5
Augment d_4 by d_6 or l_6
Augment d_1 by d_3 or l_3
each d_i represents an identifier
each l_i represents a literal
each c_i represents a condition
each p_i represents a procedure-name

The following flow chart illustrates the logic of the PERFORM cycle when three identifiers are varied.



j. A PERFORM statement that appears in a section whose priority is less than the SEGMENT-LIMIT can have in its range only those procedures contained in sections

(1) Each of which has a priority number less than 50, or

(2) Wholly contained in a single segment whose priority number is greater than 49.

A PERFORM statement that appears in a section whose priority number is equal to or greater than SEGMENT LIMIT, can have in its range only those procedures contained in sections

(1) Each of which has the same priority number as that containing the PERFORM statement, or

(2) Each of which has a priority that is less than the SEGMENT-LIMIT.

When a procedure-name in a segment with a priority number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority number, the segment referred to is made available in its initial state (that is, with all ALTERable GO TOs set to their initial setting) for each execution of the PERFORM statement.

k. A PERFORM statement in a section not in the DECLARATIVES may have as its range, procedures wholly contained within the DECLARATIVES; however, a PERFORM statement in a section within the DECLARATIVES may not have any non-DECLARATIVE procedures within its range.

l. A PERFORM statement within an INPUT or OUTPUT PROCEDURE associated with a SORT verb may not have within its range any procedures outside of that INPUT or OUTPUT procedure.

READ

Function

The READ statement makes available a logical record from an input file and allows performance of a specified imperative statement when end-of-file or invalid key is detected.

General Format

```
READ file-name RECORD
[ INTO identifier ]  { AT END / INVALID KEY }  statement-1 [ , statement-2] ... .
```

Technical Notes

a. An OPEN INPUT or OPEN I-O statement must be executed for the file prior to execution of the first READ statement for that file.

b. The AT END clause is valid only for those files whose access mode is SEQUENTIAL (explicitly or implicitly).

The INVALID KEY clause is valid only for those files whose access mode is RANDOM or INDEXED.

If an end-of-file condition is encountered during the execution of a READ statement for a sequential file, the statement(s) specified in the AT END clause is executed, and no logical record is made available.

The logical end-of-file depends upon the type of device to which the file is assigned (see DECsystem-10 Monitor Calls manual, which appears in the DECsystem-10 Software Notebooks). After execution of the imperative-statement(s) in the AT END clause, no further READ statements can be executed for that file without first executing a CLOSE statement followed by an OPEN statement for the file.

If, during the execution of a READ statement for a file whose access mode is RANDOM, the ACTUAL KEY is found to contain a value not within the range specified by the FILE-LIMITS clause for that file or a value for a record that has not been written (i.e., a zero-length record), the statement(s) specified in the INVALID KEY clause is executed and no logical record is made available.

When a READ statement is executed for a file whose access mode is RANDOM and the ACTUAL KEY contains a value of 0, the first nonzero-length record having a key higher than the last record processed (by a READ or WRITE statement) is made available. If no such record exists (i.e., end-of-file), the INVALID KEY statement(s) is executed and no record is made available. If the file has been opened but no READ or WRITE statement has been executed, the first nonzero-length record is made available. The user can use this method to sequentially read a file whose access mode is RANDOM.

When a READ statement is executed for a file whose access mode is INDEXED and the SYMBOLIC KEY contains a value other than LOW-VALUES, a search of the file is made to find the record that has a key equal to the contents of the SYMBOLIC KEY associated with the file. If that record is found, it is moved to the record area for the file; if it is not found, the statement(s) associated with the INVALID KEY clause is executed, and no record is made available.

When a READ statement is executed for a file whose access mode is INDEXED and the SYMBOLIC KEY contains a value of LOW-VALUES, the first logical record having a key higher than the last record processed (by a READ, WRITE, REWRITE, or DELETE statement) is made available. If no such record exists (i.e., end-of-file), the INVALID KEY statement(s) is executed, and no record is made available. If the file has been opened but no READ, WRITE, REWRITE, or DELETE statement has been executed, the first record of the file is made available.

c. If a file described by an OPTIONAL clause is not present, the imperative-statement(s) in the AT END or INVALID KEY clause is executed on the first READ for that file. Any specified USE procedures are not performed.

d. If logical end-of-reel is recognized during execution of a READ statement, the following operations are carried out.

(1) The reel is rewound and the user's ENDING LABEL PROCEDUREs are executed, if specified in a USE statement.

(2) If the file is assigned to more than one device, the devices are swapped. The previous reel is rewound and the message

MOUNT REEL nn OF file-name ON device-name

is typed on the user's Teletype console. The program then waits for the operator to type

CONTINUE

after he has mounted the next reel.

(3) The standard beginning label procedure (see Chapter 8) and the user's BEGINNING LABEL PROCEDURE are executed, if specified in a USE statement.

(4) The first data record on the new reel is made available.

e. If a file consists of more than one type of logical record, these records automatically share the same storage area. This is equivalent to an implied REDEFINE for the record area. Only information in the current record is accessible.

f. If the INTO identifier option is specified, the READ statement is then equivalent to a READ without the INTO option, followed by a MOVE of the record area associated with the file-name to identifier.

RELEASE

Function

The RELEASE statement transfers records to the initial phase of the sort operation.

General Format

RELEASE record-name [FROM identifier]

Technical Notes

a. A RELEASE statement may be used only in an input procedure associated with a SORT statement for a file whose SD description contains record-name.

b. If the FROM option is used, the contents of identifier are moved to record-name, then the contents of record-name are released to the sort subroutines.

c. After the RELEASE statement is executed, the contents of record-name may no longer be available.

d. Refer to the description of the SORT verb for examples.

RETURN

Function

The RETURN statement obtains sorted records from the final phase of a sort operation.

General Format

RETURN file-name RECORD [INTO identifier] AT END statement-1 [, statement-2] ... .

Technical Notes

a. File-name must be described by an SD file descriptor.

b. A RETURN statement may be used only in an output procedure associated with a SORT statement for file-name.

c. If the INTO phrase is specified, the current record is moved from the record area associated with file-name to identifier.

d. After executing the statement(s) in the AT END clause, no RETURN statements may be executed until another SORT is executed.

e. Refer to the description of the SORT verb for examples.

Function

The REWRITE statement replaces an already existing record in a file whose access mode is INDEXED.

General Format

REWRITE record-name [FROM identifier] INVALID KEY statement-1 [, statement-2]... .

Technical Notes

a. Record-name must be a record associated with a file whose access mode is INDEXED.

b. When the REWRITE statement is executed, a record in the file is located whose key value is equal to the contents of the SYMBOLIC KEY associated with the file. The contents of the SYMBOLIC KEY item are moved to the RECORD KEY item and the contents of the record are then replaced with the contents of record-name. If no such record exists in the file, the statement(s) associated with the INVALID KEY clause is executed.

c. At the time the REWRITE statement is executed, the file must be open for OUTPUT or INPUT-OUTPUT.

d. If the FROM option is used, the statement is equivalent to:

MOVE identifier TO record-name
REWRITE record-name (without the FROM option)

SEARCH

Function

The SEARCH statement is used to search a table until a specified condition exists.

General Format

Option 1

```
SEARCH identifier-1  [VARYING identifier-2] [AT END statement-1 [, statement-2 ] ...]
WHEN condition-1 { statement-3 [, statement-4] ... }
                 { NEXT SENTENCE                   }
[ , WHEN condition -2 { statement-5 [ , statement-6] ... } ] ... .
                     { NEXT SENTENCE                      }
```

Option 2

```
SEARCH ALL identifier-1  [AT END statement-1 [,statement-2] ... ]
    WHEN condition-1 { statement-3 [, statement-4] ... } .
                     { NEXT SENTENCE                   }
```

Technical Notes

a. If any of the optional clauses are present, they must appear in the order shown.

b. Identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause with an INDEXED BY option. In option 2, identifier-1 must also contain a KEY option in its OCCURS clause.

c. Identifier-2 must be an index, or an elementary numeric item with no places to the right of the decimal point.

d. In option 1, condition-1, condition-2, etc., can be any condition described in Section 6.5.

e. In option 2, condition-1 must consist of a relation condition incorporating the EQUAL TO or equal sign, or a condition-name condition where the VALUE clause contains only a single literal, or a compound condition consisting of two or more such simple conditions connected by AND.

A data-name that appears in the KEY clause of identifier-1 must appear as the subject or object of a test, or be the name of the data-item with which the tested condition-name is associated. However, all preceding data-names in the KEY clause must also be included within condition-1.

f. If the AT END clause is not present, AT END NEXT SENTENCE is assumed.

g. If the VARYING option is not specified, the first index specified in the INDEXED BY option for identifier-1 is used.

If the VARYING option is used, and identifier-2 is the name of an item specified in the INDEXED BY option for identifier-1, then identifier-2 is used as the index. If identifier-2 is not specified in the INDEXED BY option for identifier-1, the first index-name in the INDEXED BY option is used as the index, and identifier-2 will contain the value of the index at each step of the search.

h. If option 1 of the SEARCH verb is used, a serial type of search takes place, starting with the current index setting.

If, at the start of execution of the SEARCH statement, the index contains a value that is not positive or is greater than allowed (greater than the number of occurrences or greater than any DEPENDING item), the statement(s) specified in the AT END statement is executed.

If, at the start of execution of the SEARCH statement, the index is within the allowed range of values, the WHEN conditions are evaluated one at a time. If any condition is true, the associated statement(s) is executed. If no condition is true, the index is incremented by 1, and the search operation is executed again.

The contents of the index are always left as they were when the search is terminated, either by a WHEN condition, or the AT END condition.

i. If option 2 of the SEARCH verb is used, a binary search takes place. It is up to the user to ensure that the keys associated with the table are in the correct sequence (either ascending or descending). The initial contents of the index are ignored; instead, the table is examined until the WHEN condition is satisfied (in which case statement-3 and any following statements are executed) or until the entire table is examined (in which case the AT END statement(s) is executed).

When the search is terminated, the contents of the index reflect the occurrence number of the entry that satisfied the WHEN condition if it was satisfied, or is abitrary if it was not satisfied.

j. In either option, after any WHEN or AT END statement(s) is executed, control is transferred to NEXT SENTENCE unless that statement contained a GO TO.

k. If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (i.e., this is a multidimensional table), only the index associated with identifier-1 is modified during the search. To search an entire multidimensional table, the SEARCH statement must be executed several times.

```
01  TABLE.
    02  TABL1 OCCURS 200 TIMES INDEXED BY I,
        ASCENDING KEYS A, B.
        03 A,     PICTURE XXX.
        03 FOO,   PICTURE X(20).
        03 B,     PICTURE 9(4).
        03 DES,   PICTURE X(40).
        03 AM,    PICTURE S9(5)V99.
              :

SEARCH ALL TABL1, AT END GO TO WHAT-HAPPENED;
    WHEN A(I)="XYZ" AND B(I)=350 GO TO GO-ONE.
```

SEEK

Function

The SEEK statement initiates the accessing of a mass storage data record for subsequent reading or writing.

General Format

SEEK file-name RECORD

Technical Notes

a. The SEEK statement uses the contents of the ACTUAL KEY item to position the read-write arms on a mass storage device. If the key is invalid, no action is taken; however, if the contents of ACTUAL KEY are not changed before the next READ or WRITE statement is executed, that READ or WRITE statement will then take the INVALID KEY path.

b. The file must be assigned to a mass-storage device, and the ACCESS MODE must be RANDOM.

c. The statement cannot be used for files whose ACCESS MODES are SEQUENTIAL or INDEXED.

Function

The SET statement allows a data-item to be incremented, decremented, or set to a value.

General Format

$$\underline{\text{SET}}\ \text{identifier-1}\ [\,,\ \text{identifier-2}\,]\ldots \left\{ \begin{array}{l} \underline{\text{TO}} \\ \underline{\text{UP BY}} \\ \underline{\text{DOWN BY}} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-3} \\ \text{literal-1} \end{array} \right\}$$

Technical Notes

a. All identifiers must be numeric elementary items described without any positions to the right of the assumed decimal point.

All literals must be integers, or the figurative constant ZERO.

b. The SET statement causes identifier-1, identifier-2, ... to be set (TO), incremented (UP BY), or decremented (DOWN BY) the value of identifier-3 or literal-1.

SORT

Function

The SORT statement provides the capability of ordering a file of records according to a set of user-specified keys within a record.

General Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1
                    { DESCENDING }

    [ , data-name-2] ... [ ON { ASCENDING  } KEY data-name-3
                              { DESCENDING }

        [ , data-name-4] ...] ...

{ INPUT PROCEDURE IS procedure-name-1 [ THRU procedure-name-2] }
{ USING file-name-2                                            }

{ OUTPUT PROCEDURE IS procedure-name-3 [ THRU procedure-name-4] }
{ GIVING file-name-3                                           }
```

Technical Notes

a. File-name-1 must be described in an SD file description entry in the Data Division. Each data-name must represent data items described in records associated with file-name-1.

b. File-name-2 and file-name-3 must be described in an FD file description, not an SD file description, in the Data Division. All records associated with file-name-2 must be large enough to contain all of the KEY data-names.

c. The data-names following the word KEY are listed in order of decreasing significance without regard to how they are divided into KEY clauses.

d. The data-names may be qualified but not subscripted, and must appear in a record description in an SD file description. If the records are ASCII (DISPLAY-7), their maximum size cannot exceed 3400 characters.

e. SORT statements may appear anywhere in the Procedure Division except in the Declaratives portion or in an input or output procedure associated with a sort.

f. When the ASCENDING clause is used, the sorted sequence is from the lowest value to the highest value; when a DESCENDING clause is used, the sorted sequence is from the highest value to the lowest value.

g. The input procedure, if present, must consist of one or more sections or paragraphs that appear contiguously in a source program and do not form a part of any output procedure. The input procedure must contain at least one RELEASE statement in order to transfer records to the sort subroutine.

h. The output procedure, if present, must consist of one or more sections or paragraphs that appear contiguously in a source program and do not form a part of any input procedure. The output procedure must contain at least one RETURN statement in order to make sorted records available for processing.

i. ALTER, GO and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedures; similarly, ALTER, GO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedures.

j. If an input or output procedure is specified, those procedures are PERFORMed by the SORT statement, and all rules relating to the range of a PERFORM must be observed.

k. If the USING option is specified, all the records in file-name-2 are automatically transferred to the sort subroutine. At the time of the execution of the SORT statement, file-name-2 must not be open. Any USE procedures associated with file-name-2 will be executed as appropriate.

The USING option is equivalent to the following input procedure:

```
%1. OPEN INPUT file-name-2
%2. READ file-name-2 INTO sort-record; AT END GO TO %3.
    RELEASE sort-record.
    GO TO %2.
%3. CLOSE file-name-2.
```

l. If the GIVING option is specified, all the sorted records in file-name-1 are automatically transferred to file-name-3. At the time of the execution of the SORT statement, file-name-3 must not be open. Any USE procedures associated with file-name-3 will be executed as appropriate.

The GIVING option is equivalent to the following output procedure:

```
%4. OPEN OUTPUT file-name-3.
%5. RETURN sort-file INTO record-name-3; AT END GO TO %6.
    WRITE record-name-3.
    GO TO %5.
%6. CLOSE file-name-3.
```

m. If an input or output procedure is present, file-name-2 or file-name-3 must be opened before reading or writing sort-records and closed when sorting is ended.

n. Two programs that show sorting can be found in Chapter 1.

STOP

Function

The STOP statement halts the object program.

General Format

Technical Notes

a. If the literal option is used, the literal is displayed on the user's Teletype console, and the program waits for the operator to type

CONTINUE

Following receipt of this message, the program continues execution at the statement following the STOP.

The literal may be a figurative constant; in this case, a single character is displayed.

b. If the RUN option is used, all files currently open are closed, and execution of the program is terminated.

STRING

Function

The STRING statement is used to concatenate the partial or complete contents of several data items into a single data item.

Format

```
STRING {identifier-1} [, identifier-2] ... DELIMITED BY {identifier-3}
       {literal-1   }  [, literal-2   ]     ...          {literal-3   }
                                                         {SIZE        }

    [, {identifier-4} [, identifier-5] ... DELIMITED BY {identifier-6}  ] ...
       {literal-4   }  [, literal-5   ]     ...          {literal-6   }
                                                         {SIZE        }

    INTO identifier-7   [WITH POINTER identifier-8]

    [; ON OVERFLOW statement-1]
```

Technical Notes

a. Source Items

1. The data items referenced by identifier-1 identifier-2, ... are called source data items. Source data items must be alphanumeric data items or numeric data items with DISPLAY-6 or DISPLAY-7 usage.

2. If a source item is an unsigned numeric data item, it is treated in the execution of the STRING statement as if it were an alphanumeric data item.

3. If a source item is a signed numeric data item, it is treated in the execution of the STRING statement as if it had been moved to an unsigned numeric data item of the same size and that new data item is used as the source item.

4. If subscripting or indexing is needed to identify a source data item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of that particular source item are used.

5. Literal-1, literal-2 ... are called source literals. Source literals must be alphanumeric literals or alphanumeric figurative constants without the ALL modifier.

6. If a source literal is a figurative constant, it refers to a single-character literal of the specified type.

b. Delimiter Items

1. Each series of source items specified in the STRING statement must be followed by a DELIMITED BY phrase. This phrase specifies the delimiter condition to be associated with each source item in that series.

2. The data items referenced by identifier-3 and identifier-6 are called delimiter data items. Delimiter data items must be alphanumeric data items or numeric data items with DISPLAY-6 or DISPLAY-7 usage.

3. If a delimiter item is an unsigned numeric data item, it is treated in the execution of the STRING statement as if it is an alphanumeric data item.

4. If a delimiter item is a signed numeric data item, it is treated in the execution of the STRING statement as if it had been moved to an unsigned numeric data item of the same size and that new data item is used as the delimiter item.

5. If subscripting or indexing is needed to identify a delimiter data item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of the source item corresponding to that particular delimiter item are used.

6. Literal-3 and literal-6 are called delimiter literals. Delimiter literals must be alphanumeric literals or alphanumeric figurative constants without the ALL modifier.

7. If a delimiter literal is a figurative constant, it refers to a single-character literal of the specified type.

8. If a delimiter data item or a delimiter literal is specified, the content of the data item, during the execution of the STRING statement, or the value of the literal is the delimiter string for each source item corresponding to that delimiter item.

   In this case, the delimiter condition for each of the corresponding source items is the first occurrence in the source item of a character string that matches the delimiter string. If there is no such character string in the source item, the delimiter condition is the rightmost boundary of that source item.

   N.B. Two character strings match if, and only if, they are of equal length and each character of the first string is equivalent, according to the rules for code conversion, to the corresponding character of the second string.

9. If the DELIMITED BY SIZE phrase is specified, the only delimiter condition for each of the corresponding source items is the rightmost boundary of the source item.

c. Destination

1. The data item referenced by identifier-7 is called the destination. The destination must be an unedited alphanumeric data item. It cannot be justified (i.e., the JUSTIFIED clause cannot be used for this item).

2. If subscripting or indexing is needed to identify the destination, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the execution of the STRING statement are used.

d. Pointer

1. The data item referenced by identifier-8 is called the pointer. The pointer must be an unedited integer data item of sufficient size to contain a value one greater than the size of the destination.

2. The pointer serves as a character index for the destination.

3. If subscripting or indexing is needed to identify the pointer, the values of the required subscripts and/or indexes and the depending items, if any, prior to the execution of the STRING statement are used.

4. If the POINTER phrase is specified, the pointer is directly available to the user. It must be initialized before the execution of the STRING statement to a value greater than zero and not greater than the size of the destination.

5. If the POINTER phrase is not specified, the STRING statement is always executed as if the user had specified a pointer and set the initial value to 1. In this case, the pointer is not directly available to the user.

6. The STRING statement is executed as if the initial value of the pointer were stored in a temporary location. This temporary location is used as the pointer during the execution of the STRING statement. The final value of the temporary location is stored into the real pointer item at the end of the execution.

e. Execution

1. When the STRING statement is executed, each source item in turn, starting with the first source item specified, is transferred to the destination character-by-character, beginning at the leftmost character position of the source item and continuing to the right, until the delimiter condition corresponding to that source item has been encountered or the destination has been filled.

2. If a delimiter item was specified for a source item and a string of characters is found in the source item matching the delimiter string, all characters of the source item preceding the matching string are used in the transfer to the destination, but none of the characters that are in the matching string and no characters following it in the source item are used in the transfer.

3. If no delimiter item was specified for a source item or no string of characters is found in the source item matching the delimiter string, all characters of the source item are used in the transfer to the destination.

4. During the execution of the STRING statement, characters are transferred to the destination from the source items as if the destination were a table of single character data items indexed by the pointer, being automatically incremented after each character transfer.

5. The first character transferred is stored in the character position of the destination indicated by the initial value of the pointer. The nth character transferred is stored in the character position indicated by the initial value of the pointer plus n-1.

6. The transfer of characters ends when one of the following conditions occurs. These conditions are specifically checked for in the order stated.

   (a) The initial value of the pointer is not a positive integer less than or equal to the size of the destination.

   (b) All appropriate characters of all source items have been transferred to the destination.

   (c) A character has been transferred to the last character position of the destination, though not all appropriate characters of all source items have been transferred.

7. If the transfer of characters to the destination is terminated because of condition (b) of note 6, those character positions of the destination to which characters were not transferred, if any, will retain the values they contained before the execution of the STRING statement. That is, remaining character positions in the destination are not space-filled.

8. After the transfer of characters to the destination has ended, the pointer is set to a value one greater than the ordinal number of the last character position of the destination to which data was transferred.

f. Overflow

1. If the transfer of characters to the destination is terminated because of either condition (a) or condition (b) of note 6, the STRING statement is considered to have caused an overflow.

2. If the ON OVERFLOW phrase is not specified, after the execution of the STRING statement, regardless of whether or not there was an overflow, control passes to the point in the program immediately following the STRING statement.

3. If the ON OVERFLOW phrase is specified, after the transfer of characters has ended and the pointer set to the appropriate value, the flow of control of the program depends on whether or not there was an overflow.

   (a) If an overflow did not occur, control passes to the point in the program corresponding to the end of the sentence containing the STRING statement (following all the statements in the ON OVERFLOW phrase).

   (b) If an overflow did occur, control passes to the point in the program corresponding to the beginning of statement-1.

SUBTRACT

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric items from one or more numeric items and set the values of one or more items to the result.

General Format

Option 1

```
SUBTRACT { identifier-1 } [ , { identifier-2 } ] ...
         { literal-1    }     { literal-2    }

    FROM identifier-m [ ROUNDED ] [ , identifier-n [ ROUNDED ] ] ...

