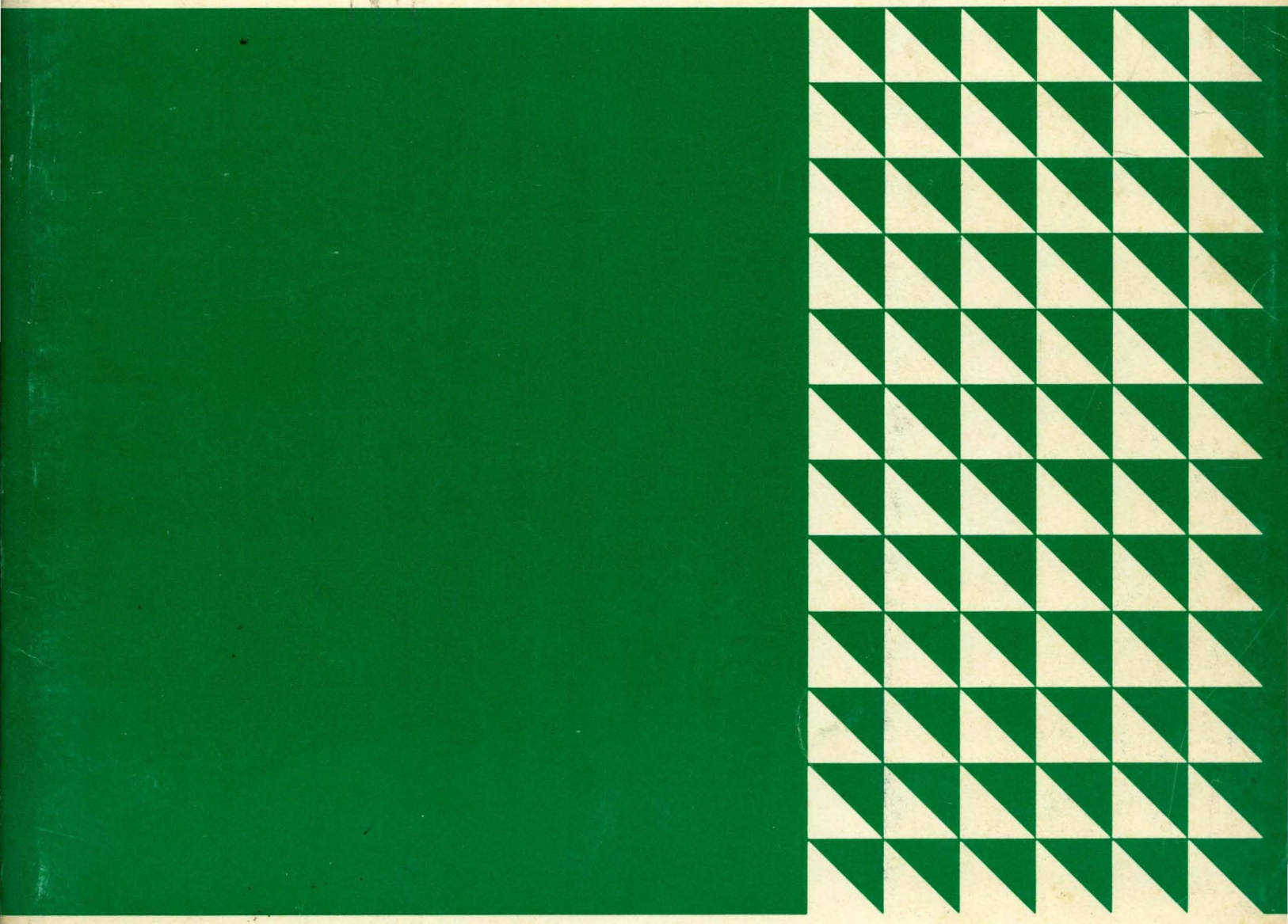
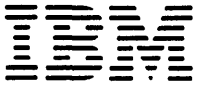
IBM

1130 COBOL
Text – Volume I



Programmed Instruction



1130 COBOL
Text – Volume I

Programmed Instruction

First Edition June 1971

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Address comments concerning the contents of this publication to IBM Corporation, DPD Education Development - Publications Services, Education Center, South Road, Poughkeepsie, New York 12602.