[ ON SIZE ERROR statement-1 [ , statement-2 ] ... ]
```

Option 2

```
SUBTRACT { identifier-1 } [ , { identifier-2 } ] ...
         { literal-1    }     { literal-2    }

    FROM { identifier-m } GIVING identifier-n [ ROUNDED ]
         { literal-m    }

    [ identifier-p [ ROUNDED ] ] ...

[ ON SIZE ERROR statement-1 [ , statement-2 ] ... ]
```

Option 3

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2
         { CORR          }

[ ROUNDED ] [ ON SIZE ERROR statement-1 [ , statement-2 ] ... ]
```

Technical Notes

a. Each SUBTRACT statement must contain at least two operands (i.e., a subtrahend and a minuend).

In options 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers to the right of the word GIVING may refer to numeric edited items. In option 3, each identifier must refer to a group item.

Each literal must be a numeric literal or the figurative constant ZERO.

b. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

c. Option 1 causes the values of the operands preceding the word FROM to be added together, and this sum to be subtracted from the values of identifier-m, identifier-n, etc.

d. Option 2 causes the values of the operands preceding the word FROM to be added together, the sum subtracted from identifier-m or literal-m, and the result stored as the new values of identifier-n, identifier-p, etc. The current values of identifier-n, identifier-p, etc., do not enter into the computation.

e. Option 3 causes the data items in the group item associated with identifier-1 to be subtracted from and stored into corresponding data items in the group item associated with identifier-2. The criteria used to determine whether two items are corresponding are described under "The CORRESPONDING Option" at the beginning of this chapter.

f. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

TERMINATE

Function

The TERMINATE statement ends the processing of a report.

General Format

```
TERMINATE report-name-1 [, report-name-2] ... .
```

Technical Notes

a. Each report-name must be defined by an RD entry in the REPORT SECTION of the DATA DIVISION.

b. All control footings associated with the report are produced as if a control break had occurred at the highest level. In addition, the last PAGE FOOTING and any REPORT FOOTING report groups are produced.

c. A second TERMINATE statement for a particular report may not be executed until another INITIATE statement is executed for that report.

d. The TERMINATE statement does not close the file associated with the report; a CLOSE statement must be executed after the TERMINATE statement is executed.

e. An example of the use of the TERMINATE statement is included in the sample report-writing program in Chapter 5, Section 5.7.

TRACE

Function

The TRACE statement causes the compiler to trace paragraphs or to stop tracing paragraphs. When a paragraph is traced, its name, enclosed in angle brackets (< >) is typed each time that the paragraph is entered.

General Format

TRACE { ON / OFF }

Technical Notes:

a. The compiler generates trace calls for each paragraph in the program if the /P switch is not included in the command string. If the /P switch is included in the command string, the trace calls are not generated.

b. Although the compiler generates trace calls when the /P switch is not present, tracing is not performed unless the user includes the TRACE ON statement in his program.

c. The TRACE ON statement causes all ensuing paragraphs to be traced; i.e., their names, enclosed in angle brackets (< >), are typed each time they are entered. Tracing continues until either the end of program is reached or a TRACE OFF statement is encountered.

d. The TRACE OFF statement stops tracing of all ensuing paragraphs until either the end of program is reached or a TRACE ON statement is encountered.

e. When compiling for a production run, the user should include the /P switch in the command string so that trace calls will not be generated and TRACE statements in the program will be ignored. The following example shows paragraphs with TRACE OFF and TRACE ON statements included.

```
PROCEDURE DIVISION.
PARA.
    .
    .
    .
TRACE ON.
PARB.
    .
    .
    .
```

```
TRACE OFF.
PARC.
    :
    :
TRACE ON.
PARD.
    :
    :
```

Paragraphs PARB and PARD are traced. Paragraph PARC is not traced because the TRACE OFF statement is included immediately before it. If the /P switch is included in the command string when this program is compiled, the TRACE statements will be ignored and trace calls will not be generated.

UNSTRING

Function

The UNSTRING statement is used to split a single data item (e.g., a text string) into several parts, depending on the occurrence of specified delimiters, and to store the parts into separate data items where they may be more easily accessed by the COBOL program.

Format

```
UNSTRING identifier-1
[ DELIMITED BY [ALL] { identifier-2 } [ , OR [ALL] { identifier-3 } ] ... ]
                     { literal-1   }               { literal-2   }
  INTO identifier-4 [, DELIMITER IN identifier-5] [, COUNT IN identifier-6]
    [ , identifier-7 [, DELIMITER IN identifier-8] [, COUNT IN identifier-9] ] ...
  [WITH POINTER identifier-10] [TALLYING IN identifier-11]
  [; ON OVERFLOW statement-1]
```

Technical Notes

a. Source Items

1. The data item referenced by identifier-1 is called the source. The source must be a DISPLAY data item. If it is a numeric data item, it is treated during the execution of the UNSTRING statement as if it were alphanumeric.

2. If subscripting or indexing is needed to identify the source, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the execution of the UNSTRING statement are used.

b. Destination Items

1. The data items referenced by identifier-4, identifier-7, ..., are called destination items. Destination items may be any kinds of data items.

2. If subscripting or indexing is needed to identify a destination item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of data to that destination item are used.

c. Delimiter Items

1. The data items referenced by identifier-2, identifier-3, ..., are called delimiter data items. Delimiter data items must be alphanumeric data items or numeric data items with DISPLAY-6 or DISPLAY-7 usage.

2. If a delimiter item is an unsigned numeric data item, it is treated in the execution of the UNSTRING statement as if it were an alphanumeric data item.

3. If a delimiter item is a signed numeric data item, it is treated in the execution of the STRING statement as if it had been moved to an unsigned numeric data item of the same size and that new data item is used as the delimiter item.

4. If subscripting or indexing is needed to identify a delimiter data item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of data to each successive destination item are used.

5. Literal-1, literal-2, ..., are called delimiter literals. Delimiter literals must be alphanumeric literals or alphanumeric figurative constants without the ALL modifier.

6. If a delimiter literal is a figurative constant, it refers to a single-character literal of the specified type.

7. If a delimiter data item or a delimiter literal is specified, the content of the data item, during the execution of the UNSTRING statement, or the value of the literal is a delimiter string for the source.

8. If more than one delimiter item is specified, the delimiter items are separated by the connective OR. In this case, the several delimiter strings are ordered by the order in which the delimiter items specifying them occur in the UNSTRING statement.

9. If the ALL phrase is specified with a delimiter item, the delimiter string that that item specifies is considered to consist of as many occurrences of that simple delimiter string as can be found contiguously stored in the source.

10. A delimiter condition is an occurrence in the source of a character string, not contained in the portion of the source that has already been scanned, that matches one of the delimiter strings, or the rightmost boundary of the source.

d. Delimiter Storage Items

1. A DELIMITER IN phrase may be specified only if the DELIMITED BY phrase is specified.

2. The data items referenced by identifier-5 and identifier-8 are called delimiter storage items. Delimiter storage items may be any kinds of data items.

3. If subscripting or indexing is needed to identify a delimiter storage item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of data to the destination item corresponding to that delimiter storage item are used.

e. Count Storage Items

1. A COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

2. The data items referenced by identifier-6 and identifier-9 are called count storage items. Count storage items must be unedited integer data items of sufficient size to contain a value equal to the size of the source.

3. If subscripting or indexing is needed to identify a count storage item, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the transfer of data to the destination item corresponding to that count storage item are used.

4. A count storage item is used to store the number of characters of the source that were examined during the execution of the UNSTRING statement and approved for transfer to the destination corresponding to that count storage item.

   N.B. This is not necessarily the same as the number of characters that were actually transferred, because the destination may be too small to hold all that were approved for transfer.

f. Pointer

1. The data item referenced by identifier-10 is called the pointer. The pointer must be an unedited integer data item of sufficient size to contain a value one greater than the size of the source.

2. The pointer serves as a character index for the source.

3. If subscripting or indexing is needed to identify the pointer, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the execution of the UNSTRING statement are used.

4. If the POINTER phrase is specified, the pointer is directly available to the user. It must be initialized before the execution of the UNSTRING statement to a value greater than zero and not greater than the size of the source.

5. If the POINTER phrase is not specified, the UNSTRING statement is always executed as if the user had specified a pointer and set the initial value to 1. In this case, the pointer is not directly available to the user.

6. The UNSTRING statement is executed as if the initial value of the pointer were stored in a temporary location. This temporary location is used as the pointer during the execution of the UNSTRING statement. The final value of the temporary location is stored into the real pointer item at the end of the execution.

g. Destination Counter

1. The data item referenced by identifier-11 is called the destination counter. The destination counter must be an unedited integer data item of sufficient size to contain a value equal to the number of destinations specified in the UNSTRING statement.

2. The destination counter is used to store the number of destination items to which data was transferred by the execution of the UNSTRING statement.

3. If subscripting or indexing is needed to identify the destination counter, the values of the required subscripts and/or indexes and the depending items, if any, just prior to the execution of the UNSTRING statement are used.

4. If the TALLYING phrase is specified, the destination counter is directly available to the user, and it must be initialized before the execution of the UNSTRING statement.

5. If the TALLYING phrase is not specified, the UNSTRING statement is always executed as if the user had specified a destination counter and set the initial value to 0. In this case, the destination counter is not directly available to the user.

6. The UNSTRING statement is executed as if the initial value of the delimiter storage item were stored in a temporary location. This temporary location is used as the delimiter storage item during the execution of the UNSTRING statement. The final value of the temporary location is stored into the real delimiter storage item at the end of the execution.

h. Execution

1. The execution of the UNSTRING statement is an iterative process. There is one iteration for each destination item specified in the UNSTRING statement, starting with the first destination item specified and continuing in order through the series of destination items. However, the iteration process will be stopped after all the data in the source has been used, even if not all destination items have been used.

2. Each iteration of the process involved in the execution of the UNSTRING statement consists of the following steps:

   (a) Select a set of characters from the source.

   (b) Move a representation of these characters to the destination item for that iteration.

   (c) Set the count storage item corresponding to that destination item, if one is specified.

   (d) Move some characters to the delimiter storage item corresponding to that destination item, if one is specified.

   (e) Advance the pointer to indicate a new position in the source.

   (f) Increment the destination counter.

3. During the execution of the UNSTRING statement, the source is treated as if it were a table of single character data items indexed by the pointer. The character position of the source indicated by the pointer, during each iteration of the UNSTRING process, is called the pointer-indicated position for that iteration. Only the pointer-indicated position for an iteration and those source character positions to its right are used during that iteration. Character positions to the left of that position are not involved in that iteration in any way.

4. During each iteration of the UNSTRING process, a scan of the source is done to determine which characters of the source will be selected as the character set to be moved to the appropriate destination item. This scan begins at the pointer-indicated position and continues to the right in the source.

5. The scan of the source watches for any of the following conditions (in the specified order):

   (a) A string of contiguous characters in the source matching one of the delimiter strings.

   (b) The rightmost boundary of the source.

   (c) A number of characters sufficient to completely fill the destination item for this iteration.

   When one of these conditions is found, the scan ends and the set of characters to be moved to the destination item is then known.

6. The source scan described in Note 5 proceeds as follows:

   (a) Each character position of the source, starting at the pointer-indicated position and continuing to the right, is first checked to see if any source character-string beginning at that position matches the delimiter-string specified by the first delimiter item in the UNSTRING statement. If such a string is found, condition (a) of Note 5 is satisfied.

   (b) If no such string is found, the same character position is then checked to see if any source character-string beginning at that position matches the second specified delimiter-string. This process is repeated using each successive delimiter string until either condition (a) of Note 5 is satisfied or all specified delimiter strings have been tried.

   (c) If condition (a) of Note 5 is not satisfied for the source character position under consideration and one of the specified delimiter-strings, that character position is then selected as part of the source to be moved to the current destination item.

   (d) The above process continues until either no more source character positions remain (condition (b) of Note 5), or enough successive character positions of the source have been selected to entirely fill the destination item (condition (c) of Note 5).

7. During each iteration of the UNSTRING process, the set of contiguous source character positions selected by the process described in Note 6 is considered to be a complete individual data item, and is moved to the current destination item according to the rules for the MOVE statement, including any class or usage conversion that might be necessary. This data item might contain no character positions at all (if the pointer-indicated position satisfied condition (a) of note 5), or it may contain as much as the entire source.

8. If a count storage item is specified with the destination item for an iteration of the UNSTRING process, the number of source characters that are moved to the destination item is stored in that count storage item.

9. If there is a delimiter storage item specified for a particular iteration of the UNSTRING process, then:

   (a) If the selection of source character positions described in Note 5 is terminated because of condition (a) of Note 5, the string of contiguous source character positions that contain the match for a delimiter string is treated as a complete individual data item and is moved to the delimiter storage item according to the rules for the MOVE statement, including truncation if necessary.

If, in this case, the delimiter string that was matched is described with the ALL phrase, the set of source character positions containing a match for the simple delimiter string, plus every immediately succeeding set of contiguous source character positions containing a match for the same delimiter string, are used in the data item that is moved to the delimiter storage item.

(b) If the selection of source character positions described in Note 5 is terminated because of condition (b) or (c) of Note 5, spaces are moved to the specified delimiter storage item.

10. In an iteration of the UNSTRING process, after the appropriate data has been stored in the destination item, the delimiter storage item, and the count storage item, the pointer is set to a value one more than the ordinal number of the last source character position that participated in the selection process. This includes all character positions that were selected for the move to the destination item and all that were moved to the delimiter storage item (or would have been so moved if the delimiter storage item were specified).

11. At the conclusion of execution of the UNSTRING statement, the destination counter is set to a value equal to its initial value plus the number of destination items to which data was transferred by the execution of the UNSTRING statement.

i. Overflow

1. If the initial value of the pointer is less than one or greater than the size of the source, execution of the UNSTRING statement is aborted before any data is transferred, and the UNSTRING statement is considered to have caused an overflow.

2. If, during the execution of an UNSTRING statement, data has been transferred to all of the destination items in accordance with Note 7, but the updated pointer still contains a value less than or equal to the size of the source (i.e., not all of the source character positions have been used in the UNSTRING process), the UNSTRING statement is considered to have caused an overflow.

3. If the ON OVERFLOW phrase is not specified, after the execution of the UNSTRING statement, regardless of whether or not there was an overflow, control passes to the point in the program immediately following the UNSTRING statement.

4. If the ON OVERFLOW phrase is specified, after the transfer of characters has ended and the pointer and destination counter are set to the appropriate values, the flow of control of the program depends on whether or not there was an overflow.

(a) If an overflow did not occur, control passes to the point in the program corresponding to the end of the sentence containing the UNSTRING statement (following all the statements in the ON OVERFLOW phrase).

(b) If an overflow did occur, control passes to the point in the program corresponding to the beginning of statement-1.

Function

The USE statement specifies procedures for input-output label and error handling that are in addition to the standard procedures provided.

General Format

Format 1

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name-1 [OPEN] }
                                      { INPUT            }
                                      { OUTPUT           }
                                      { I-O              }
                                      { INPUT-OUTPUT     } .
```

Format 2

```
USE { BEFORE } STANDARD [ { BEGINNING } ] [ { REEL } ]
    { AFTER  }           { ENDING    }     { FILE }
                                           { UNIT }

    LABEL PROCEDURE ON { file-name-1 }
                       { INPUT       }
                       { OUTPUT      }
                       { I-O         }
                       { INPUT-OUTPUT } .
```

Format 3

```
USE BEFORE REPORTING identifier-1 .
```

Technical Notes

a. USE statements may appear only in the DECLARATIVES portion of the PROCEDURE DIVISION.

The DECLARATIVES portion follows immediately after the PROCEDURE DIVISION header and begins with the word

DECLARATIVES.

The DECLARATIVES portion ends with the words

END DECLARATIVES.

Following this must be a section-header as the first entry of the main portion of the PROCEDURE DIVISION.

The DECLARATIVES portion itself consists of USE sections, each consisting of a section-header, followed by a USE statement, followed by the associated procedure paragraphs.

The general format for the DECLARATIVES portion is given below.