ACKNOWLEDGMENT

The following information is reprinted from COBOL Edition 1965, published by the Conference on Data Systems Languages (CODASYL), and printed by the U.S. Government Printing Office.

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgement of the source, but need not quote this entire section.

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

"No warranty, expressed or implied, is made by the 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 System Languages.

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

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

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

PREFACE

In this programmed course you will learn to code programs using the 1130 subset of the American National Standard COBOL language, concentrating upon basic programming techniques in typical commercial card and disk applications. A description of the type of applications and a list of the specific language features and programming techniques covered are presented in each lesson introduction along with the approximate study time.

Problem descriptions in most of the lessons include a detailed flowchart as a coding guide. A few problems will require that you construct a flowchart before you begin coding the problem solutions. This course assumes that you can already construct and interpret flowcharts. This course also assumes that you have some knowledge of computing system fundamentals and basic concepts of the IBM-1130. If you are unfamiliar with any of these areas you should notify your advisor before beginning this course.

CONTENTS

LIST OF FIGURES

INSTRUCTIONS TO THE STUDENT

This course has been especially designed to provide you with a general orientation to the American National Standard COBOL language as well as to teach you to use specific language features and the PLM and it has been designed to do this as efficiently and quickly as possible. Whether you acquire these skills efficiently and quickly, however, will be determined by the way you go through the course.

This course has been carefully programmed. It is presented in a sequence of steps or frames. You will be asked to respond to each frame. A confirmation will then follow the frame showing a correct response so that you may verify your response.

It is important that you respond carefully to each frame. Students who merely "read" a programmed text often find they are unable to perform a task in a subsequent frame, while those who respond to each frame and then check their responses find they can perform the task quickly and easily. The following frames will illustrate the types of responses you will be asked to make.

1. You may be asked to select a correct answer from several choices given in the frame. Since the correct response is given following the frame, you will need a card (an IBM card would be appropriate) to mask the correct response until you have formulated your own response. Take a card now, and place it just below the next set of three asterisks on this page. Now move the card down the page to the following set of three asterisks.

* * *

This is the confirmation portion of the frame where the correct response is found.

---

2. Most frames contain new information or present information in a new context. Every frame will contain an instruction or a question requiring a response. In this frame you are to choose or select from two possibilities to complete a statement, and then move your card down to expose the confirmation and the next frame.

   The language that is taught in this course is:

   a. FORTRAN

   b. American National Standard COBOL

* * *

b

---

3. The confirmation in the preceeding frame was b. You should have marked b as your response. If both choices were correct the confirmation might be "Both" or "Either"; if neither a nor b were correct, the confirmation would be "Neither."

   Try the same question again, and remember to move your card after you have formulated your response.

   The language that is taught in this course is:

   a. PL/I

   b. ALGOL

* * *

Neither

---

4. Many times you will be asked to select from more than two choices. If you were given four choices, a, b, c, and d, and you selected all of them as correct, the confirmation might be:

   a. Both

   b. All of these

   c. Any of these

* * *

b,c

(If none of the choices is correct, the confirmation will be "None of these.")

---

5. The confirmations "Either" and "Both" mean that choice a and choice b should have been selected. "All of these" and "Any of these" mean that every possible choice should have been selected. Suppose choices a, b, c, and d are presented and the confirmation is "Any of these." You should have selected:

   a. any but not all of the choices

   b. four choices

   c. a, b, c, and d

   d. at least one choice

* * *

b,c

---

6. You are not expected to spend eight hours a day studying this course. If possible, include other activities between lessons. Avoid the other extreme of too much time between lessons. Don't allow time periods of a week to elapse between lessons.

   Match the actions below with the points in a lesson at which the action should be performed.

   1) End of lesson
   2) Middle of frame

   a. Get a cup of coffee
   b. Take a nap
   c. Go out to lunch
   d. Continue with the programmed text

* * *

1) a,b,c
2) d

---

7. In some frames you will be asked to construct a response rather than select one. A lunch break should be taken at the end of a

* * *

lesson

---

8. A blank line in a frame does not indicate the length of response you are asked to construct. A blank line could indicate that you are to write:

   a. one five-letter word.

   b. six or eight words.

* * *

Either

---

9. You will also be asked to write American National Standard COBOL statements as you progress through the course. The confirmation to these frames will be shown as American National Standard COBOL statements on coding forms. In order to respond to a frame in which you are asked to write statements, you would need:

   a. COBOL coding forms.

   b. pencils.

* * *

Both

Before you begin the sequence of frames for Lesson 1, be sure you have the following items on hand.

Language Specifications Manual
COBOL coding forms, Form X28-1464
Pencils
IBM card

Several lessons in this course will require that you code an American National Standard COBOL solution to a problem incorporating statements and rules covered in the lesson and previous lessons. In the confirmation for these solutions, statements taught in the immediate lesson are followed by a number in parentheses indicating the initial frame in the sequence covering that statement. You should go to the appropriate sequence and review the information whenever you do not understand the use of a statement in the confirmation solution.

Some of the statements and features taught in this portion of the course and in the subsequent portion on coding techniques and disk applications are IBM extensions to the American National Standard COBOL. The extensions are indicated by shading in the Language Specifications Manual so that, whenever necessary, you can distinguish between standards and IBM extensions.

Now begin Lesson 1.

LESSON 1

LESSON 1

INTRODUCTION

In 1959 a group of computer professionals, representing the U.S. government, manufacturers, universities and users, formed the Conference On DAta SYstems Language (CODASYL). At the first meeting, the conference agreed upon the development of a common language for the programming of commercial problems. The proposed language would be capable of continuous change and development, it would be problem-oriented and machine-independent, and it would use a syntax closely resembling English, avoiding the use of special symbols as much as possible. The COmmon Business Oriented Language (COBOL) which resulted met most of the requirements.

As its name implies, COBOL is especially efficient in the processing of business problems. Such problems involve relatively little algebraic or logical processing; instead, they usually manipulate large files of basically similar records in a relatively simple way. This means that COBOL emphasizes the description and handling of data items and input/output records.

In the years since 1959, COBOL has undergone considerable refinement and standardization. Now, an extensive subset for a standard COBOL has been approved by ANSI (The American National Standards Institute), an industry-wide association of computer manufacturers and users; this standard is called American National Standard COBOL.

Aside from the COBOL language itself, you should know what happens from the time your program is written until the desired output documents are completed. This sequence is illustrated on the following page.

After the programmer has written the COBOL program on coding sheets, the program is punched into cards. This is the source program. A program called the compiler is loaded into the source computer. Then the source program is loaded into the computer and the compiler translates it into a language the computer can understand (machine language). This version of the original COBOL program is the object program. The object program is then ready to be loaded into the object computer to process the data as specified by the programmer. A single computer may be used as the source computer and then as the object computer.

In this lesson you will look at a COBOL program and from it determine relevant information about the problem and the program.

This lesson will require approximately three-quarters of an hour.