```
PROCEDURE DIVISION.
DECLARATIVES.
section-name-1 SECTION.  USE ......
paragraph-name-1a.  (statement)
[paragraph-name-1b.  (statement)]
.
.
[section-name-2 SECTION.  USE ......]
.
.
.
END DECLARATIVES.
section-name-m SECTION.
```

b. The USE statement may follow on the same line as the section-header and must be terminated by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

c. The USE statement itself is never executed, rather it defines the conditions calling for the execution of the USE procedures.

d. The designated procedures are executed at the appropriate time as follows (see also Chapter 8):

(1) Format 1 causes the designated procedures to be executed after completing the standard input-output error routine.

(2) Format 2 causes the designated procedures to be executed at one of the following times, depending upon the options used.

(a) Before or after a beginning or ending input label check procedure is executed.

(b) Before a beginning or ending output label is created.

(c) After a beginning or ending output label is created, but before it is written on tape.

(d) Before or after a beginning or ending input-output label check procedure is executed.

(3) Format 3 causes the designated procedures to be executed just prior to the production of the named report group.

e. There must not be any references to any non-DECLARATIVES procedure within a USE procedure. Conversely, there must be no reference to procedure-names that appear within the DECLARATIVES portion in the non-DECLARATIVES portion, except that PERFORM statements may refer to a USE section or to a procedure contained entirely within such a USE section.

f. No input/output can be performed, other than use of ACCEPT and DISPLAY statements, during execution of a USE procedure.

g. Format 1 causes the associated procedures to be executed after the standard input-output error routine has been executed. If the INPUT option is used, the procedures will be executed for all INPUT files; if the OUTPUT option is used, they will be executed for all OUTPUT files; if the I-O or the INPUT-OUTPUT option is used, they will be executed for all INPUT-OUTPUT files; if the file-name-1 option is used, they will be executed only for that particular file.

If the filename-1 OPEN option is used, the system performs the associated procedures only if a "FILE BEING MODIFIED" error occurs when an attempt is made to open an output file. After performing the procedure, the system automatically tries again to open the file, repeating this process until the file is opened. This option allows the user to suspend his job until it can access a file that another user is modifying.

h. Format 2 causes the associated procedures to be executed at the appropriate times, as indicated by the options selected (see note d and Chapter 8). If the words BEGINNING or ENDING are not included in Format 2, the designated procedures are executed for both the beginning and ending labels. If neither UNIT, REEL, nor FILE is included, the designated procedures are executed for both REEL (or UNIT) labels and for FILE labels.

If the INPUT, OUTPUT, INPUT-OUTPUT, or I-O option is specified, the label procedure applies to every file of that type except those files described as LABEL RECORDS ARE OMITTED.

If the file-name-1 option is used, its file description must not contain a LABEL RECORDS ARE OMITTED clause.

i. Within a given format, a file-name must not be referred to, either implicitly (i.e., by an INPUT, OUTPUT, INPUT-OUTPUT, or I-O option) or explicitly (i.e., by a file-name-1 option), in more than one USE statement.

j. Identifier-1 in Format 3 represents a report group named in the REPORT SECTION of the DATA DIVISION. An identifier must not appear in more than one USE statement. The report group must not be TYPE DETAIL.

WRITE

Function

The WRITE statement transfers a logical record to an output file.

General Format

Option 1

```
WRITE record-name-1 [FROM identifier-1]
[{BEFORE} ADVANCING {identifier-2 LINES}]
[{AFTER }           {integer-1 LINES   }]
                    {mnemonic-name     }
```

Option 2

```
WRITE record-name-1 [FROM identifier-1] INVALID KEY statement-1 [, statement-2] ... .
```

Technical Notes

a. An OPEN OUTPUT or OPEN I-O or OPEN INPUT-OUTPUT statement must be executed for the file prior to the execution of the WRITE statement.

b. After the WRITE is executed, the data in record-name-1 may no longer be available.

c. Record-name-1 must be the name of a logical record in a DATA RECORDS clause of the FILE SECTION of the DATA DIVISION.

d. Format 1 is valid for any file currently open for output, with ACCESS MODE IS SEQUENTIAL.

The ADVANCING clause allows control of the vertical positioning of the paper form for print files as follows.

(1) If the ADVANCING clause is not specified and the recording mode is ASCII (see Chapter 8), BEFORE ADVANCING 1 LINE is assumed.

(2) If identifier-2 or integer-1 is specified, it must represent a positive integer or zero. The form is advanced the number of lines equal to the value of identifier-2 or integer-1.

(3) If mnemonic-name is specified, the form is advanced until the specified channel is encountered on the paper-tape format control loop. Mnemonic-name must be defined by a CHANNEL clause in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

(4) If the BEFORE option is used, the record is printed before the form positioning.

(5) If the AFTER option is used, the record is printed after form positioning occurs, and no form positioning takes place after the printing.

If end-of-reel is encountered while writing on magtape, the WRITE statement performs the following operations.

(1) The user's ENDING LABEL PROCEDURE is executed, if specified by a USE statement.

(2) A file mark is written, and the tape is rewound.

(3) If the file was assigned to more than one tape unit, the units are swapped.

(4) The operating system types the message.

MOUNT AVAILABLE TAPE ON device-name

and waits for the operator to type CONTINUE.

(5) A label is written on the new tape, if labels are not OMITTED, and any user's BEGINNING LABEL PROCEDURE is executed.

e. Format 2 is valid only for random-access files whose access mode is RANDOM or INDEXED.

The imperative-statement(s) in the INVALID KEY clause of a RANDOM file is executed when an attempt is made to execute a WRITE to a segment outside the range of the FILE-LIMITS of the file.

When a WRITE statement is executed for a file whose access mode is RANDOM and the ACTUAL KEY contains a value of 0, records will be written sequentially in the file (i.e., no records will be left null). If the previous operation performed on the file was by a READ statement, the previous record will be replaced (i.e., the record will be updated).

When a WRITE statement is executed for a file whose access mode is INDEXED, the contents of the SYMBOLIC KEY item are moved to the RECORD KEY item and the record is written.

The statement(s) in the INVALID KEY clause of an INDEXED file is executed when the SYMBOLIC KEY contains a value equal to the key of an already existing record in the file (refer to the REWRITE statement).

f. When executing a WRITE statement for a SEQUENTIAL file opened for I-O (or INPUT-OUTPUT), the logical record is placed on the file as the next logical record, if the previous input-output operation was a WRITE, or it replaces the previous record, if the previous input-output operation was a READ.

g. If the FROM option is used, the statement is equivalent to:

MOVE identifier-1 TO record-name-1

WRITE record-name-1 (without the FROM option)

Chapter 7
The COBOL Library

The COBOL Library contains source language entries that are available for inclusion in a user's source program at compile time.

For example, a library entry might consist of a PROCEDURE DIVISION section containing procedure statements associated with a frequently used rate computation. If a programmer wishes to include this computation procedure in his program, he need only write a COPY statement referencing this library entry at the point where he wants this procedure included; it is unnecessary for him to write out the full procedure himself. The effect of the COPY is the same as if the text contained in the library entry were actually written as part of the source program. In addition to PROCEDURE DIVISION entries, ENVIRONMENT DIVISION paragraphs and DATA DIVISION FDs and record descriptions can be copied from the library into a user's source program.

COPY

Function

The COPY statement causes inclusion of a library entry into a COBOL source program at the point where the COPY statement appears.

General Format

```
COPY library-name
     [ REPLACING word-1 BY { word-2       }
                           { identifier-1 }
                           { procedure-name-1 }
          [ , word-3 BY { word-4       } ]  ... ] .
                        { identifier-2 }
                        { procedure-name-2 }
```

Technical Notes

a. The COPY statement may appear as follows:

(1) In any of the following paragraphs of the ENVIRONMENT DIVISION:

OBJECT-COMPUTER.	COPY library-name....
SPECIAL-NAMES.	COPY library-name....
FILE-CONTROL.	COPY library-name....
I-O-CONTROL.	COPY library-name....

(2) In place of any of the following clauses in the DATA DIVISION.

FD file-name	COPY library-name....
SD sort-name	COPY library-name....
RD report-name	COPY library-name....
level no. name	COPY library-name....

(3) In a paragraph in the Procedure Division:

paragraph-name. [statement-1] COPY statement. [statement-2].

statement-1 may not start with NOTE.

b. In the ENVIRONMENT DIVISION, no other statement or clause may appear in the same sentence as the COPY statement.

c. COPY causes the specified library text to be copied from the COBOL Library, and the result of compilation is the same as if the text were actually a part of the source program. The COPYing process is terminated by the end of the library text.

d. Both the COPY statement itself and the statements of the library text, after any specified replacing has been performed, appear on the output listing produced by the compiler.

e. The text in the library entry must not contain any COPY statements.

f. If the REPLACING option is used, each word specified in the format is replaced by the stipulated word, identifier, or procedure-name that is associated with it in the format. That is, word-2, identifier-1, or procedure-name-1 replaces every occurrence of word-1 in the text copied from the library. The library entry itself is not altered.

Word-1 and word-3 may be a data-name, procedure-name, condition-name, mnemonic-name, or file-name.

Word-2 and word-4 may be any COBOL word, including literals but excluding picture-strings and reserved words.

Example:

```
COPY LIB001 REPLACING ITEM1 BY AMOUNT OF INCOME-REC.

Library entry:

    TAX-CALC.  MULTIPLY ITEM1 BY TAX-RATE GIVING TAX-DUE.....

Result of COPY:

    TAX-CALC.  MULTIPLY AMOUNT OF INCOME-REC BY TAX-RATE
    GIVING TAX-DUE.....
```

Chapter 8
Standard I-O Processing

This chapter describes how the data is structured, stored, accessed, and moved in I-O operations at run-time. Each of the following topics is described and explained in detail.

a. Access mode
b. Recording mode
c. File tables
d. Channel tables
e. Blocking
f. Label records
g. Multiple-file tape
h. SAME AREA clause
i. SAME RECORD AREA clause
j. File-limits

8.1 ACCESS MODE

Each file has one of the three access modes: SEQUENTIAL, RANDOM, or INDEXED. For random-access and indexed-sequential files, the mode must be specified by means of the ACCESS MODE clause of the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION (refer to Chapter 4).

8.1.1 SEQUENTIAL Mode

In the SEQUENTIAL mode records are accessed in the order in which they appear on the file. Each READ (after the first) brings into memory from the peripheral device the logical record on that device that immediately follows the logical record previously read from or written on that device.

If the file is open for output, each write appends the record to the end of the file. If the file is open for input-output, the write replaces either the record previously read (if the last operation was READ) or the record following the one previously written (if the last operation was WRITE). If the file is open for input-output and is read to the end, each write appends a record to the end of the file.

8.1.2 RANDOM Mode

The record to be accessed is specified by the contents of the ACTUAL KEY, without regard to the physical characteristics of the device. The following conditions must exist:

a. The device must be a random-access medium.

b. The records must be blocked (the blocking factor may be 1).

c. If the recording mode is ASCII (see Section 8.2), the blocking factor must be 1.

The contents of the ACTUAL KEY determine which record, relative to the beginning of the file, is to be read or written. For example, to read the fifth record of a file, the following statements would appear in the source program.

Example:

```
.
.
.
    SELECT FILA ASSIGN TO DSK;
           ACCESS MODE IS RANDOM.
           ACTUAL KEY IS FILE-KEY.
.
.

    MOVE 5 TO FILE-KEY.
    READ FILA; INVALID KEY GO TO YOU-LOSE.
.
.
.
```

NOTE

FILE-KEY is a computational item defined by the user. It consists of 10 or fewer digits with no decimal places.

8.1.2.1 Creating Random-Access Files - A random-access file can be created in one of two ways - randomly or sequentially. To create the file randomly (i.e., by writing into scattered or random records), the user need only open the file, move a value into the ACTUAL KEY for each record, and output each record. When a file is created randomly, there may be zero-length records in the file because data was not written into the record that has that ACTUAL KEY. These dummy records can be written into but cannot be read.

To create the file sequentially, the user can open the file, move 0 into the ACTUAL KEY, and output each record. No zero-length records will be in the file if it is written sequentially.

8.1.3 INDEXED Mode

A file whose access mode is INDEXED (an indexed-sequential or ISAM file) is actually two files: a file containing an index or table of contents, and the actual data file. Both files must be on a random access device, and the data file must be blocked (the blocking factor may be 1).

Each record in an indexed-sequential file has a field, called a RECORD KEY. This field is located in the same relative place for each record, but each record has a different value in its RECORD KEY. Before reading, writing, rewriting, or deleting a record in an indexed sequential file, the user moves the value of the RECORD KEY for the desired record to the SYMBOLIC KEY, which is a field in the WORKING-STORAGE SECTION similar in function to an ACTUAL KEY. For example, to read the record containing "DEC" in its RECORD KEY, the following statements would appear in the program.

```
                                .
                                .
                                .
SELECT ISAM-FILE ASSIGN TO DSK;
       ACCESS MODE IS INDEXED;
       SYMBOLIC KEY IS ISAM-SYMBOLIC KEY;
       RECORD KEY IS ISAM-RECORD-KEY.
                                .
                                .
                                .
FILE SECTION.
FD     ISAM-FILE
       BLOCK CONTAINS 8 RECORDS
       VALUE OF IDENTIFICATION IS "ISAMFLIDX".
01     ISAM-RECORD.
       02 FILLER PIC X(12).
       02 ISAM-RECORD-KEY   PIC X (3).
       02 FILLER   PIC X(75).
                                .
                                .
                                .
WORKING-STORAGE SECTION.
77     ISAM-SYMBOLIC-KEY  PIC XXX.
                                .
                                .
                                .
MOVE "DEC" TO ISAM-SYMBOLIC-KEY.
READ ISAM-FILE; INVALID KEY GO TO YOU-LOSE.
```

To access an indexed sequential file sequentially, the user must declare that the ACCESS MODE IS INDEXED and move LOW-VALUES into the SYMBOLIC KEY. This will cause the file to be accessed in sequential order starting from the first record in the file or the last record accessed (even if an INVALID KEY error occurs). While accessing the file in this manner, the user must not change the value of the SYMBOLIC KEY or the file will be accessed according to the new value.

8.1.3.1 Format of the Index Entry - Each index pointer block in an indexed sequential file is written as if it were a block of a SIXBIT file. (SIXBIT file format can be found in the COBOL Users Guide.) There is one record per block. The format of the block is:

word 1: is the header word. The right half contains the size of the index block in bytes, as if it were SIXBIT (i.e., six bytes per word). The left half contains a number representing the level of the index (the lowest level is 0).

word 2: contains the version number. This is initially set to 0 by the ISAM program, and is incremented by 1 whenever this block is divided due to the insertion of an entry when a WRITE is executed.

Following word 2 are the index entries, packed to the top of the block. Each entry has the format:

word 1: contains the pointer to a data block (if this is index level 0) or a pointer to the next lower level index block (if this is index level 1 or higher).

word 2: contains the version number of the index or data block to which this item points.

words 3-11 contain the value of a key. If the key is nonnumeric, it extends over as many words as are necessary. If the key is numeric, it is kept in COMPUTATIONAL form (even if the record key for the file is DISPLAY). It is one word if 10 or fewer digits are in the key; it is two words if 11 or more digits are in the key.

8.1.3.2 Format of the Data File - The format of the data file is similar to that of RANDOM and SEQUENTIAL files (refer to the COBOL Users Guide, DEC-10-LCUGA-A-D), with the following exceptions.

a. The left half of the header word of a SIXBIT record contains a version number. Only the version number of the first record of a block has any meaning; it pertains to all records for that block.

b. ASCII records are line-blocked; they occupy an integral number of words. They always end with a single carriage-return, line-feed pair.

c. A word is added to the beginning of each ASCII record. The left half contains a version number, bits 18 through 34 contain the size of the record in bytes, and bit 35 is always 1.

8.2 RECORDING MODE

8.2.1 Default Conditions

At both compilation time and run time, assumptions are made regarding the structure, or recording mode, of data as it appears on the external devices. If a recording mode is not specified for a file, the recording mode is assumed to be ASCII if all the data records for that file are described implicitly or explicitly as DISPLAY-7, if the WRITE verb is used with the ADVANCING option, or if the file descriptor has a REPORT clause. In all other cases, the recording mode is assumed to be SIXBIT. At run time, if the device is found to be other than magnetic tape, DECtape, or a random-access device, the recording mode is assumed to be ASCII. Conversions necessary to conform to the USAGE of the records are made automatically by the I-O routines.

8.2.2 ASCII

ASCII records contain a contiguous set of characters. Each record is terminated by a printer control character (ASCII code 12-15, 20-24), or by a string of such characters. The first record in a block starts in the first character position; each succeeding record in that block starts in the first character position following the previous record.

8.2.3 SIXBIT

SIXBIT records contain a contiguous set of words. The right half of the first word, supplied by the I-O routines, contains the number of characters in the record. The last word in the record will, if necessary, have filler added to insure that the record ends at a word boundary, so that records always occupy an integral number of words.

8.2.4 BINARY

Binary records contain a contiguous set of words. There is no particular format for binary files; every word is data.

If there is more than one record described for the file, the READ statement will always read enough words to fill the largest record. The WRITE statement will write variable-size records as it does in other modes.

8.3 FILE TABLES

There is a file table for each file named in a SELECT statement in the user's program (see SELECT statement in Chapter 4). Figure 8-1 shows the structure of a file table. The various fields in that table are described in the paragraph following the figure.

	0 4	6 12	17	35
1	NAME OF THE FILE (IN SIXBIT)			
2				
3				
4				
5				
6	COMPILER VERSION	CHANNEL	NUMBER OF DEVICES	ADDRESS OF FIRST DEVICE NAME
7	NUMBER OF FILE LIMITS	NOT USED	POSITION	ADDRESS OF NEXT FILE TABLE
8	NUMBER OF BUFFERS	RECORD SIZE		RERUN COUNT
9	FLAGS			ADDRESS OF RECORD AREA
10	MAXIMUM SIZE OF ANY NON-STANDARD LABEL			MULTIPLE-FILE ADDRESS
11	NOT USED	BLOCKING FACTOR		ADDRESS OF ACTUAL KEY
12	BYTE POINTER FOR VALUE OF IDENTIFICATION			
13	BYTE POINTER FOR VALUE OF DATE-WRITTEN			
14	ADDRESS OF A FILE TABLE THAT SHARES BUFFER AREA			ADDRESS OF ERROR USE PROCEDURE
15	ADDRESS OF BEFORE BEGINNING REEL			ADDRESS OF BEFORE BEGINNING FILE
16	ADDRESS OF AFTER BEGINNING REEL			ADDRESS OF AFTER BEGINNING FILE
17	ADDRESS OF BEFORE ENDING REEL			ADDRESS OF BEFORE ENDING FILE
18	ADDRESS OF AFTER ENDING REEL			ADDRESS OF AFTER ENDING FILE
19	FLAGS			ADDRESS OF USER-NUMBER
20	BYTE POINTER TO SYMBOLIC KEY			
21	BYTE POINTER TO RECORD KEY			
22	CODES FOR ISAM KEY			
23	RESERVED FOR EXPANSION			
24				
25				
26	FILE LIMIT PAIRS (AS NEEDED)			

Figure 8-1 Structure of a File Table

8.3.1 Explanation of Fields

NAME OF THE FILE	The name specified by the SELECT clause. It is used only for error messages.
COMPILER VERSION	The version number of the current compiler.
CHANNEL	The relative address of the channel table entry assigned to this file when opened.
NUMBER OF DEVICES	The number of devices specified by the ASSIGN clause (see Chapter 4). This number cannot exceed 63.
ADDRESS OF FIRST DEVICE NAME	The address of the first device name in the device-name table. Device names are kept in a table in the order in which they are assigned. After the first name, the addresses of any other assigned device names follow in sequence (i.e., each one is consecutive and contiguous to its predecessor).
NUMBER OF FILE-LIMITS	The number of file-limit pairs associated with the file (see FILE LIMIT clause in Chapter 4).
POSITION	The position of this file relative to the beginning of a multi-file tape. This position is specified by the POSITION clause in the I-O CONTROL paragraph (see Chapter 4).
ADDRESS OF NEXT FILE TABLE	The address of the first word of the next file table. Each file table points to the succeeding table, in reverse order to that in which the files are named in the SELECT statement. The last file table has zeros in this field.
NUMBER OF BUFFERS	The number of buffers requested from the monitor. If the access mode is RANDOM or INDEXED this number is 1. If the access mode is SEQUENTIAL, this number is 2, unless the user requests a different number of buffers by means of the RESERVE clause (see Chapter 4).
RECORD SIZE	The size of the record given in bytes. The size cannot exceed 4095 characters.
RERUN COUNT	The value of the integer if the RERUN EVERY integer RECORDS clause is given for this file.
FLAGS	A half-word in word 9 containing flags and fields with the meanings shown in Table 8-1.
ADDRESS OF RECORD AREA	The address of the first location in core memory allocated to a record in this file.
MAXIMUM SIZE OF ANY NON-STANDARD LABEL	The size of the largest label record described in the FD clause.
MULTIPLE FILE ADDRESS	The link between file tables which share a device. If the same device is used by more than one file, the file tables will be linked in a circular, or round-robin, manner through this field.
BLOCKING FACTOR	The number of records in a block, as specified by the BLOCK CONTAINS clause (see Chapter 5).
ADDRESS OF ACTUAL KEY	The address of the word containing the value to be used as an actual key.

Table 8-1
Flags and Fields in Word 9 of File Table

Bits	Meaning
0-1	Recording mode: 00 = SIXBIT 01 = BINARY 10 = ASCII 11 = EBCDIC
2-3	Type of labels: 00 = OMITTED 01 = STANDARD 10 = NON-STANDARD
4	This file is open for input.
5	This file is open for output.
6	This is a random-access device.
7	AT END path has been taken.
8	Device is a console.
9	Data and recording modes differ.
10	Rerun is to be taken at end of reel.
11	Rerun is to be taken every now and then (see RERUN COUNT field).
12	File is optional and not present.
13	File is optional.
14	Type of data in core: 0 = DISPLAY-6 (SIXBIT) 1 = DISPLAY-7 (ASCII)
15	Not used
16-17	Access mode: 00 = SEQUENTIAL 01 = RANDOM 10 = INDEXED

BYTE POINTER FOR VALUE OF IDENTIFICATION

A byte pointer that points to the byte preceding the first character of a string of characters containing the value of identification.

BYTE POINTER FOR VALUE OF DATE-WRITTEN

A byte pointer that points to the byte preceding the first character of a string of characters containing the value of date-written.

ADDRESS OF A FILE TABLE THAT SHARES BUFFER AREA

The link between file tables which share a buffer area. If this file appears in a SAME AREA clause, all associated files in that clause are linked in a circular, or round-robin, manner through this field.

ADDRESS OF ERROR USE PROCEDURE

The address (words 14 through 18) of the first instructions of the USE procedure as declared in the DECLARATIVES (see Chapter 6). The compiler generates the following code to call a USE procedure:

```
PERF.  17, %Y
JRST   <use procedure>
POPJ   17,
```

%Y is a location in working storage that will contain the return location for a PERFORM. When the I-O routines want to execute a USE procedure, they pick up the appropriate address in an AC and then execute:

```
PUSHJ 17, (AC)
```

FLAGS

A half-word in Word 19 containing flags and fields with the meanings shown in Table 8-2.

Table 8-2
Flags and Fields in Word 19 of File Table

Bits	Meaning
0-2	Recording density 000 = system standard 001 = 200 bpi 010 = 556 bpi 011 = 800 bpi
3	Recording parity 0 = odd 1 = even
4-17	Reserved for expansion

ADDRESS OF USER-NUMBER

The address of the location containing the project-programmer number for the file.

BYTE POINTER TO SYMBOLIC KEY	A byte pointer that points to the byte preceding the first character of the symbolic key field (for key codes 0, 1, and 2) or the address of the key (for key codes 3, 4, and 5).
BYTE POINTER TO RECORD KEY	Similar to byte pointer to symbolic key, except that the pointer to the record key is relative to the beginning of the record.
CODES FOR INDEXED KEY	A word containing codes with the meanings shown in Table 8-3.

Table 8-3
Codes for Indexed Key

Bits	Meaning
0-14	Not used
15-17	Type of key 0 = nonnumeric 1 = numeric display $\leqslant$ 10 digits 2 = numeric display > 10 digits 3 = COMP $\leqslant$ 10 digits 4 = COMP >10 digits 5 = COMP-1
18-19	Not used
20	0 = field is signed 1 = field is unsigned
21-23	Not used
24-35	Size of key in bytes (for codes 0, 1, 2 only)

FILE-LIMIT PAIRS	For each file-limit, three words are allocated. The left-half of word 1 contains the address of the lower limit; the right-half of word 1 contains the address of the higher limit. Words 2 and 3 contain the actual values of these limits at OPEN time.

8.4 CHANNEL TABLES

For each open file, a 20-word entry is placed in a channel table. Table 8-4 shows the contents of each word in that entry.

Table 8-4
Channel Table Entry

Word	Bits	Contents
21		Number of bytes per word in the record area.
20		Device name in SIXBIT for RERUN.
19	0 1 2 3 4-14 15 16 17 18-35	Not used. The file is optional. Labels are nonstandard. Labels are standard. Not used. Version number discrepancy between index levels. WRITE invalid key. READ, REWRITE, or DELETE invalid key. Record size in words.
18		Number of inputs executed.
17		Number of outputs executed.
16	0-17 18-35	Buffer location. Larger of record or label size in words.
15	0-11 12-15 16 17 18-35	Magnetic tape reel number. I-O channel number. File is "locked" close. File is positioned over the read/write heads. Not used.
14		Relative number of the current physical block.
13		Number of buffers per logical block.
12		Number of buffers yet to be processed for current block.
11		Number of records required to fill current logical block.
10		IOWD pointing to device-name being used with multi-device file.
7-9		A 3-word header used for output.
4-6		A 3-word header used for input.
3		Number of records remaining until next rerun dump.
2		Current record number.
1		Characteristics of the device, as returned by a DEVCHR UUO.

8.5 BLOCKING

A file is said to be blocked when a BLOCK CONTAINS clause is part of its description; conversely, a file is unblocked when no BLOCK CONTAINS clause is specified. The number of records in a block is termed the blocking factor. The next two paragraphs show how I-O processing differs for blocked and unblocked files.

8.5.1 Reading and Writing Blocked Files

For each READ statement referencing a SEQUENTIAL blocked file, the I-O routines transfer to the record area (i.e., core memory allocated to contain the current record) the next consecutive record from the file. If a number of records, equal to the blocking factor, have been transferred since the previous block was read, the I-O routines do a physical read of the device.

When a RANDOM blocked file is read, both the blocking factor and the ACTUAL KEY are used to specify which physical segment to read and which record in that segment to transfer to the record area.

When an INDEXED blocked file is read, the SYMBOLIC KEY is used to specify the record to be transferred to the record area.

Writing SEQUENTIAL blocked files is accomplished in a manner similar to the reading of them, except that records are transferred from the record area to the buffer area.

When a RANDOM blocked file is written, the I-O routines do the following:

a. Read a physical segment whose address is determined by the blocking factor and the ACTUAL KEY, if necessary.

b. Transfer the record to a portion of that segment.

c. The segment is written either when another segment is to be read or when the file is closed.

When an INDEXED blocked file is written, the I-O routines perform the following:

a. Read, if necessary, the physical block that is to contain the record.

b. If the write is due to a REWRITE statement, the contents of the record area replace the record on the block and the block is rewritten.

c. If the write is due to a WRITE statement and the new record will fit in an unused part of the block, the contents of the record area are transferred to the block and the block is rewritten. If the new record will not fit into the block, the block is split into two more or less equal parts, the new record is added, and both blocks are written.

8.5.2 Reading and Writing Unblocked Files

When an unblocked file is either read or written, the I-O routines have complete control over the time when the actual read or write takes place. The user need not concern himself with the physical characteristics of the device; in fact, he does not have to know anything about such characteristics as block or segment sizes.

8.6 LABEL RECORDS

The term label records refers to header or trailer labels on the file. The presence or absence of label records is specified by the LABEL RECORDS clause (see Chapter 5). Their format can be standard or non-standard.

8.6.1 Standard Label Records

The standard label for DECtape and random-access devices is the directory block used by the monitor. For magnetic tape, the standard label is 80 characters in length and is written in a separate block from the data, with the same recording mode as the data. If the recording mode is ASCII, the 80-character label includes a final carriage return and line feed. Table 8-5 shows the contents of each character in a standard label for non-random-access devices.

Table 8-5
Standard Label for Nonrandom-Access Media

Characters	Contents
1-4	HDR1 = Beginning file. EOF1 = Ending file. EOV1 = Ending reel.
5-13	Value of identification.
14-21	Always spaces.
22-27	Not used.
28-31	Reel number. The first reel is always 0001.
32-41	Not used.
42-47	Creation date: two characters each for the year, month, and day, respectively.
48-80	Not used.

8.6.1.1 Ending Labels - Magnetic tapes are the only devices with ending labels. Each ending label is preceded by and followed by an end-of-file mark.

8.6.2 Non-Standard Label Records

Non-Standard labels are specified by the LABEL RECORDS clause (see Chapter 5). Similar to standard labels, they are written in a separate block on the device.

When a file is opened, the beginning non-standard label will be read (as input) or written (as output) automatically by the I-O routines. If the file is being opened for output, the data for the record must be supplied by a USE procedure in the DECLARATIVES (see Chapter 6). If the file is being opened for input, no checks are made by the I-O routines to determine the validity of the label; the user may write any check in a USE procedure.

8.7 MULTIPLE-FILE TAPE

Only magnetic tapes can contain multiple files in the COBOL sense. This may seem to conflict with the obvious fact that random-access devices contain more than one file, but the programmer will recall that files on a random-access device are treated by the monitor as though they are on separate devices. Each file on a multi-file magnetic tape must have labels, and must be wholly contained on one reel.

8.8 SAME AREA CLAUSE

The SAME AREA clause specifies that two or more files are to share the same memory area during processing. Since the area shared includes all storage areas (including alternate areas) assigned to the files specified in this clause, only one file at a time can be open.

8.9 SAME RECORD AREA CLAUSE

The SAME RECORD AREA clause specifies that two or more files are to use the same memory for processing the current logical record. All or any of the files specified in this clause may be open at any time. The record area contains only one record at any time.

8.10 FILE-LIMITS

File-limits must be specified for RANDOM files. They may also be specified for input SEQUENTIAL files, in which case only that portion of the file that is within the file-limits is read.

File-limits for files whose access mode is RANDOM specify the allowed range, or ranges, for the ACTUAL KEY (see ACTUAL KEY in Chapter 5). When the contents of the ACTUAL KEY fall outside all ranges given in the FILE-LIMITS clause for that file, the read or write transfers control to the statement specified in the associated INVALID KEY clause (see Chapter 6).

8.11 SUBPROGRAMS

The subprogram capability allows the COBOL programmer to write and compile separate programs to be executed together. Programmers who write in other languages can also make use of COBOL subprograms by calling them from their programs according to the standard calling sequence.

Subprograms enable the COBOL programmer to divide a large task into smaller and more manageable units that can be executed together. Also, if many programmers are working on portions of a COBOL job, their parts can be written as subprograms, compiled and debugged separately, and joined together at run-time. A COBOL subprogram is a program that is compiled separately but is normally run with another program that calls it. A main program, in contrast, is a program that is not called by another program and in which execution begins. The principal distinction between a main program and a subprogram (aside from the way in which the program is used) is that after compilation a main program has a starting address and a subprogram does not. The compiler, when it sees a LINKAGE SECTION, ENTRY statement, PROCEDURE DIVISION header with a USING clause, or a GOBACK statement assumes that the program is a subprogram and does not generate a starting address for it. Otherwise, it assumes that the program is intended to be used as a main program and does generate a starting address. Thus, a command of the form:

```
.R COBOL          or          .COMPILE MAIN1, SUB1
*=MAIN1
*=SUB1
```

will cause the main program to be given a starting address while the subprogram (SUB1) will not have a starting address generated for it.

There are cases when a user might want to use a main program (i.e., one that does not contain any of the special subprogram syntax) as a subprogram. In such a case, he can force the compiler to suppress the starting address by including the /I switch in the command string. For example:

```
.R COBOL          or          .COMPILE /COBOL MAIN1, MAIN2(,I)
*=MAIN1
*=MAIN2/I
```

The program to be used as a subprogram (MAIN2) will not have a starting address generated for it. (,I) is the way that the user can pass binary compiler switches such as /I to the compile-class monitor commands. Refer to DECsystem-10 Operating System Commands for more information about compile-class commands.

There may also be cases when a user wishes to use a subprogram as a main program. In this case, he can force the compiler to generate a starting address by including the /J switch in the command string. For example:

```
.R COBOL          or          .COMPILE /COBOL SUB1(,J), SUB2
*=SUB1/J
*=SUB2
```

The subprogram that will be used as a main program will have a starting address generated for it. The other subprogram (SUB2), because it contains subprogram syntax, will not have a starting address generated for it.

Subprograms can be treated as main programs as long as the data in the LINKAGE SECTION is not referred to in the PROCEDURE DIVISION. This restriction is due to the fact that the data in the LINKAGE SECTION does not have space reserved for it; instead, it uses the space reserved for data items in the DATA DIVISION of the main program. Thus, if no main program is present, the items in the LINKAGE SECTION cannot be used.

8.11.1 Using Subprograms

A subprogram is written and compiled separately from a main program, and thus has most elements of a main program. Files can be opened for input, output, and input-output in a subprogram, but no program can perform I/O operations on files in another program. The LINKAGE SECTION is included in a subprogram so that the subprogram can manipulate the data from another program without performing I/O on this data. Also, unlike a main program, a subprogram cannot be segmented.

When a program calls a subprogram, the user must designate in the CALL statement of the calling program where the subprogram is to be entered. If the CALL statement in the main program refers to the name of the subprogram (PROGRAM-ID), the subprogram is entered at the beginning of the executable code (PROCEDURE DIVISION) and execution starts at that point. Execution of the subprogram proceeds until an EXIT PROGRAM or GOBACK statement is encountered. Control then returns to the calling program at the statement following the CALL statement. If no EXIT PROGRAM or GOBACK statement is encountered, execution proceeds to the end of the subprogram (unless it calls another subprogram) and control is never returned to the calling program. If the CALL statement in the main program contains the name of an entry point in the subprogram, the subprogram is entered at the ENTRY statement naming this entry point and execution proceeds from there. Control is returned to the calling program when an EXIT PROGRAM or GOBACK statement is encountered. If neither are present, the subprogram is executed to completion (unless it calls another subprogram) and control is not returned to the calling program.

The USING clause in CALL, the PROCEDURE DIVISION header, and ENTRY need not be present if the user does not wish to pass data from the main program to a subprogram. However, if one or more are present in the ENTRY statement or PROCEDURE DIVISION header in the subprogram, an equal or

greater number of identifiers must be present in the CALL statement of the calling program. A USING clause can be present in the CALL statement in the calling program without a corresponding USING clause in the ENTRY statement or the PROCEDURE DIVISION header of the called program; it will be ignored.

8.11.2 Example of Subprogram Usage

The example below shows a main program and two subprograms. The subprograms are called if certain conditions are met in the data in the main program. From the main program, calls are made to entry points in subprogram B, which then calls subprogram C. Because the name of subprogram C is used in the CALL statements in B, the entry point is the beginning of the PROCEDURE DIVISION. Note the presence of the USING clauses with identifiers in the main program CALL statements and in the ENTRY statements and PROCEDURE DIVISION header in the subprogram. The identifiers in the subprograms are defined in their respective LINKAGE SECTIONS, while the identifiers in the main program are defined in the FILE and WORKING-STORAGE SECTIONS. The identifiers in the subprograms use the core space of the identifiers in the main program because no space is reserved for items in the LINKAGE SECTION. Thus, REC and TOT in ENTRY statements BP and BN use the core space of INR and TOTALS from the main program. Also, when subprogram C is called, the identifiers AMT, B, and T are related to items in the LINKAGE SECTIONS of subprogram B which are, in turn, related back to the identifiers in the main program.

Each subprogram has, in addition to a LINKAGE SECTION, a WORKING-STORAGE SECTION. The data items in this section are unique to each subprogram (even if they are used in calls to other subprograms) and have space reserved for them.

```
*           MAIN PROGRAM
IDENTIFICATION DIVISION.
PROGRAM-ID. A.
ENVIRONMENT DIVISION.
I-O SECTION.
FILE-CONTROL.
        SELECT INF ASSIGN DSK.
```

```
DATA DIVISION.

FILE SECTION.
FD      INF VALUE OF ID "TEST  DAT".
01      INR, DISPLAY-7.
          02 TYPE-KEY        PIC  X(4).
          02 INF-AMT         PIC  S9(5)V99.

WORKING-STORAGE SECTION.
01      TOTALS               PIC  S9(6)V99.
01      TAX                  PIC  SV999              VALUE  0.035.

PROCEDURE DIVISION.
START.
        OPEN INPUT INF.
LOOP.
        READ INF;
                AT END CLOSE INF
                DISPLAY TOTALS,
                STOP RUN.
        IF TYPE-KEY = "BP",
                CALL BP USING INR, TOTALS,
                GO TO LOOP.
        IF TYPE-KEY = "BN",
                CALL BN USING INR, TOTALS,
                GO TO LOOP.

*       SUBPROGRAM FOR TYPE B RECORDS

IDENTIFICATION DIVISION.
PROGRAM-ID. B.

DATA DIVISION.

WORKING-STORAGE SECTION.
01      DISCOUNT             PIC  SV999              VALUE  0.165.
01      NEG-DISCNT           PIC  SV999              VALUE  -0.165.

LINKAGE SECTION.
01      TOT                  PIC  S9(6)V99.
01      REC, DISPLAY-7.
          02 FILLER          PIC  X(4).
          02 AMT             PIC  S9(5)V99.

PROCEDURE DIVISION.
ENTRY BP USING REC, TOT.
        CALL C USING REC, DISCOUNT, TOT.
        EXIT PROGRAM.

ENTRY BN USING REC, TOT.
        CALL C USING REC, NEG-DISCNT, TOT.
        EXIT PROGRAM.
```

```
*       SUBPROGRAM TO DO THE ARITHMETIC
IDENTIFICATION DIVISION.
PROGRAM-ID. C.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      HOLD                PIC  S9(5)V99.
LINKAGE SECTION.
01      A, DISPLAY-7.
        02 FILLER            PIC  X(4).
        02 AMT               PIC  S9(5)V99.
01      T                   PIC  S9(6)V99.
01      B                   PIC  SV999.
PROCEDURE DIVISION USING A, B, T.
START.
        MULTIPLY AMT BY B GIVING HOLD.
        IF B > 0,
                ADD AMT TO T,
                SUBTRACT HOLD FROM T.
        IF B < 0,
                SUBTRACT AMT FROM T,
                ADD HOLD TO T.
        EXIT PROGRAM.
```

Chapter 9
Source Library Maintenance Program

The COBOL Source Library Maintenance Program maintains on disk or DECtape a library file of COBOL source-language text that can be copied into a source program at compile time. Specifically, the program has the capability of adding source-language data to the library file, replacing and/or deleting lines, and listing entries in that file. Thus it allows the user to specify those source routines that are used in many programs and to place them in a common file for use by the COBOL compiler. The routines on the library file are made available through the use of the COPY verb. (See pages 7-2 and 7-3 for the description of the COPY verb.)

9.1 EQUIPMENT

9.1.1 Machine Requirements

Required hardware for the COBOL Source Library Maintenance Program consists of the following:

a. A PDP-10 central processor.
b. A diskfile.
c. A terminal.

9.1.2 Machine Options

With a small sacrifice of running time, the original and/or updated version of the library file can be put on DECtape. Any device capable of handling ASCII output can be used as the listing device.

9.2 SALIENT FEATURES OF MAINTENANCE PROGRAM

a. The routine runs in 2K of user core.
b. Though the routine is designed around disk, other devices can be used where appropriate.

c. The routine runs under the control of BATCH.

d. Commands can be taken from any device.

9.3 INPUT FORMAT

The input file is a collection of COBOL source-language routines, each identified by a unique 8-character library-name. The library file must be on a directory device.

The data in the library is divided into the three sections shown in Table 9-1.

Table 9-1
Data Sections in the Library File

Section	Description
Source Language	This is a collection of named COBOL routines (in ASCII) that are to be referenced by the COPY verb. The routines are arranged in alphabetic order by library-name. The maximum number of routines that can appear in the file is 3969.
Fine Table	This is a table of library-names. It may extend over as many as 63 blocks. Each entry has a pointer to the data in the source language section.
Rough Table	This table points to the blocks in the Fine Table. It is one block (126 words) in length.

9.4 OUTPUT FORMAT

The format of the output files is identical to that of the input file. (See Section 9.3.)

9.5 ORGANIZATION OF THE MAINTENANCE PROGRAM

9.5.1 Internal Organization

All subroutines are resident. The Maintenance program uses the core UUO to increase its core usage, if necessary.

9.5.2 Operational Organization

The Maintenance program copies the input file to disk, updating as it does. When the update is completed, the new library is copied to the output file; however, if the output is to disk, the Maintenance program does a RENAME.

9.6 OPERATING PROCEDURE

9.6.1 Start-Up

As one of the CUSPs, the Maintenance program is started with the R, RUN, or GET and START monitor commands. There is no REENTER procedure.

The operator specifies the devices to be used by typing

FILE1,FILE2 = FILE3

where FILEn is of the form DEV:NAME.EXT [PROJ,PROG].

FILE1 is the file to contain the output, FILE2 is the listing file, and FILE3 contains the library to be updated.

9.6.2 Default Assignments

If FILE2 is not specified, no listing of corrected routines is produced.

If FILE3 is not specified, it is assumed that there is no input library; in this case, only insertions can be made.

If filename extensions are not specified, the following extensions are assigned:

a. LIB for FILE1 and FILE3, and
b. LST for FILE2.

If devices are not specified, DSK is assigned.

If a filename is not specified for FILE3, LIBARY is assumed; if no filename is specified for FILE1 or FILE2, the filename of FILE3 is assumed.

If the input file and the output file have the same filename and extension, and are both on the disk, the extension of the input file is changed to BAK at the completion of the run.

9.6.3 Switches

The following switches may be used:

Z - Clear an output directory (for DECtape only).

W - Rewind (listing on magtape only).

See Appendix D, Table D-3 for a complete list of COBOL switches.

9.6.4 Listing the Contents of a Library File

The user can obtain a listing of the contents of a library file by means of the following command string.