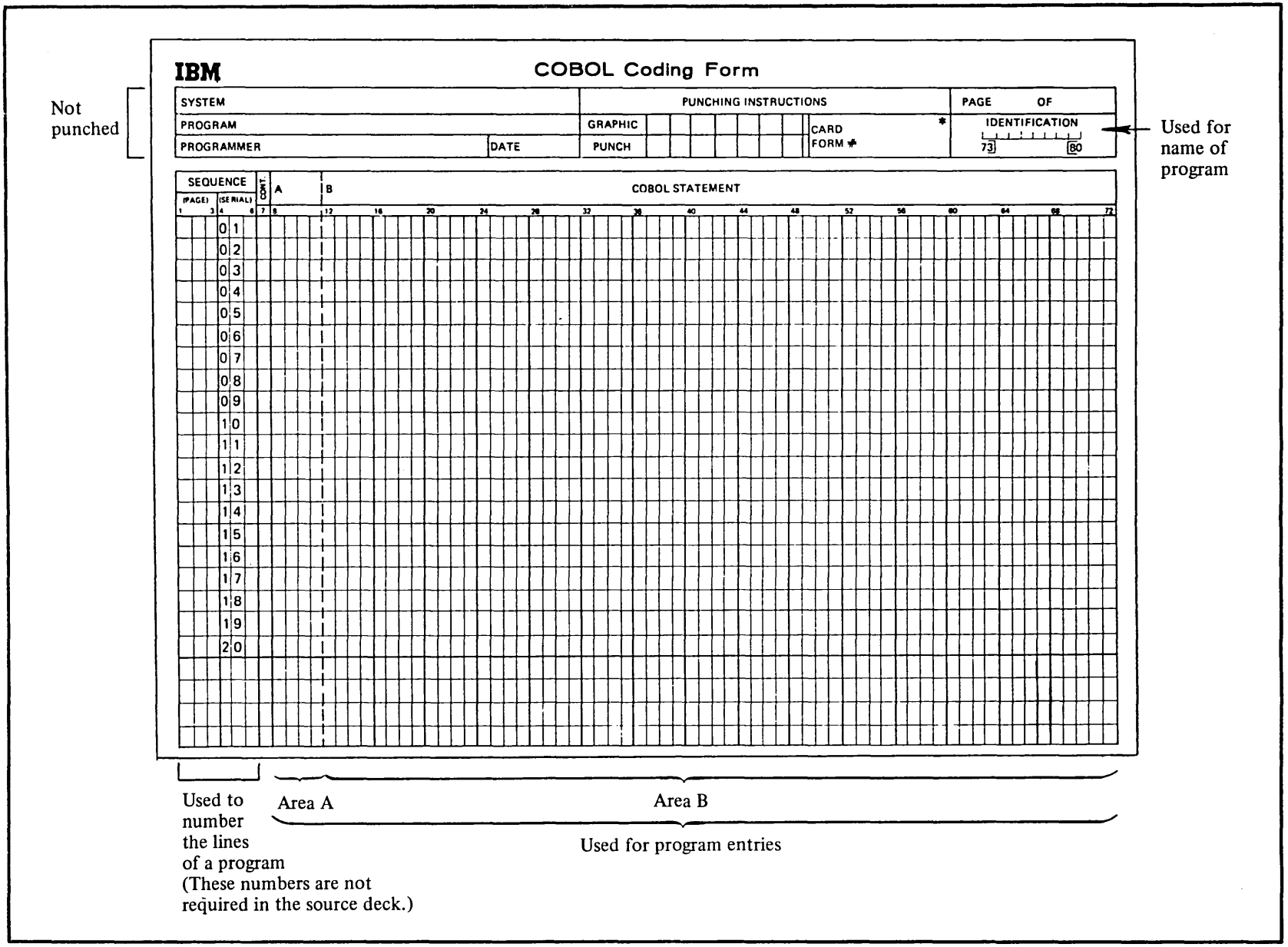

IBM
COBOL Coding Form
SYSTEM
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
CARD FORM #
PAGE OF
IDENTIFICATION
73
80
SEQUENCE
(PAGE)
(SERIAL)
CONT.
A
B
COBOL STATEMENT
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
Not punched
Used for name of program
Used to number the lines of a program (These numbers are not required in the source deck.)
Area A
Area B
Used for program entries

Figure 1

1. Figure 1 shows the special form on which you will write your COBOL programs. Figure 1 shows that a COBOL program is written on a:

   a. COBOL source program.

   b. COBOL Coding Form.

* * *

b

---

2. Columns 1-6 of the coding form are used for sequencing. Figure 1 show that:

   a. you can use these columns to number the lines of your program.

   b. sequence numbers must always be punched in the source deck.

* * *

a

---

3. The text of a COBOL program is made up of statements. Each statement is a program entry. According to Figure 1, the statements you use in your COBOL program will be written in:

   a. columns 1-6.

   b. columns 1-72.

* * *

Neither (columns 8-72)

---

4. There are two areas defined on a COBOL coding form called area A and area B. Figure 1 shows that area A includes columns ..... and area B includes columns ..... .

* * *

8-11
12-72

---

5. Some program entries in a COBOL program must begin in area A and others must begin in area B. An entry beginning in area A should begin in the first column in that area. If an entry must begin in area A, that entry should begin in:

   a. column 8.

   b. column 12.

* * *

a

Figure 2 contains a problem statement and its related COBOL program. Although the problem itself has been simplified for use as an example, the problem statement is probably more detailed than you would generally receive. (Only the portion of the coding form containing our program is shown in the example.)