```
.R LIBARY
*libnam.LST←libnam.ext /L
```

Libnam.LST is a disk file that contains a listable form of the library file named libnam.ext. The /L switch causes LIBARY to create a listing file, which is stored in libnam.LST. This listing can be printed by means of any monitor command available for this purpose (e.g., LIST or TYPE).

9.7 COMMAND LANGUAGE

9.7.1 Commands to Position Files

The six commands used to position the input and scratch files are shown in Table 9-2.

Table 9-2
Commands for Positioning Files

Command	Function
INSERT library-name	The input file is copied to the scratch file, starting at the current position of the files, until a source routine with a name alphabetically greater than the one specified is encountered. The new name is inserted in the Fine Table, and the program awaits another command.
INSERT library-name, dev:file.ext [ppn]	The entire file is inserted into the library with the name indicated by library-name. The file must be ASCII. If there are line numbers in the file, they are included in the file. If there are no line numbers, they are added to the lines, starting with 10 and incrementing by 10. These line numbers are not COBOL sequence numbers; they are the line numbers created by the LINED program.
DELETE library-name	The input file is copied to the scratch file until the source routine with the name specified is encountered. The input file is then positioned after that source routine.
REPLACE library-name	The program does a DELETE followed by an INSERT.

Table 9-2 (Cont)
Commands for Positioning Files

Command	Function
REPLACE library-name, dev:file.ext [ppn]	The file named library-name is replaced with the specified file. The new file must be ASCII. If there are line numbers in the file, they are included in the file. If there are no line numbers in the file, they are added to the lines, starting with 10 and incrementing by 10.
CORRECT library-name	The input file is copied to the scratch file until a source routine with the name specified is encountered. Typing /N after the CORRECT command causes new line numbers to be applied to the output version of the source language routine.
END	The remainder of the input file is copied to the scratch file, and the output file is created, and the program then terminates.
RESTART	The remainder of the input file is copied to the scratch file. The scratch file then becomes the input file, and a new scratch file is started. This command allows the user to update routines out of library-name order.
EXTRACT library-name dev:file.ext [ppn]	A file with the specified name and extension on the specified device is created from the file named library-name. If the /N switch is included after the file descriptor, line numbers are put on the lines of the output file. If the /N switch is not included, the file will not have line numbers.

9.7.2 Commands to Alter Contents of Source File

The three commands used to alter the contents of a source file are shown in the following table.

Table 9-3
Commands for Altering Contents of Source File

Command	Function
Dnnnnnn	The input file is copied to the scratch file until nnnnnn, the specified line, is encountered. That line is then skipped.
Innnnnn COBOL-statement	The input file is copied until either a line having a larger line-number or a new source language routine is encountered. The COBOL-statement is inserted at that point. A space or tab must be included between the line number and the COBOL statement. The space will not be included in the statement; the tab will be included in the statement.
Rnnnnnn COBOL-statement	The input file is copied until the specified line is encountered. The COBOL statement with that line-number is replaced by the statement in the command. A space or tab must be included between the line number and the COBOL statement. The space will not be included in the statement; the tab will be included in the statement.

9.7.3 Example of the Use of the Commands

A library on disk contains the routines PAYCOMP, FIND-MP, and MP-DESCR. The user wants to do the following:

a. Delete PAYCOMP.

b. Correct MP-DESCR.

c. Insert a new routine called JOB-DESC.

The MP-DESCR routine contains the following source statements:

```
000010      LABEL RECORDS ARE OMITTED
000020      DATA RECORD IS MP-RECORD.
```

The dialog at the Teletype might appear as follows:

```
R LIBARY
LIBARY.NEW=LIBARY.OLD
INSERT                  JOB-DESC
I000010                 LABEL RECORDS ARE STANDARD;
I000020                 VALUE OF ID IS "JOBS  DAT";
I000030                 DATA RECORD IS JOB-RECORD.
CORRECT                 MP-DESCR/N
I000005                 BLOCK CONTAINS 5 RECORDS
DELETE                  PAYCOMP
END
```

The file LIBARY.NEW now contains the following:

a. FIND-MP

b. JOB-DESC

c. MP-DESCR, altered to appear as follows:

```
000010        BLOCK CONTAINS 5 RECORDS
000020        LABEL RECORDS ARE OMITTED
000030        DATA RECORD IS MP-RECORD.
```

To insert one or more files in a library, the user can issue the following commands to LIBARY.

```
.R LIBARY
*ALIB,ALIB=
*INSERT AFIL,AFIL
*INSERT BFIL,BFIL
*END

*↑C
```

The file ALIB.LIB contains the following files.

```
A F I L                 COBOL LIBRARY              26-JUN-74        09:52
000010          DISPLAY "A".



B F I L                 COBOL LIBRARY              26-JUN-74        09:52
000010          DISPLAY "B".
```

9.8 ERROR RECOVERY

9.8.1 Input Errors

The input file is not a library file if one of the following conditions exists:

a. The Rough Table is not in order by library-name.
b. A Fine Table is not in order by library-name.
c. A library routine is not in order by line-number.

If the input file is not a library file, the program types

?INCORRECT LIBRARY FILE FORMAT

and terminates with a CALL [SIXBIT/EXIT/].

9.8.2 Operator Errors

If an improper command is detected, an error message is typed, and the program looks for another command. For a complete list of operator error messages see Table 9-4.

Table 9-4
Operator Errors

Message	Meaning
?THAT ROUTINE HAS ALREADY BEEN PASSED	An attempt was made to alter a routine that had already been copied to the output file.
?THAT ROUTINE DOES NOT EXIST	An attempt was made to correct, delete, or replace a routine not in the input file.
?THAT ROUTINE ALREADY EXISTS	An attempt was made to insert a routine that already exists in the input file.
?THAT LINE HAS ALREADY BEEN PASSED	An attempt was made to insert, delete, or replace a source line that had already been copied to the scratch file.
?THAT LINE DOES NOT EXIST	An attempt was made to replace or delete a line not found in the source routine.
?THAT LINE ALREADY EXISTS	An attempt was made to insert a line that already exists in the source routine.
?IMPROPER LIBRARY-NAME	The specified library-name is longer than 8 characters or contains characters other than A-Z, 0-9, and the hyphen.
?IMPROPER COMMAND	A command was issued that was not recognized by LIBARY.

9.8.3 Hardware Errors

If an error is detected while reading or writing on a device, the program types

?ERROR ON FILE DEV: filename, extension

If the operator wants the read or write re-tried, he types

AGAIN

9.9 SOFTWARE INTERFACES

9.9.1 Format of the Rough Table

The Rough Table consists of a single block of 128 words. The first word is unused; the last word contains all 1s. The intervening space is divided into sixty-three 2-word entries. The format of an entry is shown in Table 9-5.

Table 9-5
Format of a Rough Table Entry

Word	Bits	Contents
1 2	0-35 0-11	The routine name (in SIXBIT) that is found in the first entry of a Fine Table.
2	12-28	The block-number of the Fine Table block that contains the routine name mentioned above. The block-number is relative to the beginning of the file.
2	29-35	Not used.

9.9.2 Format of the Fine Table

The Fine Table consists of one or more blocks of 128 words. In each block, the first word is not used, and the last word is all 1s. The intervening space is divided into sixty-three 2-word entries. The format of an entry is shown in Table 9-6.

Table 9-6
Format of a Fine Table Entry

Word	Bits	Contents
1 2	0-35 0-11	The name of the routine in the library (in SIXBIT).
2	12-28	The block-number that contains the start of the first line of the source routine mentioned in the preceding entry. The block-number is relative to the beginning of the file.
2	29-35	The relative word-number within that block which contains the sequence number of the first line of that routine.

9.9.3 Format of the Source Routines

Source routines contain lines of COBOL source language. Each line has a line-number word, followed by a string of ASCII. The line-number word has a line number, or sequence number, in bits 0-34 and a 1 in bit 35. Any space left between the last character of a line and the following line-number is filled with nulls. The last line of a routine is followed by a line with a sequence number of all 1s.

Appendix A
COBOL Reserved Words

In the listing below, words preceded by no symbols are standard COBOL reserved words that are also reserved in DECsystem-10 COBOL. Words preceded by * are ANSI standard reserved COBOL words that are not reserved in DECsystem-10, but should be avoided for compatibility with other COBOL compilers. Words preceded by ** are reserved in DECsystem-10 COBOL but not in the ANSI standard.

A

ACCEPT
ACCESS
ACTUAL
ADD
*ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
**ANY
ARE
AREA
AREAS
ASCENDING
**ASCII
ASSIGN
AT
AUTHOR

B

BEFORE
BEGINNING
**BINARY
BLANK
BLOCK
BY

C

**CALL
**CANCEL
CD
CF
CH
**CHANNEL
CHARACTERS
CLASS
*CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMMUNICATION
COMP
**COMP-1
**COMPILE
COMPUTATIONAL
**COMPUTATIONAL-1
COMPUTE
CONFIGURATION
**CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
**CURRENT

D

DATA
**DATABASE-KEY
DATE
DATE-COMPILED
DATE-WRITTEN
**DBKEY
DE
DECIMAL-POINT
DECLARATIVES
**DECSYSTEM-10
**DECSYSTEM10
**DEFERRED
**DELETE
DELIMITED
DELIMITER
**DENSITY
DEPENDING
DEPTH
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
**DISPLAY-6
**DISPLAY-7
DIVIDE
DIVISION
DOWN
**DUP
**DUPLICATE

E

**EBCDIC
EGI
ELSE
EMI
**EMPTY
ENABLE
END
ENDING
ENTER
**ENTRY
ENVIRONMENT
**EPI
EQUAL
EQUALS
ERROR
ESI
**EVEN
EVERY
EXAMINE
**EXCL
**EXCLUSIVE
EXIT

F

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
**FIND
FIRST
FOOTING
FOR
**FORTRAN-IV
**FORTRAN
FROM

G

GENERATE
**GET
GIVING
GO
**GOBACK
GREATER
GROUP

H

HEADING
HIGH-VALUE
HIGH-VALUES

I

I-O
I-O-CONTROL
**ID
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
**INSERT
INSTALLATION
INTO
INVALID
**INVOKE
IS

J

JUST
JUSTIFIED

K

KEY
KEYS

L

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
**LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

M

**MACRO
**MEMBER
MEMORY
MESSAGE
MODE
**MODIFY
MODULES
MOVE
MULTIPLE
MULTIPLY

N

NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC

O

OBJECT-COMPUTER
OCCURS
**ODD
OF
OFF
OMITTED
ON
**ONLY
OPEN
OPTIONAL
OR
OUTPUT
OVERFLOW
**OWNER

P

PAGE
PAGE-COUNTER
**PARITY
**PDP-10
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
**PRIOR
**PRIVACY
PROCEDURE
PROCEED
PROCESSING
**PROGRAM
PROGRAM-ID
**PROT
**PROTECTED

Q

QUEUE
QUOTE
QUOTES

R

RANDOM
RD
READ
RECEIVE
RECORD
**RECORDING
RECORDS
REDEFINES
REEL
**RELATIVE
RELEASE
REMAINDER
REMARKS
**REMOVE
RENAMES
REPLACING

REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
**RETR
**RETRIEVAL
RETURN
*REVERSED
REWIND
**REWRITE
RF
RH
RIGHT
ROUNDED
RUN
**RUN-UNIT

S

SAME
**SCHEMA
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT
SEGMENT-LIMIT
SELECT
**SELECTIVE
SEND
SENTENCE
SEQUENTIAL
SET
**SETS
*SIGN
**SIXBIT
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
**STORE
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
**SUB-SCHEMA
SUBTRACT
SUM
**SUPPRESS
**SWITCH
SYMBOLIC
SYNC
SYNCHRONIZED

T

TABLE
TALLY
TALLYING
TAPE
TERMINAL
TERMINATE
TEXT
THAN
THROUGH
THRU
TIME
TIMES
TO
**TODAY
**TRACE
TYPE

U

UNIT
UNSTRING
UNTIL
UP
**UPDATE
**UPDATES
UPON
USAGE
**USAGE-MODE
USE
**USER-NUMBER
USING

V

VALUE
VALUES
VARYING

W

WHEN
WITH
**WITHIN
WORDS
WORKING-STORAGE
WRITE

Z

ZERO
ZEROES
ZEROS

Appendix B
Character Collating Sequence

The following table gives the collating sequences for ASCII (DISPLAY-7) and SIXBIT (DISPLAY-6) fields as used in condition comparisons.

SIXBIT	Character	ASCII 7-Bit	SIXBIT	Character	ASCII 7-Bit	Character	ASCII 7-Bit
00	Space	040	40	@	100	`	140
01	!	041	41	A	101	a	141
02	"	042	42	B	102	b	142
03	#	043	43	C	103	c	143
04	$	044	44	D	104	d	144
05	%	045	45	E	105	e	145
06	&	046	46	F	106	f	146
07	'	047	47	G	107	g	147
10	(	050	50	H	110	h	150
11	)	051	51	I	111	i	151
12	*	052	52	J	112	j	152
13	+	053	53	K	113	k	153
14	,	054	54	L	114	l	154
15	-	055	55	M	115	m	155
16	.	056	56	N	116	n	156
17	/	057	57	O	117	o	157
20	0	060	60	P	120	p	160
21	1	061	61	Q	121	q	161
22	2	062	62	R	122	r	162
23	3	063	63	S	123	s	163
24	4	064	64	T	124	t	164
25	5	065	65	U	125	u	165
26	6	066	66	V	126	v	166
27	7	067	67	W	127	w	167
30	8	070	70	X	130	x	170
31	9	071	71	Y	131	y	171
32	:	072	72	Z	132	z	172
33	;	073	73	[	133	{	173
34	<	074	74	\	134	|	174
35	=	075	75	]	135	}	175
36	>	076	76	↑	136	~	176
37	?	077	77	←	137	Delete	177

Appendix C
Standard Calling Sequence

The standard calling sequence is used when COBOL programs call COBOL subprograms by means of the CALL statement and when COBOL programs call MACRO, FORTRAN-10, or FORTRAN-IV subroutines by means of the ENTER verb. Note that the calls to subroutines written for the new FORTRAN-10 compiler use the standard calling sequence, while calls to subroutines written for the old FORTRAN-IV compiler use another calling sequence. Both are described below.

The calling sequence, while it may be of interest to COBOL, FORTRAN-10, and FORTRAN-IV programmers, is required knowledge only for the programmer who wishes to write MACRO subroutines. The COBOL, FORTRAN-10, and FORTRAN-IV programmer need only use the statements in COBOL, FORTRAN-10, and FORTRAN-IV that define entry points in the subprograms, the compiler will handle entry and return.

When calling subprograms written in languages other than COBOL, the programmer must take care that the input/output operations are performed only in the main program or only in the subroutine written in another language. This restriction is necessary because the I/O channels cannot be used by two languages in the same job.

C.1 CALLING SEQUENCE FOR COBOL, MACRO, AND FORTRAN-10

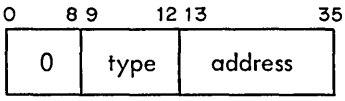
The calling sequence used when a CALL statement calls a COBOL subprogram or an ENTER statement calls a MACRO or FORTRAN-10 subroutine is:

```
MOVEI 16, address of first entry in argument list
PUSHJ 17, subprogram address
```

If the USING clause appears, an argument list is created containing an entry for each identifier or literal in the order of appearance in the USING clause. It is preceded by a word containing, in its left half, the negative of the number of entries in the list. If no USING clause is present, the

(continued on next page)

argument list contains an empty word and the preceding word is set to 0. Each entry in the list is one 36-bit word of the form:



Bits 0-8 are always 0.

Bits 9-12 contain a type code that indicates the USAGE of the argument.

Bits 13-35 contain the address of the argument or the first word of the argument; the address can be indexed or indirect.

The types, their codes, how the codes appear in the argument list, and the locations specified by the addresses are described below.

a. For 1-word COMPUTATIONAL items

CODE: 2
IN ARGUMENT LIST: XWD 100, address
ADDRESS: that of the argument itself

b. For 2-word COMPUTATIONAL items

CODE: 11
IN ARGUMENT LIST: XWD 440, address
ADDRESS: that of the high-order word of the argument

c. For COMPUTATIONAL-1 items

CODE: 4
IN ARGUMENT LIST: XWD 200, address
ADDRESS: that of the argument itself

d. For DISPLAY-6 and DISPLAY-7 items

CODE: 15
IN ARGUMENT LIST: XWD 640, address
ADDRESS: that of a 2-word descriptor for the argument

WORD1: a byte pointer to the identifier or literal

WORD2: bit 0 is 1 if the item is numeric
bit 1 is 1 if the item is signed
bit 2 is 1 if the item is a figurative constant (including ALL)
bit 3 is 1 if the item is a literal
bits 4 through 11 are reserved for expansion
bit 12 is 1 if the item has a PICTURE with one or more P's just before the decimal point (e.g., 99PPV)
bits 13 through 17 are the number of decimal places. If bit 12 is 1, this is the number of P's.
bits 18 through 35 contain the size of the item in bytes.

e. For procedure-names (which cannot be used for calls to COBOL subprograms)

CODE: 7
IN ARGUMENT LIST: XWD 340, address
ADDRESS: that of the procedure

C.1.1 Example of the Standard Calling Sequence

```
.
.
.
77 FIELD1 PICTURE S9(6) COMP.
77 FIELD2 USAGE INDEX.
77 FIELD3 PICTURE S9(15) COMP.
77 FIELD4 PICTURE XX; DISPLAY-7.
77 FIELD5 PICTURE 99V9.

01 DUMMY.
    02 FIELD6 OCCURS 3 TIMES INDEXED BY FIELD7.
    .
    .
    .
ENTER MACRO ROUTIN USING FIELD1, FIELD2, FIELD3, FIELD4, FIELD5, FIELD7.
```

The preceding coding will generate:

```
MOVEI       16, %LIT00+60
PUSHJ       17, ROUTIN
        .
        .
        .
%LIT00+21 / POINT     7,FIELD4
            XWD       0,2
%LIT00+23 / POINT     6,FIELD5
            XWD       400001,3
        .
        .
        .
%LIT00+57 / XWD      -6, 0
%LIT00+60 / XWD       0, FIELD1
            XWD       0, FIELD2
            XWD       440, FIELD3
            XWD       640, %LIT00+21
            XWD       640, %LIT00+23
            XWD       0, FIELD7
```

C.2 CALLING SEQUENCE FOR FORTRAN-IV

The calling sequence used when an ENTER statement calls a FORTRAN-IV subroutine is:

```
JSA 16, subroutine address
```

If the USING clause appears, a single parameter for each identifier or literal follows the subroutine linkage. (ARG is a PDP-10 instruction that does nothing.)

a. For 1-word COMP, index data items, and index-names:

   ARG 0, identifier

b. For 2-word COMP:

   ARG 11, identifier

c. For COMP-1:

   ARG 2, identifier

d. For DISPLAY-6 or DISPLAY-7:

   ARG 10, parameters

   Parameters is the location of the first two consecutive words describing the item:

   WORD1: a byte pointer to the identifier or literal

   WORD2: bit 0 is 1 if the item is numeric
   bit 1 is 1 if the item is signed
   bit 2 is 1 if the item is a figurative constant (including ALL)
   bit 3 is 1 if the item is a literal
   bits 4 through 11 are reserved for expansion
   bit 12 is 1 if the item has a PICTURE with one or more P's just before the decimal point (e.g., 99PPV)
   bits 13 through 17 are the number of decimal places. If bit 12 is 1, this is the number of P's.
   bits 18 through 35 contain the size of the item in bytes.

e. For Procedure-names:

   ARG 17, procedure-name

C.2.1 Example of the Calling Sequence for FORTRAN-IV

```
PROCEDURE DIVISION.
ONLY SECTION.
PARA-NAME. ENTER FORTRAN-IV ROUTIN.
    .
    .
    .
```

The preceding will generate:

```
JSA     16,ROUTIN
```

Appendix D
Command Strings

The general form of the command string is as follows:

RELFIL,LSTFIL=SRC1,SRC2,...

Table D-1
Explanation of Command String Terms

Term	Meaning
RELFIL	The file that is to hold the generated code. If no generated code is desired, the descriptor for RELFIL is replaced by a hyphen. Example: -,LSTFIL=SRC1,SRC2...
LSTFIL	The file that is to hold the generated listing. If no listing is desired, the descriptor for LSTFIL is replaced by a hyphen. Example: RELFIL,-=SRC1,SRC2,...
SRC1,SRC2	The source files required to form one input program.

Each file description has the following form:

DEVICE:FILE.EXT [PROJECT,PROGRAMMER]/SWITCH

Certain default assignments are made by the compiler whenever terms are omitted from the command strings or the file descriptions.

a. If the device is omitted in any output file description, DSK is assumed. If the device is omitted in an input file description, either the preceding device or DSK (if no preceding device is specified) is assumed.

b. If the filename for RELFIL and/or LSTFIL is omitted, the filename of the first source file is used.

Table D-2
Explanation of File Description Terms

Term	Meaning
DEVICE	The name of a device. The name is composed of 6 or fewer letters and/or digits.
FILE	The name of a file. The name is composed of 6 or fewer letters and/or digits.
EXT	The filename extension. It is composed of 3 or fewer letters and/or digits.
PROJECT	A user's project number.
PROGRAMMER	A user's programmer number.
SWITCH	Any of the switches shown in Table D-3.

c. If the filename extension is omitted from RELFIL, .REL is assumed; if it is omitted from LSTFIL, .LST is assumed; if omitted from the source file descriptor, CBL is assumed; and if omitted from the library file descriptor, LIB is assumed.

d. If the [PROJECT,PROGRAMMER] option is omitted on any file, the file area for the user is used.

Examples:

```
DTA1:RELOUT.A/Z,LPT:=DSK:SRCIN.C [27,36] /M / S
```

The compiler compiles the program found in the file SRCIN.C in the area reserved for project-programmer 27,36. It treats columns 1-6 of the source as a sequence number (/S). The generated code is written on DTA1, after the directory is cleared (/Z). The listing, including maps (/M) is put on the LPT.

```
=LIB1/L,PROG/A
```

The compiler compiles the program found in PROG.CBL (CBL is assumed because the filename extension is omitted from the source file descriptor) on the disk, using LIB1.LIB whenever a COPY verb is seen (/L). The generated code goes into the file DSK:PROG.REL, and the listing onto the file DSK:PROG.LST. The generated code is listed (/A).

- LIB1/L,PROG/A

This is identical to the preceding example, with the exception that no generated code is produced because the file descriptor for the file has been replaced by a hyphen.

Table D-3
COBOL Switch Summary

Switch	Action by Compiler
A	Allows the listing of the code generated (the source program is listed whenever a listing file is specified).
C	Produces a cross-reference listing of all user-defined items in the source program.
E	Checks the program for errors but does not generate code.
H	Types a description of COBOL command strings and lists the switches. When this switch is used, the other parts of the command string are ignored.
I	Forces the compiler to suppress generation of a starting address for a main program.
J	Forces the compiler to generate a starting address for a subprogram.
L	Uses the preceding file descriptor as a library file whenever it encounters the COPY verb. 1. This switch is legal only with source files. 2. The file descriptor is not part of the main program. 3. More than one descriptor may have the /L switch. If the first source file is not a library file, the file LIBARY.LIB is used (if present on the DSK) until the /L file is described.
M	Prints a map showing the parameters of each user-defined item (e.g., data-names and procedure-names).
N	The source errors are not typed on the user's terminal.
P	Indicates production mode. Trace calls are not generated and user symbols are suppressed.
R	A two-segment object program is produced. The high segment will contain the resident sections of the Procedure Division; the low segment will contain all else. When the object program is loaded with the linking loader, LIBOL will be added to the high segment.
S	The source file is in conventional format (with sequence numbers in columns 1-6 and with comments starting in column 73).
W	Rewinds the device before reading or writing. (This is valid for magnetic tape only.)
Z	Clears the directory of the device before writing. (This is valid for output DECtape only.)

Appendix E
The SORT Program

SORT is a stand-alone program that sorts files according to user-specified keys. The keys are sorted in either ascending or descending order on numeric or alphanumeric data. The command string to SORT has the form:

dev:outfil.ext/sw_1/sw_2.../sw_n=dev:infil.ext/sw_1/sw_2.../sw_n

where: dev is the name of a device,
outfil.ext specifies the name and extension of the output file,
infil.ext specifies the name and extension of the input file, and
/sw_1/sw_2.../sw_n are switches for both the input and output files. They are listed in Table E-1.

If a file is contained on more than one device (i.e., a multi-reel file), the devices must be magnetic tapes. Up to six devices can be specified for each file. The device names for a multi-reel file are written in the command string as follows:

dev_1:...dev_n:file.ext

When a device is not specified for either file, DSK is assumed. No assumptions are made if filenames or extensions are omitted. The default conditions for the switches are described with the switches.

The command string can be continued on more than one line by means of a hyphen at the end of the line.

The user can place several command strings into a command file and specify the file when running SORT by means of the following command:

@dev:file.ext [p,pn]

where: file.ext is the name of the file in which the commands are stored. If the extension is omitted, .CCL is assumed. If the project-programmer number is omitted, that of the user running the SORT program is assumed.

At least three scratch devices are used by the SORT program. The user can specify one or more of these scratch devices by means of the /T switch (refer to Table E-1). If none are specified by the user, the SORT program assumes DSK.

Table E-1
Summary of SORT Switches

Switch	Meaning
/A	The file is recorded in ASCII mode.
/Bn	A block contains n records; n is a decimal number. If the /B switch is omitted, it is assumed that the file is unblocked.
/Kabcm.n	/K defines the sort key according to the following parameters: a = S The field has an operational sign. a = U The field has no operational sign; its magnitude is used. If this parameter is omitted, a numeric field is assumed to have an operational sign (S). b = X The field is alphanumeric. b = C The field is COMPUTATIONAL. b = F The field is COMPUTATIONAL-1 (floating point). b = N The field is numeric display. If this parameter is omitted, the field is assumed to be alphanumeric if the sign parameter (a) is also omitted. If this parameter is omitted and the sign parameter is included, the field is assumed to be numeric display (b=N). Data formats are described in Chapter 5, the USAGE clause. c = A The field is to be sorted in ascending order. c = D The field is to be sorted in descending order. If this parameter is omitted, the field is sorted in ascending order. m is the starting byte or position of the field (e.g., the starting column on a card). n is the size of the field in either bytes or digits, depending on the context. More than one key can be entered with the /K switch, providing the keys are separated from one another by commas (e.g., /Kabcm.n,abcm.n...). The keys are sorted in the order that they are entered in the command string.
/Lam	/L specifies the labeling convention. a = S The labels are standard. a = O The labels are omitted. a = N The labels are nonstandard. m specifies the size of a nonstandard label in bytes.

Table E-1 (Cont)
Summary of SORT Switches

Switch	Meaning
/Lam (Cont)	If the /L switch is omitted, it is assumed that the labels are omitted unless a file name is specified or a directory device is used. In the latter cases, standard labels are assumed.
/Rm	/R indicates the record size, where m is the size of the largest record in bytes.
/S	The file is recorded in SIXBIT mode.
/Tdev or /Tdev:	/T indicates that the specified device is to be used as a scratch device during the sort. More than one device can be specified, providing the devices are separated by commas (e.g., /Tdev$_1$,dev$_2$...).

When neither the /A switch nor the /S switch is specified for the input file, the file is assumed to be recorded in SIXBIT mode if there are any COMP or COMP-1 keys; otherwise, it is assumed to be recorded in ASCII mode. For the output file, if neither recording mode switch is included, the recording mode of the input file is assumed.

The /K, /R, and /T switches can be specified with either the input or output file. The other switches must follow the extension of the file to which they pertain.

If the record size (/R switch) is not specified in the command string, the SORT program types the following message and waits for the user to enter the size of the largest record.

```
RECORD SIZE:
```

If the /L switch indicates that labels are standard (/LS) and a file name is not specified, SORT types one of the following messages and waits for the user to enter a filename and extension.

```
INPUT FILE NAME:
OUTPUT FILE NAME:
```

E.1 SORT EXAMPLES

The command string below causes the sorting of a file of 80-character records. Both the input file and the output file are unblocked. The input file is on MTA1. The sorted output is to be placed on MTA0. Labels are omitted on the input file and are standard on the output file. The input file is recorded in ASCII mode and the output file will also be recorded in ASCII mode. Two keys are specified: an 8-character key in columns 9 through 16 and a 1-character key in column 2.

```
.R SORT
*MTA0:SORTED.OUT/R80=MTA1:/LO/KX9.8/KX2.1/A
```

The following command string causes sorting of a file which contains 40-word records (240 bytes). Both files are on DSK, both are recorded in SIXBIT mode, and standard labels are assumed for both files. The input file has three records per block; the output file is unblocked. The keys to be sorted are: an unsigned 8-digit COMP field in word 2, a signed 12-digit COMP field in words 5 and 6, and an 18-byte alphanumeric field in words 10 through 12. The command string is continued on another line.

```
.R SORT
*SORTED.OUT/KUC7.8,C25.12,55.18=SORT.IN-
/B3/R240
```

Appendix F
COBOL Report Generator (COBRG)

COBRG is a program that produces COBOL source programs that generate reports. COBRG consists of two parts: a basic report-writing program in unspecialized form; and a preprocessor that specializes the basic program according to user-defined parameter values. One or more COBOL source programs are generated by COBRG. These programs must then be compiled and executed to produce the desired reports. The name of each program is defined by the user in his input specifications.

Certain processes are always performed in a report program. An input file is read, certain fields are moved to an output record, the contents of some fields are accumulated to produce totals, and the totals are printed at various times throughout the report. The programs produced by COBRG perform these functions for the user, thus relieving him of the necessity of coding each report program himself.

The following terms are used in the descriptions of the input specifications:

FIELD	Any contiguous string of characters
BREAK	A field is said to break when the contents of that field in the current record differ from the contents of that field in the preceding record
ACCUMULATOR	A numeric item in which the contents of an input field are added to produce a total for one or more records
HEADING	The line at the top of each printed page in the report that usually specifies the contents of each column of the report
DETAIL LINE	A line of printed output that lists some field from the input records. If any detail lines are printed, there is one line for each input record
TOTAL LINE	A line of printed output in which the value of accumulated totals is listed

F.1 INPUT TO COBRG

Input to COBRG consists of a file containing one or more sets of specification lines that describe the report or reports to be produced. Each parameter in a specification line is separated from the others

by a single comma; each line is terminated by a carriage return. While most parameters are required in the specification lines, some can be omitted. If they are omitted from the middle of the line, a comma should be entered to denote this fact. If they are omitted from the end of the line, no designation is required.

Certain COBRG reserved words cannot be used as data names in the specifications. Refer to Section F.6 for a complete list of these words.

The specifications for each program to be generated must always begin with a NAME specification. Each program's specifications are terminated by the NAME specification for another program or by an end-of-file condition on the input file. The specifications should be entered in the order that the formats are listed below.

F.1.1 NAME Specification

The NAME specification line contains the parameters that specialize the IDENTIFICATION, DATA, and ENVIRONMENT DIVISIONS of the report generator. A NAME specification must be the first specification for each set of specifications. The parameters are as follows:

Parameter 1	Must always be N to indicate NAME.
Parameter 2	The program identification; can be up to six characters in length.
Parameter 3	The blocking factor for the input data file; if omitted, it is assumed that the file is unblocked.
Parameter 4	The device for the input data file; if omitted, DSK is assumed.
Parameter 5	The device for the output (printing) file; if omitted, DSK is assumed.
Parameter 6	The value of identification of the input data file (filename and extension without the separating period); this parameter can be omitted if the input is from a nondirectory device.
Parameter 7	The value of identification of the output file; this parameter can be omitted if the output is to a nondirectory device.
Parameter 8	The date the specifications were written; can be up to nine characters in length. This parameter can be omitted.
Parameter 9	The author; can be up to 20 characters in length. This parameter can be omitted.

F.1.2 BREAK Specification

The BREAK specification defines one field of the input record that is to be tested for breaks. There is one BREAK specification for each field to be tested. Up to 25 BREAK lines can be entered. The parameters are as follows:

Parameter 1	Always B to indicate BREAK.
Parameter 2	The break level; can be any digit from 0 to 9 or F (for final). 0 has the lowest priority; 9 has the highest priority. More than one field can have the same priority number, in which case a break occurs whenever any of those fields break.

Parameter 3	The number of lines to space before printing the total. Up to nine lines can be specified; 0 signifies no spacing.
Parameter 4	The printer control channel to which a skip is made before the total is printed. This can be in the range 0 through 8; 0 signifies no skipping.
Parameter 5	The number of lines to space after the total is printed; 0 means no spacing.
Parameter 6	The printer control channel to which a skip is made after the total is printed; 0 means no skipping.
Parameter 7	The size of the break field expressed as characters or digits.
Parameter 8	The name of the break field. The field must be in the input record description.

F.1.3 HEADER Specification

The HEADER specification gives the heading that will be placed on the top of each page of the report. Two HEADER lines can be entered; the first specifies the heading for print columns 1 through 60, and the second specifies the heading for print columns 61 through 132. Parameters 2 through 5 and parameter 7 should not be included on the second specification line. The parameters are as follows:

Parameter 1	Always H to indicate HEADER.
Parameter 2	The size of the field that will contain the page number in the heading line.
Parameter 3	The starting column for the page number in the heading line.
Parameter 4	The number of lines to space after printing the heading line; 0 indicates no spacing.
Parameter 5	The printer channel to which a skip is made after the heading line is printed; 0 indicates no skipping.
Parameter 6	The data to be placed in columns 1 through 60 or columns 61 through 132 of the heading.
Parameter 7	The break priority numbers at which the page number is to be reset to 1; the page number is always 1 on the first page. This parameter can be omitted.

F.1.4 ACCUMULATOR Specification

The ACCUMULATOR specification determines which fields are to be accumulated. Up to 10 ACCUMULATOR specifications can be included. The parameters are as follows:

Parameter 1	Always A to indicate ACCUMULATOR.
Parameter 2	A unique digit that specifies the accumulator.
Parameter 3	The size of the accumulator in digits.

Parameter 4	The number of decimal places in the accumulator.
Parameter 5	The name of the field in the input record which is to be accumulated; up to 24 characters can be written.
Parameter 6	The name of the field in the printer record into which the input item is to be moved for listing on detail lines. If the item is not to be listed on detail lines, this parameter should be omitted.

F.1.5 TOTAL Specification

The TOTAL specification determines which totals are to be printed and at which breaks they are to be printed. Up to 50 TOTAL specifications can be included. The parameters are as follows:

Parameter 1	Always T to indicate TOTAL.
Parameter 2	The number of the accumulator that is to be put into an output field.
Parameter 3	The name of the output field into which the accumulator is to be moved; up to 24 characters can be written.
Parameter 4	The break numbers at which the accumulator is to be printed; these numbers should not be separated by commas. F signifies the final break.

F.1.6 LIST Specification

The LIST specification determines which items are to be printed on detail lines. Up to 50 LIST specifications can be included. The parameters are as follows:

Parameter 1	Always L to indicate LIST.
Parameter 2	A break priority number; the field will be moved to the detail line immediately following the total lines after a break at this level. A signifies that the item is to appear on all detail lines.
Parameter 3	The name of the input item that is to be listed; this can also be a literal.
Parameter 4	The name of the output field into which the item is to be moved.

F.1.7 EMIT Specification

The EMIT specification enables the user to place textual information on the total lines. Up to 50 EMIT specifications can be included. The parameters are as follows:

Parameter 1	Always E to indicate EMIT.
Parameter 2	The break priority number for the total line to which the information is to be moved.

Parameter 3	A literal which is to be placed on the total line. If the information is alphanumeric, it must be enclosed in quotation marks.
Parameter 4	The name of the output field into which the item is to be moved.

F.1.8 INPUT Specification

Each INPUT specification contains a COBOL DATA DIVISION statement to describe the input record. As many statements as are necessary to describe the input record can be placed in the INPUT specifications. If the input file is recorded in ASCII mode, the specification for the input record must include the USAGE is DISPLAY-7 (or DISPLAY-7) clause. The parameters are as follows:

Parameter 1	Always I to indicate INPUT.
Parameter 2	The COBOL statement which must start in column 7, the continuation column.

F.1.9 OUTPUT Specification

Each OUTPUT specification contains a COBOL DATA DIVISION statement to describe the output record. As many statements as are necessary to describe the output record can be placed in the OUTPUT specifications. The 01-level for the output record must not be included. The parameters are as follows:

Parameter 1	Always O to indicate OUTPUT.
Parameter 2	The COBOL statement which must start in column 7, the continuation column.

F.2 OUTPUT FROM COBRG

The output from COBRG consists of one or more COBOL source programs and a listing file. The name of each COBOL source file is that given in the NAME specifications as the program identification. COBRG adds an extension of .CBL to each of these filenames. The source file is always placed on DSK. The listing file, which is also placed on DSK, contains a list of input specifications for all the programs that were generated, and any errors that were detected by COBRG.

F.3 COBRG COMMAND STRING

To operate COBRG, the user need only run the program and, in response to the asterisk, enter a command string of the following form:

```
listfil.ext = infil.ext
```

where: listfil.ext is the name and extension of the listing file, and
infil.ext is the name and extension of the input file containing the specifications for one or more programs.

The default assumptions for the command string are as follows:

a. If the name of the listing file is omitted, the name of the input file is assumed.

b. If the extension of the listing file is omitted, .LST is assumed.

The name of the input file must be specified. The output COBOL source programs that are produced by COBRG are not specified in the command string. They are placed on DSK with user-assigned names and extensions of .CBL.

For example, to produce COBOL programs and a listing file from an input file named TEST, the user could enter the command string:

```
.R COBRG↵
*TEXT.LST=TEST ↵
```

He can achieve the same results with the following command string:

```
.R COBRG ↵
*=TEST ↵
```

F.4 THE REPORT-WRITING PROGRAM

The report-writing program (or programs) is the COBOL source program that is produced by COBRG from the specification file. This program is compiled and then executed with the user's data to produce the report. The input to the report-writing program is a data file that is assumed to be sorted. The first sort-key represents the highest-level break; the last sort-key represents the lowest-level break. The input can be read from any device; however, that device must be specified in the NAME specification for COBRG unless it is to be read from DSK. The output from the report-writing program is the user-defined report. The report, while normally output to the line printer, can be placed on any device. Unless the output device is DSK, it must be specified in the NAME specification for COBRG.

F.5 EXAMPLE OF USING COBRG

The following example produces a COBOL program that prints the total number of customers and total sales for each city and state. The input data records have the following format:

```
01  INPUT-RECORD.
  02  CUSTOMER-NAME          PICTURE X(30).
  02  CITY                   PICTURE X(20).
  02  STATE                  PICTURE XX.
  02  TOTAL-SALES            PICTURE S9(10)V99.
```

The specifications for the report-writing program are as follows:

```
N,ROGER,20,CDR,LPT,CARDINDAT,,31DEC70,JOLLY ROGER
B,1,0,0,0,1,2,STATE
B,0,1,0,2,0,20,CITY
H,3,66,2,0,CUSTOMER   CITY     STATE     SALES
H,PAGE
A,0,14,2,TOTAL-SALES,PRINT-SALES
A,1,5,0,1
T,0,PRINT-SALES,01F
T,1,PRINT-NUM,01F
L,A,CUSTOMER-NAME,PRINT-CUSTOMER
L,A,TOTAL-SALES,PRINT-SALES
L,A,CITY,PRINT-CITY
L,A,STATE,PRINT-STATE
E,1,"STATE TOTAL",TITLE-1
E,0,"CITY TOTAL",TITLE-1
E,F,"FINAL TOTAL",TITLE-1
I 01  INPUT-RECORD.
I       02  CUSTOMER-NAME       PIC X(30).
I       02  CITY                PIC X(20).
I       02  STATE               PIC XX.
I       02  TOTAL-SALES         PIC S9(10)V99.
O       02  PRINT-CUSTOMER.
O         03  FILLER            PIC X(19).
O         03  TITLE-1           PIC X(11).
O       02  FILLER              PIC XX.
```

```
0       02 PRINT-CITY          PIC X(20).
0       02 FILLER              PIC XX.
0       02 PRINT-STATE         PIC XX.
0       02 FILLER              PIC XX.
0       02 PRINT-SALES         PIC ZZZ,ZZZ,ZZZ,ZZZ 99-.
0       02 FILLER              PIC XX.
0       02 PRINT-NUM           PIC Z,ZZ9.
```

The above specifications are placed into a disk file named WRITE.REP and the COBRG program is run. The listing and source files are placed on DSK.