Problem Statement

A program is to be written to process data on an IBM-1130 computer. The program is also to be compiled on this computer.	A
The I/O devices to be used are a 1442 card reader and 2310 disk drives.	B
Two input files are to be used. The first file, an old master file on cards, contains customer number, name, address, balance and the maximum balance ever carried for the customer.	C
The second input file is a disk file. It contains only the customer number and the month's total purchase.	D
For each record in the old master file there is a record in the transaction file.	
A new master file is to be created on disk (the output file). This file is to contain customer number, name, address, present balance and maximum balance.	E
The new present balance is to be computed by adding the total from the transaction file to the balance in the old master file.	F
If this new balance is greater than the old maximum balance, the maximum balance is to be changed accordingly.	G

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

100     IDENTIFICATION DIVISION.
101     PROGRAM-ID. LESSON-1-EXAMPLE.
102     ENVIRONMENT DIVISION.
103     CONFIGURATION SECTION.
104     SOURCE-COMPUTER. IBM-1130.
105     OBJECT-COMPUTER. IBM-1130.                                    ]  A
106     INPUT-OUTPUT SECTION.
107     FILE-CONTROL.
108         SELECT TRANSACTION-FILE ASSIGN TO DF-1-500-X.
109         SELECT MASTER-FILE ASSIGN TO RD-1442.                     ]  B
110         SELECT NEW-MASTER-FILE ASSIGN TO DF-2-600-X.              ]
111     DATA DIVISION.
112     FILE SECTION.
113     FD  MASTER-FILE.                                              ]
114         LABEL RECORDS ARE OMITTED.                                |
115     01  CUSTOMER-RECORD.                                         |
116         02 CUSTOMER-NUMBER PICTURE X(6).                         |
117         02 NAME PICTURE X(20).                                   |  C
118         02 HOME-ADDRESS PICTURE X(30).                           |
119         02 OLD-BALANCE PICTURE 9999V99.                          |
120         02 MAXIMUM-BALANCE PICTURE 9999V99.                      |
121         02 FILLER PICTURE X(12).                                 ]
122     FD  TRANSACTION-FILE.                                         ]
123         BLOCK CONTAINS 10 RECORDS.                                |
124         LABEL RECORDS ARE STANDARD.                               |
201     01  PURCHASE-RECORD.                                         |  D
202         02 CUSTOMER-NUMBER-T PICTURE X(6).                       |
203         02 TOTAL-PURCHASE PICTURE 9999V99.                       |
204         02 FILLER PICTURE X(68).                                 ]
205     FD  NEW-MASTER-FILE                                           ]
206         BLOCK CONTAINS 4 RECORDS                                  |
207         LABEL RECORDS ARE STANDARD.                               |
208     01  NEW-CUSTOMER-RECORD.                                     |
209         02 CUSTOMER-NUMBER PICTURE X(6).                         |  E
210         02 NAME PICTURE X(20).                                   |
211         02 HOME-ADDRESS PICTURE X(30).                           |
212         02 PRESENT-BALANCE PICTURE 9999V99.                      |
213         02 MAXIMUM-BALANCE PICTURE 9999V99.                      ]
214     PROCEDURE DIVISION.
215     BEGIN.
216         OPEN INPUT MASTER-FILE, TRANSACTION-FILE
217             OUTPUT NEW-MASTER-FILE.
218     MAIN-ROUTINE.
219         READ MASTER-FILE AT END GO TO EOJ.
220     MOVE CUSTOMER-NUMBER OF CUSTOMER-RECORD TO
2201        CUSTOMER-NUMBER OF NEW-CUSTOMER-RECORD.
2202    MOVE NAME OF CUSTOMER-RECORD TO NAME OF NEW-CUSTOMER-RECORD.
2203    MOVE HOME-ADDRESS OF CUSTOMER-RECORD TO
2204        HOME-ADDRESS OF NEW-CUSTOMER-RECORD.
2205    MOVE OLD-BALANCE OF CUSTOMER-RECORD TO OLD-BALANCE
2206        OF NEW-CUSTOMER-RECORD.
2207    MOVE MAXIMUM-BALANCE OF CUSTOMER-RECORD
2208        TO MAXIMUM-BALANCE OF NEW-CUSTOMER RECORD.