```
.R COBRG)
*=WRITE.REP)
```

A COBOL source program called ROGER.CBL and a listing file are produced. The source program is compiled and executed with the input file that has the following data:

```
ABERDEEN WIDGET          HARTFORD        CN          100000
PRUFROCK CEMENT          HARTFORD        CN          512367
ALGAE LTD.               NEW BEDFORD     CN           51243
DIGITAL EQUIPMENT        MAYNARD         MA         1000000
SAWHORSE CORP.           HUDSON          NH             143
```

The printed report produced by ROGER.CBL is shown below.

```
CUSTOMER                 CITY            STATE       SALES        PAGE 1

ABERDEEN WIDGET          HARTFORD        CN          1,000,00
PRUFROCK CEMENT          HARTFORD        CN          5,123,67

        CITY TOTAL                                   6,123,67     2

ALGAE LTD.               NEW BEDFORD     CN            512,43

        CITY TOTAL                                     512,43     1

        STATE TOTAL                                  6,636,10     3

<NEW PAGE>
CUSTOMER                 CITY            STATE       SALES        PAGE 2

DIGITAL EQUIPMENT        MAYNARD         MA         10,000,00
        :
        :
        FINAL TOTAL                                 16,637,53     5
```

F.6 COBRG RESERVED WORDS

The following is a list of COBRG reserved words. These words, in addition to the COBOL reserved words that are listed in Appendix A, cannot be used as data-names in the specification of the input and output files.

ACCUMULATOR-nn
COMPARE
FINISH
HEADER
IN-FILE
INIT-SW
LEVEL-xxxx
LINE-COUNT
L-PRINT
ONLY
OUTPUT-LINE
PAGE-COUNT
PL-EXIT
PL-HEADER
PRINT
READ-IN
RESET-INIT
RESET-LEVEL-xxxx
SAVE-nn
START
TOP-OF-FORM

NOTE: nn is any number and xxxx is any character string.

Appendix G
The Rerun Program

The RERUN program is used in conjunction with a checkpoint file to restart a COBOL program that has been terminated abnormally because a system failure occurred, a device error was detected, or a disk quota was exceeded. Checkpoint files are core-image dump files that are created in one of two ways: normally, the user includes one or more RERUN statements in his program (refer to I/O CONTROL in Chapter 4); however, the user can also create a checkpoint file by typing CTRL-C twice, followed by REENTER, while the program is running.

The COBOL system creates a checkpoint file by writing a core-image dump file of the program onto disk; at the same time, the COBOL system closes and reopens all disk and magnetic tape output files. The dump is not performed, however, if any files are open for input/output (updating) or if an indexed sequential file is open when the dump is requested. Each time the checkpoint file is written, the COBOL system types the message, DUMP COMPLETED, to notify the user.

If the COBOL program is interrupted during execution, the user can restart the program by means of the RERUN program. The RERUN program reads the dump file back into core, restores the files to their state at the time the checkpoint file was written, and then passes control to the COBOL program so that it can continue processing to completion. RERUN assumes that the operating environment at the time the COBOL program was interrupted is the same as the environment at the time the checkpoint file was written. Thus, the files must be associated with the same types of devices and devices must have the same logical names.

G.1 OPERATING RERUN

To restart a COBOL program from the last checkpoint file written before execution stopped, the user runs the RERUN program by typing:

```
.R RERUN
```

The program responds with the message:

```
TYPE CHECKPOINT FILENAME
```

The user must type the name of the checkpoint file in which the core-image dump is stored. The COBOL system uses the filename of the program as the name of the checkpoint file and adds the extension CKP. If the COBOL program does not have a filename because it was not SAVEd, the COBOL system takes the checkpoint filename from the PROGRAM-ID in the program and adds the extension CKP. If the program has been divided into a two-segment file, the high-segment filename must be the same as the low-segment filename.

If a logical device name is encountered in the program by RERUN, the user is requested to assign the logical name to a specific device. RERUN types the message:

```
ASSIGN device name
TYPE CONTINUE WHEN DONE
```

G.2 EXAMPLES OF USING RERUN

In the following example, the user has a COBOL program that was terminated by a system failure. Checkpoints had been inserted in the program by means of RERUN statements. The program has a filename of ACCNT; thus, the checkpoint filename is ACCNT.CKP. Instead of running the program again from the beginning, the user employs the RERUN program to restart his program from the last checkpoint written before the program stopped. He types:

```
.R RERUN
```

and RERUN responds:

```
TYPE CHECKPOINT FILENAME
```

The user types:

```
ACCNT.CKP
```

RERUN loads the checkpoint file into memory, reopens and repositions the magnetic tape and disk files, and passes control to the COBOL program so that it can continue processing to completion.

In the example below, a user running a COBOL program is notified that the system is going down. He does not have any RERUN statements in his program, yet he wishes to create a checkpoint file so that the processing done by his COBOL program up to that point is not wasted. He creates the checkpoint file by typing CTRL-C twice and then typing REENTER. The checkpoint file is written by the COBOL system onto disk with a filename of PROG13 (taken from the PROGRAM-ID) and an extension of CKP. After the system is restored, the user can restart the program by running the RERUN program. The dialogue is as follows:

```
.R RERUN
TYPE CHECKPOINT FILENAME
PROG13.CKP
```

The program PROG13 is loaded into core, its files are reopened, and it continues running to completion.

Appendix H
Indexed Sequential File Maintenance

Indexed sequential files (also called ISAM files) are data files in which records are accessed through a hierarchy of indexes according to a key within each data record. This file type is commonly used for applications in which the programmer wishes to access records without regard to their locations in the file. The records are read, written, rewritten, and deleted according to the key in each record. Some examples of files for which record keys can be specified are:

payroll (key is employee number)
inventory control (key is part number)
production control (key is job or batch number)

H.1 DESCRIPTION OF INDEXED SEQUENTIAL FILES

An indexed sequential file consists of two files: one, the data file, contains the actual data; the other, the index file, contains pointers to record keys within the data file. The location of the record key within each record is specified by the user when he builds an indexed sequential file using the ISAM program (described below). To build an indexed sequential file, the user provides a sequential file and some necessary information to the ISAM program. The program then copies the data from the sequential file and creates an index file to reference the data file.

All reading and writing of the index file is performed by the run-time operating system (LIBOL); the user need not be concerned with this function. When using indexed sequential files, the programmer need only specify which record is to be read, written, rewritten, or deleted. The operating system performs all searching, insertion, deletion, and updating of both the index and data files.

Indexed sequential files must reside on disk or drum storage. Also, because each indexed sequential file is actually two files, two I-O channels are required - one for the data file and one for the index.

H.1.1 Data File

The data file can be recorded in either SIXBIT or ASCII mode; in either mode, the file must be blocked. When building an indexed-sequential file (by means of the ISAM program), the user must input a

sequential file that contains record keys in the same relative location in each record. Each record must have a unique key, and the keys must be arranged in ascending order (numeric or alphabetic). The user can also indicate to the ISAM program that some records in each block are to be left empty, and some empty blocks should be added to the file. The empty records and blocks are to allow for insertion or addition of new records in the file.

When the user is processing the indexed sequential file, insertions and additions are made by the operating system. Records are inserted in a block in ascending order. If there are no empty records in the block, the block is split into two more or less equal blocks, and the record is added to the appropriate block. When records are added to the file, they are placed in the empty blocks that were allocated when the file was built. If the user does not allocate empty records and blocks when he is building the file, the operating system will request additional blocks from the monitor when the file is full. If the monitor cannot allocate additional blocks, an error message is issued.

H.1.2 Index File

The index file is created by the ISAM program from the description of the data file. It contains up to ten levels of indexes, the lowest of which contains pointers to the record keys in the data file. Each successive level of index points to all of the blocks containing the next lower level index. The highest level index is contained in one block and points to the blocks containing the next lower level index. Index levels are provided so that the entire index need not be searched each time that a record key is accessed. When a record key is accessed, the operating system reads the highest level index to find which lower level index contains a pointer to the approximate location of that key. The block of the next lower level index that contains the approximate location of the key is then searched. If this is the lowest level index, it points to the first record of the data block in which the record is stored. The data block is then searched for the appropriate record key, and the record is made available. If this is not the lowest level index, the next lower level is searched until the lowest level is reached. The figure on the next page illustrates the search.

Within the index file, in addition to the index blocks, are two other blocks - the statistics block and the storage allocation table. The statistics block is a header containing all the necessary information about the index file and the data file. Included in these statistics are: the name and extension of the data file, the number of levels in the index, the blocking factor of both files, and a description of the record key. The storage allocation table shows which data blocks are in use and which are free. There are as many blocks of this table as are necessary to contain this information.

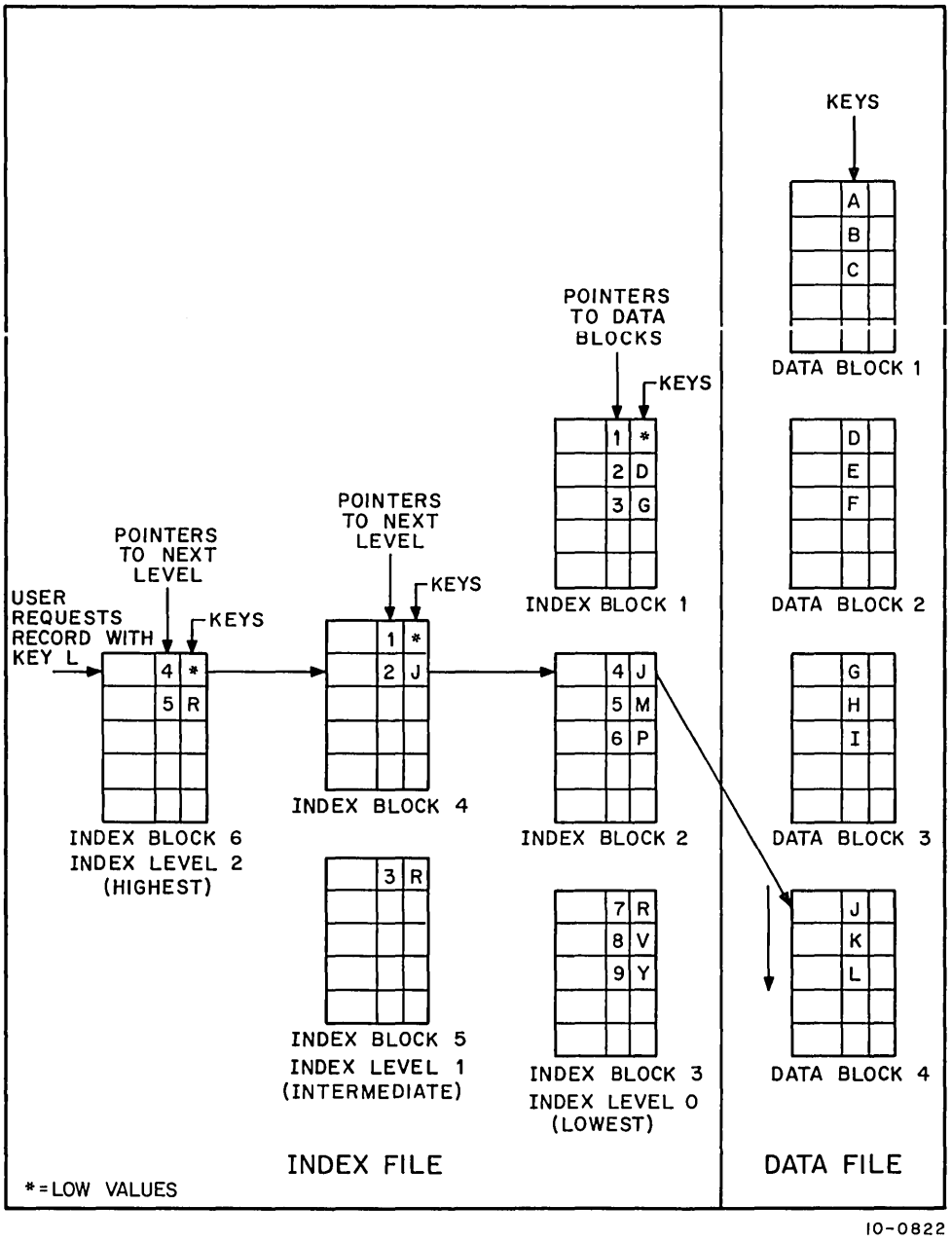


Figure H-1 Locating a Record in an Indexed Sequential File

H.2 PROGRAM TO MAINTAIN INDEXED SEQUENTIAL FILES (ISAM)

Indexed sequential files are created, maintained, and compacted for storage by means of the ISAM program. ISAM performs the following functions for those users who have indexed sequential files.

a. Builds an indexed sequential file from a sequential file.

b. Maintains an indexed sequential file by reorganizing it.

c. Packs an indexed sequential file into a sequential file for backup storage.

H.2.1 Building an Indexed Sequential File

When the user wishes to build an indexed sequential file, he must have as input a sequential file in which the record keys are arranged in ascending order. The ISAM program will change this file to an indexed sequential data file with a user-specified number of empty records and blocks. ISAM then creates the index file according to the description of the data file.

To run the ISAM program and select the option for building the indexed sequential file, the user types the following.

```
.R ISAM
*dev1:indfil.ext[ppn1],dev2:datfil.ext=dev3:seqfil.ext[ppn2]/B
```

dev1, dev2, and dev3 are the devices for the index, data, and input sequential file. Dev1 and dev2 must be disk or drum. The default for dev1, dev2, and dev3 is DSK.

indfil.ext is the name and extension of the index file. If the filename is not specified, the name of the input file is assumed. If the extension is omitted, .IDX is assumed.

datfil.ext is the name and extension of the data file. If the filename is omitted, the name of the index file is assumed. If the extension is omitted, .IDA is assumed.

seqfil.ext is the name and extension of the input sequential file. This filename must be specified, but the extension can be omitted. If it is omitted, .SEQ is assumed.

[ppn1], [ppn2] specify directories for the index file and the input file, respectively. If either is omitted, then the directory of the logged-in user is assumed. The data file must reside in the same directory as the index file.

/B is the switch signifying that ISAM will be used to build an indexed sequential file. If the switch is omitted from the command string, /B is assumed.

The equal sign (=) can be omitted if the specifications for the output files are omitted.

After reading the command string, ISAM asks a series of questions, which are described below. Every question must be answered.

MODE OF INPUT FILE:

The user replies either SIXBIT (or S) or ASCII (or A) according to the mode of the input file.

MODE OF DATA FILE:

The user specifies either SIXBIT (or S) or ASCII (or A) according to the mode in which he desires the data file to be recorded. If the mode of the input file is ASCII and that of the data file is to be SIXBIT, those characters that have no equivalent in SIXBIT are converted in the same manner as they are converted in standard COBOL operations.

MAXIMUM RECORD SIZE:

The user specifies the size of the largest record in the input file in bytes. Note that for ASCII records the user should not count the carriage return and line feed that are appended to each ASCII record.

KEY DESCRIPTOR:

The user describes the key upon which the file is to be indexed in a code that has the form:

sxm.n

s designates the sign of the key:
- S - the key is signed
- U - the key is unsigned

x indicates the key type:
- X - the key is nonumeric
- N - the key is numeric display

m is the number of the byte in the record where the key begins.

n is the size of the key in bytes.

RECORDS PER INPUT BLOCK:

The user gives the blocking factor of the input file. If the file is unblocked, 0 should be specified.

TOTAL RECORDS PER DATA BLOCK:

The user gives the total number of records to be contained in each block of the data file.

EMPTY RECORDS PER DATA BLOCK:

The user specifies the number of records that are to be initially left empty in each block of the data file.

TOTAL ENTRIES PER INDEX BLOCK:

The user specifies the total number of index entries to be contained in each block of the index file.

EMPTY ENTRIES PER INDEX BLOCK:

The user specifies the number of index entries that are to be initially left empty in each index block.

PERCENTAGE OF DATA FILE TO LEAVE EMPTY:

The user gives, as a percentage of the total number of blocks, the number of blocks to be initially left empty in the data file.

PERCENTAGE OF INDEX FILE TO LEAVE EMPTY:

The user gives, as a percentage of the total number of blocks, the number of blocks to be initially left empty in the index file.

MAXIMUM NUMBER OF RECORDS FILE CAN BECOME:

The user replies with the maximum number of records that the data file can possess before the file is next maintained. This number sets the upper limit of the size of the data file. It is required because storage allocation tables must be set up in the index when the file is created. There is no harm in making this number excessively large.

An example of building an indexed sequential file follows.

```
.R ISAM
*TEST.IDX, TEST.IDA=TEST.SEQ /B
MODE OF INPUT FILE: SIXBIT
MODE OF DATA FILE: SIXBIT
MAXIMUM RECORD SIZE: 40
KEY DESCRIPTOR: SN37.4
(The key is signed numeric display; it begins in the
thirty-seventh byte; and it is four bytes long.)
```

(continued on next page)

```
RECORDS PER INPUT BLOCK: 3
TOTAL RECORDS PER DATA BLOCK: 2
EMPTY RECORDS PER DATA BLOCK: 1
TOTAL ENTRIES PER INDEX BLOCK: 3
EMPTY ENTRIES PER INDEX BLOCK: 1
PERCENTAGE OF DATA FILE TO LEAVE EMPTY: 60
PERCENTAGE OF INDEX FILE TO LEAVE EMPTY: 10
MAXIMUM NUMBER OF RECORDS FILE CAN BECOME: 12000
```

H.2.2 Maintaining an Indexed Sequential File

The ISAM program allows the user to recreate an existing indexed sequential file after the file has become crowded. More empty space is added to the file and the number of index levels is decreased. The input is the index portion of the indexed sequential file, and the output is a new indexed sequential data and index file. The command string for the ISAM maintain option is as follows.

```
.R ISAM
*dev1:indfil.ext[ppn1],dev2:datfil.ext=infil.ext[ppn2]/M
```

dev1, dev2, and dev3 are disk or drum devices on which the files are stored. If any of the devices is omitted, DSK is assumed.

indfil.ext is the name and extension of the new index file. If the name is omitted, the name of the input file is assumed. If the extension is omitted, .IDX is assumed.

datfil.ext is the name and extension of the new data file. If the name is omitted, the name of the new index file is assumed. If the extension is omitted, .IDA is assumed.

infil.ext is the name and extension of the index file of the old indexed sequential file. The name of the file must be specified, but the extension can be omitted. No extension is assumed if the extension is omitted.

[ppn1], [ppn2] specify directories for the new index file and the old index file, respectively. If either is omitted, the directory of the logged-in user is assumed. The new data file must reside in the same directory as the new index file.

/M is the switch indicating that the maintain option is being requested. The switch must be specified.

If the output file specifications are not included in the command string, the equal sign (=) can be omitted.

After the command string has been scanned, ISAM asks a series of questions about values for the new indexed sequential file. The mode of the file, the record size, and the key cannot be changed. The values from the old file are given in parentheses with the question. If the user wishes to change a value, he enters the new value; if he does not wish to change a value, he presses the RETURN key. All questions refer to the output file.

TOTAL RECORDS PER DATA BLOCK (n):

The user specifies the total number of records to be contained in each block of the data file.

EMPTY RECORDS PER DATA BLOCK (n):

The user gives the number of data records that are to be initially left empty in each data block.

TOTAL ENTRIES PER INDEX BLOCK (n):

The user gives the total number of index entries to be contained in each block of the index file.

EMPTY ENTRIES PER INDEX BLOCK (n):

The user specifies the number of index entries that are to be initially left empty in each index block.

PERCENTAGE OF DATA FILE TO LEAVE EMPTY (n):

The user gives, as a percentage of the total number of blocks, the number of blocks to be initially left empty in the data file.

PERCENTAGE OF INDEX FILE TO LEAVE EMPTY (n):

The user gives, as a percentage of the total number of blocks, the number of blocks to be initially left empty in the index file.

MAXIMUM NUMBER OF RECORD FILES CAN BECOME (n):

The user specifies the maximum number of records that can be contained in the file. This number sets the upper limit on the size of the data file. It is required because storage allocation tables must be set up when the file is created.

An example of maintaining an indexed sequential file using the ISAM program follows.

```
.R ISAM
*TEST.IDX, TEST.IDA=TEST /M
TOTAL RECORDS PER DATA BLOCK (2): 4
EMPTY RECORDS PER DATA BLOCK (1): 2
TOTAL ENTRIES PER INDEX BLOCK (3):↵
EMPTY ENTRIES PER INDEX BLOCK (1):↵
PERCENTAGE OF DATA FILE TO LEAVE EMPTY (60): 50
PERCENTAGE OF INDEX FILE TO LEAVE EMPTY (10): 40
MAXIMUM NUMBER OF RECORDS FILE CAN BECOME (12000) 25000
```

H.2.3 Packing an Indexed Sequential File

Packing an indexed sequential file is the reverse of building one. An indexed sequential file is copied into a sequential file in the order specified by the index. This option is used primarily to compact an

indexed sequential file for backup storage, although the resulting sequential file can be treated as any other sequential file. The command string for the packing option of ISAM is as follows.

```
.R ISAM
*dev1:seqfil.ext[ppn1]=dev2:indfil.ext[ppn2] /P
```

dev1 and dev2 are the devices on which the sequential file is to be stored and the index file resides, respectively. The input file must be on disk or drum. If neither device is specified, DSK is assumed.

seqfil.ext is the name and extension of the output sequential file. If the name is omitted, the name of the input file is assumed. If the extension is omitted, .SEQ is assumed.

indfil.ext is the name and extension of the index file of the indexed sequential file. The name must be specified, but the extension can be omitted. If the extension is omitted, no extension is assumed.

[ppn1], [ppn2] are directories for the new sequential file and the old index file, respectively. If either is omitted, the directory of the logged-in user is assumed.

/P is the switch signifying that the packing option is being requested. It must be included.

If the output file specification is omitted, the equal sign (=) can be omitted.

After the command string has been processed, ISAM asks the following questions.

MODE OF THE OUTPUT FILE:

The user specifies either SIXBIT (or S) or ASCII (or A) according to the mode in which he wishes the sequential file to be recorded.

RECORDS PER OUTPUT BLOCK:

The user gives the blocking factor that he wishes for the sequential file (i.e., the number of records per logical block). If the file is to be unblocked, the user answers 0.

An example of the dialogue when the packing option is specified follows.

```
.R ISAM
*MTA2:TEST.SEQ=TEST.IDX /P
MODE OF THE OUTPUT FILE: SIXBIT
RECORDS PER OUTPUT BLOCK: 0
SIZE OF LARGEST OUTPUT BLOCK: 40
```

H.2.4 Ignoring Errors

When packing an indexed sequential file into a sequential file, the user can include the /I switch to force ISAM to ignore certain fatal errors. This is used to try to recover as much data as possible from a damaged indexed sequential file.

Including the /I switch in the command string to ISAM causes the program to make nonfatal those errors that concern duplicate keys or keys out of order. The messages for these errors would each be preceded by a percent sign (%) rather than a question mark (?) so that ISAM will continue the packing operation. The /I switch can be used only with the /P switch. It cannot be used alone or with any other ISAM switches unless the user also wishes to include the /L switch (see below) with the /P switch.

The command string when using the /I and /P switches is as follows.

```
.R ISAM
*dev1:seqfil.ext[ppn1]=dev2:indfil.ext[ppn2] /P /I
```

dev1 and dev2 are the devices on which the sequential file is to be stored and the index file resides, respectively. The input file must be on disk or drum. If neither device is specified, DSK is assumed.

seqfil.ext is the name and extension of the output sequential file. If the name is omitted, the name of the input file is assumed. If the extension is omitted, .SEQ is assumed.

indfil.ext is the name and extension of the index file of the indexed sequential file. The name must be specified, but the extension can be omitted. If the extension is omitted, no extension is assumed.

[ppn1], [ppn2] are directories for the new sequential file and the old index file, respectively. If neither is omitted, the directory of the logged-in user is assumed.

/P is the switch signifying that the packing option is being requested. It must be included.

/I is the switch signifying that some fatal errors are to be ignored. It must be included with the /P switch.

The equal sign (=) can be omitted if the output file specification is omitted.

H.2.5 Reading and Writing Magnetic Tape Labels

When building or packing an indexed sequential file, the user can include the /L switch to cause ISAM to read or write labels on magnetic tape. The /L switch, when used with the /B switch, causes ISAM to read standard tape labels on the input magnetic tape. When used with the /P switch, the /L switch causes ISAM to write standard tape labels on the output magnetic tape. The /L switch cannot be used alone or if the input to ISAM during building or the output during packing is not magnetic tape.

The command string when using the /L switch with the /B switch is as follows.

```
.R ISAM
*dev1:indfil.ext[ppn],dev2:datfil.ext=MTAn:seqfil.ext/B/L
```

dev1, dev2, and MTAn are the devices for the index, data, and input sequential file. Dev1 and dev2 must be disk or drum. The default for dev1 and dev2 is DSK.

indfil.ext is the name and extension of the index file. If the filename is not specified, the name of the input file is assumed. If the extension is omitted, .IDX is assumed.

datfil.ext is the name and extension of the data file. If the filename is omitted, the name of the index file is assumed. If the extension is omitted, .IDA is assumed.

seqfil.ext is the name and extension of the input sequential file. This filename must be specified, but the extension can be omitted. If it is omitted, .SEQ is assumed.

[ppn] specifies the directory for the index file. If it is omitted, the directory of the logged-in user is assumed. The data file must reside in the same directory as the index file.

/B is the switch signifying that ISAM will be used to build an indexed sequential file. If the switch is omitted from the command string, /B is assumed.

/L is the switch signifying that ISAM will read standard tape labels. It must be included.

The equal sign (=) can be omitted if the file specifications for the output files are omitted.

The command string when using the /L switch with the /P switch is as follows.

```
.R ISAM
*MTAn:seqfil.ext=devl:indfil.ext [ppn] /P/L
```

MTAn: and devl are the devices on which the sequential file is to be stored and the index file resides, respectively. The input file must be on disk or drum. If devl is not specified, DSK is assumed.

seqfil.ext is the name and extension of the output sequential file. The name and extension can both be omitted because filenames are not used on magnetic tape.

indfil.ext is the name and extension of the index file of the indexed sequential file. The name must be specified, but the extension can be omitted. If the extension is omitted, no extension is assumed.

[ppn] is a directory for the old index file. If it is omitted, the directory of the logged-in user is assumed.

/P is the switch signifying that the packing option is being requested. It must be included.

/L is the switch signifying that ISAM will write standard tape labels. It must be included.

H.2.6 Indirect Commands

The ISAM program accepts command strings and dialogue responses from indirect command files. The command string to direct ISAM to read an indirect command file is:

```
.R ISAM
*@dev:cmdfil.ext[ppn]
```

@ indicates that this is an indirect command file.

dev is the device on which the command file is stored. If it is omitted, DSK is assumed.

cmdfil.ext is the name and extension of the command file. The name and extension must be specified.

[ppn] is the directory in which the command file is stored. If it is omitted, the directory of the logged-in user is assumed.

After ISAM reads the command string, it reads the command file and performs the processing specified within it. The command file must contain the complete command string and all dialogue responses for a single ISAM operation exactly as they would be typed if the user were giving his commands directly. Nothing else can be present in the command file.

Appendix I
Debugging COBOL Programs

COBDDT is an interactive program that is used to debug COBOL programs. With COBDDT, the user can:

a. Change the contents of a data-name,
b. Set up to 20 breakpoints in a program,
c. Continue from a breakpoint or any other point,
d. Display the contents of a data-name, and
e. Trace paragraphs and sections.

I.1 LOADING AND STARTING COBDDT

COBDDT is run after it is loaded and started with a compiled program. When the user program is compiled, the /P switch should not be included in the command string to the compiler because the /P switch suppresses the user symbols that are necessary for COBDDT. The program and COBDDT can be loaded by means of either the monitor LOAD command or direct commands to the Loader. In either case, the Loader must load user symbols along with the program and COBDDT. The /S switch in the Loader command string causes the necessary user symbols to be loaded. After loading, the user issues a monitor command to start the program. However, when COBDDT is loaded with the user program, COBDDT is started, not the program. The three methods of loading and starting are shown below.

```
.LOAD % "LOCALS" dev:prognm, SYS: COBDDT
.START

.DEBUG prgnm
```

```
.R LINK
*/LOCALS dev:prognm, SYS:COBDDT /GO
.START
```

When the program is started, COBDDT is entered. This is shown by the message

```
STARTING COBDDT
*
```

The user can issue any COBDDT command (described below) at this time. If the user desires to run his program, he can issue the PROCEED command to COBDDT. His program will run either to completion or until a fatal error is encountered. If an error is encountered that would normally cause abortion of execution, COBDDT is entered with the message

```
?ENTERING COBDDT FROM:  <paragraph-name>
```

giving the name of the paragraph in which the error occurred. COBDDT can then be used to locate the error. The program cannot proceed after COBDDT has been entered due to an error.

I.2 COBDDT COMMANDS

Described below are the commands to COBDDT. Only the first letter of each command need be typed for COBDDT to recognize the command. Data-names, paragraph-names, and section-names need not be typed in full as long as each name or portion of the name is unique in the program. Paragraph-names may be qualified by section-names and/or the program-name, and data-names may be qualified by higher level data-names and/or by the program-name and/or subscripted. The subscripts for a qualified data-name must appear immediately after the first data-name.

I.2.1 The ACCEPT Command

The ACCEPT command allows the user to change the contents of a data item. The new contents of the data item are typed by the user on the next line. The ACCEPT command has the forms:

```
ACCEPT
ACCEPT data-name
```

If the data-name is not specified, the last name specified in a DISPLAY or another ACCEPT command is assumed.

Example:

```
ACCEPT ALPHA
16.25
```

I.2.2 The BREAK Command

The BREAK command sets a breakpoint (or pause) at the beginning of the specified paragraph. The form of the BREAK command is:

```
BREAK paragraph-name
```

Not more than 20 breakpoints can be set in a program. Breakpoints cannot be set at procedures in overlay sections, nor in the high segment of a reentrant program.

Example:

```
BREAK PAR1 IN COMPUTING
```

I.2.3 The CLEAR Command

The CLEAR command removes the breakpoint at a specified paragraph. The CLEAR command has the forms:

CLEAR paragraph-name
CLEAR

If the paragraph-name is not specified, all breakpoints that have been set in the program are removed.

Example:

```
CLEAR PAR1 IN COMPUTING
```

1.2.4 The DISPLAY Command

The DISPLAY command causes the contents of a data item to be displayed on the user's terminal. The forms of the DISPLAY command are:

DISPLAY
DISPLAY data-name

If no data-name is specified, COBDDT uses the last data-name specified in an ACCEPT or DISPLAY command.

Example:

```
DISPLAY ALPHA
```

1.2.5 The MODULE Command

The MODULE command causes COBDDT to look for data names and procedure names in the specified program. The form of the MODULE command is:

MODULE program-name

Normally, within a run unit containing more than one program, COBDDT searches for data names and procedure names in the current program. The MODULE command changes the program in which the search will take place. All subsequent searches for data names and procedure names will be within the specified program until COBDDT enters the next program in the run unit or until another MODULE command is issued.

Example:

```
MODULE MYPROG
```

1.2.6 The PROCEED Command

The PROCEED command causes the program either to be started or to continue execution after a breakpoint caused it to pause. The PROCEED command has the forms:

```
PROCEED
PROCEED n
```

After a PROCEED command is executed, the program runs either to completion or until another breakpoint is reached. If an integer is included with the command, the program runs until the n^{th} occurrence of the preceding breakpoint has been reached. Thus PROCEED 1 is equivalent to PROCEED.

Example:

```
PROCEED 3
```

1.2.7 The STOP Command

The STOP command is equivalent to the COBOL STOP RUN statement. All files that are open are closed, and program execution is terminated. The STOP command has the form:

```
STOP
```

1.2.8 The TRACE Command

The TRACE command either starts or stops tracing, depending on the form of the command. The forms of the TRACE command are:

```
TRACE ON
TRACE OFF
```

TRACE ON causes tracing of all paragraphs and sections as they are executed. Whenever a paragraph or section is entered, its name, enclosed in angle brackets (< >), is typed on the user's terminal.

TRACE OFF causes COBDDT to stop tracing procedures either until execution is terminated, or another TRACE ON command is executed.

1.2.9 The WHERE Command

The WHERE command causes COBDDT to list the names of all paragraphs at which breakpoints were set. The form of the WHERE command is:

```
WHERE
```

1.3 OBTAINING HISTOGRAMS OF PROGRAM BEHAVIOR

The histogram facility in COBDDT allows the programmer to obtain a report of the number of times each paragraph in his COBOL program was entered as well as the total amount of processor and elapsed time spent in each paragraph. The commands for using this feature are described in the following sections.

1.3.1 Initializing the Histogram Table

The HISTORY INITIALIZE command causes COBDDT to set up and initialize the histogram table in which are stored the statistics for the histogram. The form of this command is:

```
HISTORY INITIALIZE
```

It is not necessary to use this command, but it is advisable to do so if only a portion of the program's statistics are to be recorded. The table can also be reinitialized by means of the HISTORY INITIALIZE command if the user wishes to begin a new histogram.

1.3.2 Starting the Histogram

The HISTORY BEGIN command causes COBDDT to start gathering statistics for each paragraph entered after this command is issued. This command has the form:

```
HISTORY BEGIN
```

This command implies a HISTORY INITIALIZE command if one has not already been issued and if a histogram has not already been started. If a histogram already exists, HISTORY BEGIN will add data to that histogram. The statistics collected include:

The number of times each paragraph is entered.
The CPU time spent within each paragraph.
The elapsed time spent within each paragraph.

1.3.3 Outputting the Histogram

The HISTORY REPORT command causes COBDDT to output the available statistics in a fixed format. This format is:

1	34	46	59
paragraph name	executions	CPU time	elapsed time

The HISTORY REPORT command has the form:

HISTORY REPORT

The report will be output to logical device HISTOR if that device has been assigned by the user or to the user's terminal if HISTOR does not exist. If HISTOR is assigned to a directory device, the report will be given the name REPORT.ØØØ. If more than one report is output from a single execution, the last character of the extension will be incremented, i.e., REPORT.ØØ1, REPORT.ØØ2. Only those paragraphs that have been entered since the HISTORY BEGIN command was issued are included in the report and are output in the same order as they are defined in the program. The report can be sorted into ascending or descending order on any field using the following definitions of field positions and sizes.

Field	SORT Field Description
paragraph name	1.30
no. of executions	34.11
CPU time	46.12
elapsed time	59.12

If the HISTORY REPORT command is not issued before a STOP RUN is executed in the program, the data collected will be lost; hence, it is recommended that the user set a breakpoint at the program's concluding paragraph and issue the HISTORY REPORT command when the breakpoint is reached. A HISTORY BEGIN command must be issued before the report is requested or no statistics will be gathered.

1.3.4 Using the Histogram Feature

To use the histogram feature, the programmer can issue the following commands upon entering COBDDT for the first time:

```
HISTORY INITIALIZE
HISTORY START
BREAKPOINT concluding paragraph
```

Then he can issue the following command upon reaching the concluding paragraph:

```
HISTORY REPORT
```

This series of commands will cause COBDDT to set and initialize the histogram table, start gathering the statistics for the histogram, and output the histogram when the concluding paragraph in the program is reached.

1.4 ERROR MESSAGES FROM COBDDT

Two types of error messages are issued by COBDDT: syntax error messages, which are indicated by an up arrow (↑); and execution error messages, which are indicated by a question mark (?). The syntax error messages deal with errors in the user's command to COBDDT; the up arrow is printed under the last character of the element of the command in which the error was detected. The error messages dealing with errors in the execution of the command are preceded by the question mark.

1.4.1 Syntax Error Messages

↑ ILLEGAL COMMAND

The command issued was not a legal command to COBDDT.

↑ IMPROPER SUBSCRIPT DELIMITER

A subscript must be followed by a comma or a right parenthesis.

↑ IN OR OF MISSING

When a data-name is to be qualified IN or OF must be used.

↑ MATCHES INITIAL SEGMENT OF 2 SYMBOLS

The abbreviation of a name in the command can apply to two items. Data-names and procedure-names must be uniquely identified in a command.

† MISSING DATA-NAME

A data-name is required in the command.

† MISSING QUALIFIER

IN or OF must be followed by a qualifier.

† NOT DEFINED

A name was defined, but not for the purpose it is being used. Another cause could be that insufficient qualification was used.

† ONLY 3 SUBSCRIPTS ALLOWED

More than three subscripts were included for a data-item; COBOL allows only three subscripts.

† SUBSCRIPT ERROR

Negative, zero, or no number was found in the subscript position.

† UNDEFINED DATA-NAME

The data-name used in the command was not defined in the program.

† UNDEFINED PARAGRAPH-NAME

The paragraph-name used in the command is not defined in the program.

† UNDEFINED SECTION-NAME

The section-name used in the command is not defined in the program.

1.4.2 Execution Error Messages

? CANNOT PROCEED

The PROCEED command cannot be executed because COBDDT has been entered due to an abortive error detected in the program.

? ITEM MUST BE SUBSCRIPTED

The data-name has an OCCURS clause and must be subscripted when used.

? ITEM TOO LARGE FOR TEMP

The data item is too large to be displayed or accepted. The maximum number of characters that can be displayed or accepted is approximately 500 characters.

? NO PREVIOUS DATA-NAME

The first ACCEPT or DISPLAY command was used without a data-name.

? NO SUBSCRIPTS ALLOWED

The data-name does not have an OCCURS clause in its description and should not be subscripted.

? NOT ENOUGH SUBSCRIPTS

The data-name requires at least one more subscript.

? OUT OF BREAKPOINTS

All the allowable breakpoints (20) have been set. Some breakpoints should be cleared if the user desires to set more breakpoints.

? SUBSCRIPT TOO LARGE

One of the subscripts typed is outside the maximum range for that data item.

? SYMBOL NOT DEFINED

An attempt was made to place a breakpoint at a non-resident paragraph-name.

? TOO MANY SUBSCRIPTS

The data-name requires at least one fewer subscript.

INDEX

DECsystem-10 COBOL
PROGRAMMER'S REFERENCE MANUAL
DEC-10-LCPRA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754