221     ADD TOTAL-PURCHASE TO OLD-BALANCE
            GIVING PRESENT-BALANCE.                                   ]  F
222         IF PRESENT-BALANCE GREATER THAN MAXIMUM-
223         BALANCE OF NEW-CUSTOMER-RECORD COMPUTE MAXIMUM-           ]  G
224         BALANCE OF NEW-CUSTOMER-RECORD=PRESENT-BALANCE.           ]
301         WRITE NEW-MASTER-RECORD.
302         GO TO MAIN-ROUTINE.
303     EOJ.
304         CLOSE MASTER-FILE, TRANSACTION FILE, NEW-MASTER-FILE.
305         STOP RUN.
```

Figure 2

---

6. Read the problem statement in Figure 2. This will give you an opportunity to see the relationship between a problem statement and the COBOL program written to solve it. A COBOL program is divided into four divisions:

   Identification Division,
   Environment Division,
   Data Division, and
   Procedure Division.

   The divisions of a COBOL program must always be in the order shown in the program in Figure 2. The first division in a COBOL program must be the ..... Division.

* * *

Identification

---

7. The information in the Identification Division is used for documentation. Therefore, the purpose of the Identification Division is to:

   a. give the computer instructions for executing the program.

   b. describe, or identify, a program and distinguish it from other programs.

* * *

b

8. The division of a COBOL program used to identify the program is the ..... Division.

* * *

Identification

---

8. The division of a COBOL program used to identify the program is the ..... Division.

* * *

Identification

---

9. The second entry in the program in Figure 2 is the PROGRAM-ID paragraph. A program must be given a unique name in the PROGRAM-ID paragraph of the Identification Division. The name given to the program in Figure 2 is:

   a. LESSON-1-EXAMPLE

   b. PROGRAM-ID

* * *

a
(Although the program name is the only entry required in the Identification Division, you may also include such things as the date the program was written, the programmer name, and remarks about the program.)

---

10. The program in Figure 2 shows that the ..... Division immediately follows the Identification Division.

* * *

Environment

---

11. In the problem statement in Figure 2, the equipment to be used for the program is described in the statement(s) identified by:

   a. bracket A.

   b. bracket B.

* * *

Both

---

12. Brackets A and B in Figure 2 show that the equipment required by a program is identified in the:

   a. Identification Division.

   b. Environment Division.

* * *

b

---

13. You would look in the Environment Division of a COBOL program to identify the:

    a. input/output devices required in processing.

    b. computer(s) on which the program is to be compiled and executed.

* * *

Both

---

14. Can a COBOL program be executed on an 1130 computer of different core size than the computer on which the program was compiled?

    a. Yes.

    b. No.

* * *

a

---

15. Match.

1)	Identification Division	a.	Names the program
2)	Environment Division	b.	Describes equipment
		c.	Identifies input/output devices required and computer to be used
		d.	Identifies the program

* * *

1) a,d
2) b,c

---

16. The portion of the problem statement in Figure 2 identified by:

    a. brackets C and D describes the records contained in the input files to be used.

    b. bracket E describes the records contained in the output file to be created.

* * *

Both

---

17. Figure 2 shows that the records in files that are used in processing are described in the ..... Division.

* * *

Data

---

18. According to Figure 2 each record in the input file called MASTER-FILE contains all the information in the record described under CUSTOMER-RECORD. Variables such as CUSTOMER-NUMBER, NAME and HOME-ADDRESS describe data:

    a. punched into cards.

    b. to be used in the program.

* * *

Both

---

19. If you wanted to know what data is recorded on the records of a particular file, you would look in the ..... Division.

* * *

Data

---

20. To identify input/output devices required by a COBOL program, you would look in the ..... Division.

* * *

Environment

---

21. The Data Division of a COBOL program:

    a. identifies the program.

    b. describes records to be used in the program.

* * *

b

---

22. The last division of a COBOL program is the Procedure Division. This division contains instructions that direct the data-processing activities of the computer. Therefore, the Procedure Division:

    a. contains specific instructions for solving a data-processing problem.

    b. identifies the input/output devices required by the program.

* * *

a

---

23. Bracket F in Figure 2 shows that:

    a. the Procedure Division describes the records to be used in the program.

    b. mathematical calculations can be specified in the Procedure Division.

* * *

b

---

24. Bracket G in Figure 2 shows that the Procedure Division can contain conditional instructions. You can see that the instruction indicated by bracket G will cause:

    a. comparison of PRESENT-BALANCE with MAXIMUM-BALANCE OF NEW-CUSTOMER-RECORD.

    b. the maximum balance to be adjusted only if the present balance is greater.

* * *

Both

---

25. The Procedure Division also contains instructions to direct the input and output operations necessary in a program. For example, a READ instruction causes a record to be read, or accessed, from an input file. The data in the record is then available for processing. You can infer that a WRITE instruction would cause:

    a. a record to be written, or placed, in an output file.

    b. a record to be accessed from an input file.

* * *

a

---

26. The Procedure Division of a COBOL program contains instructions:

a. for solving a data-processing problem.

b. specifying mathematical calculations.

c. to direct input and output operations.

* * *

All of these

---

27. Match.

1)	Identification Division	a.	Describes records to be used in the program
2)	Environment Division	b.	Describes equipment required by the program
3)	Data Division	c.	Contains instructions for solving the data-processing problem
4)	Procedure Division	d.	Identifies the program
		e.	Names the program

* * *

1) d,e
2) b
3) a
4) c

Figure 3 is an example of a COBOL program. The program is a sales analysis. It was written for a department store to find departmental totals for the year and the final total for the entire store. Each employee is to be listed with the total amount he sold.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

101     IDENTIFICATION DIVISION.
102     PROGRAM-ID. SALES-ANALYSIS.
103     ENVIRONMENT DIVISION.
104     CONFIGURATION SECTION.
105     SOURCE-COMPUTER. IBM-1130.
106     OBJECT-COMPUTER. IBM-1130.
107     INPUT-OUTPUT SECTION.
108     FILE-CONTROL.
109         SELECT EMPLOYEE-MASTER ASSIGN TO DF-1-400-X.
110         SELECT PRINTFILE ASSIGN TO PR-1132-C.
111     DATA DIVISION.
112     FILE SECTION.
113     FD  EMPLOYEE-MASTER
114         BLOCK CONTAINS 5 RECORDS
115         LABEL RECORDS ARE STANDARD.
116     01  EMPLOYEE-RECORD.
117         02 DEPARTMENT PICTURE XX.
118         02 EMPLOYEE-NUMBER PICTURE X(5).
119         02 NAME PICTURE X(20).
120         02 SALES PICTURE 9999V99.
121         02 FILLER PICTURE X(20).
122     FD  PRINTFILE
123         LABEL RECORDS ARE OMITTED.
124     01  PRINT-RECORD PICTURE X(120).
201     WORKING-STORAGE SECTION.
202     77  DEPARTMENT-1 PICTURE XX VALUE IS 99.
203     77  DEPARTMENT-TOTAL PICTURE 99999V99 VALUE IS ZEROS.
204     77  FINAL-TOTAL PICTURE 999999V99 VALUE IS ZEROS.
205     01  WORK-RECORD.
206         02 FILLER PICTURE X(15) VALUE IS SPACES.
207         02 EMPLOYEE-NUMBER PICTURE X(5).
208         02 FILLER PICTURE X(15) VALUE IS SPACES.
209         02 NAME PICTURE X(20).
210         02 FILLER PICTURE X(15) VALUE IS SPACES.
211         02 SALES PICTURE 9999.99.
212         02 FILLER PICTURE X(55) VALUE IS SPACES.
213     01  TOTAL-RECORD.
214         02 FILLER PICTURE X(100) VALUE IS SPACES.
215         02 TOTALS PICTURE $$$9999.99.
216         02 FILLER PICTURE X(22) VALUE IS SPACES.
```

Continued on next page.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
217     PROCEDURE DIVISION.
218     BEGIN.
219         OPEN INPUT EMPLOYEE-MASTER OUTPUT PRINTFILE.
220     MAIN-SEQUENCE.
221         READ EMPLOYEE-MASTER AT END GO TO TAKE-TOTAL.
222         IF DEPARTMENT GREATER THAN DEPARTMENT-1 PERFORM TAKE-TOTAL.
223     SET-UP.
224         ADD SALES TO DEPARTMENT-TOTAL.
301     MOVE DEPARTMENT OF EMPLOYEE-RECORD TO DEPARTMENT OF WORK-RECORD.
3011    MOVE EMPLOYEE-NUMBER OF EMPLOYEE-RECORD
            TO DEPARTMENT OF WORK-RECORD.
3012    MOVE NAME OF EMPLOYEE-RECORD TO DEPARTMENT OF WORK-RECORD.
3013    MOVE SALES OF EMPLOYEE-RECORD TO SALES OF WORK-RECORD.
302     MOVE DEPARTMENT TO DEPARTMENT-1.
303         WRITE PRINT-RECORD FROM WORK-RECORD
304             BEFORE ADVANCING 2 LINES.
305         GO TO MAIN-SEQUENCE.
306     TAKE-TOTAL.
307         ADD DEPARTMENT-TOTAL TO FINAL-TOTAL.
308         MOVE DEPARTMENT-TOTAL TO TOTALS.
309         MOVE ZEROS TO DEPARTMENT-TOTAL.
310         WRITE PRINT-RECORD FROM TOTAL-RECORD
311             BEFORE ADVANCING 2 LINES.
312     FINISH.
313         MOVE FINAL-TOTAL TO TOTALS.
314         WRITE PRINT-RECORD FROM TOTAL-RECORD
315             BEFORE ADVANCING 2 LINES.
316         CLOSE EMPLOYEE-MASTER, PRINTFILE.
317         STOP RUN.
```

Figure 3

---

28. To find the name of the program in Figure 3, you would look in the ..... Division.

* * *

Identification

---

29. The name of the program in Figure 3 is ..... .

* * *

SALES-ANALYSIS

---

30. To identify the computer(s) on which the program is to be compiled and executed, you would look in the ..... Division.

* * *

Environment

---

31. The names following SOURCE-COMPUTER and OBJECT-COMPUTER identify the computers on which the program is to be compiled and executed, respectively. The two groups of characters in the names specify an IBM 1130 computer.

---

32. The input/output devices required by the program can be found in the ..... Division.

* * *

Environment

---

33. The Data Division of the COBOL program in Figure 3 describes:

    a. records to be used in the program.

    b. data to be used in the program.

* * *

Both

---

34. Which of the following represents information to be used in the program in Figure 3?

    a. NAME

    b. LOCAL-ADDRESS

    c. SALES

* * *

a,c

---

35. The last division in the program in Figure 3 is the Procedure Division. This division:

    a. describes the records to be used in the program.

    b. contains specific instructions for solving the data-processing problem.

* * *

b

---

36. The READ instruction in Figure 3 in the Procedure Division specifies:

    a. an input operation.

    b. a mathematical operation.

* * *

a

---

37. The statement on line 224 in Figure 3 in the Procedure Division specifies:

    a. an input operation.

    b. a conditional instruction.

* * *

Neither
(It specifies the mathematical operation of addition.)

---

38. The statement on line 303 in the Procedure Division specifies:

    a. an output operation.

    b. an input operation.

* * *

a

Read the problem statement in Figure 4. In order to be useful in this lesson, the statement is more detailed than you probably would receive.

An installation is equipped with an IBM 1130 computer.] A

A program for billing is to be written. Three input/output devices will be required: a 1442 card reader, an 1132 printer and a disk drive.] B

There are two input files. The master file, on disk , contains the customer's name, address and number.] C

The transaction file contains a card for each purchase which has the customer number, stock number of parts purchased, description of parts, quantity purchased, unit price, and the total amount of purchase.] D

All purchases for each customer are to be totaled. State tax is to be computed at 4%. Then the total amount due is to be calculated.] E

The output is to be on a preprinted bill in the following form:] F

NAME

ADDRESS

PART #	DESCRIPTION	QUANTITY	UNIT PRICE	TOTAL
7732	14"LANX CABLE	100	.82	82.00
7743	.85"BALL&SOCKET	50	.20	10.00
			SUB-TOTAL	92.00
			TAX	3.68
			TOTAL	95.68

Figure 4

---

39. As a programmer given the job of writing a COBOL program to solve the problem described in Figure 4, you might first decide on a name for the program. You would record this name in the ..... Division of your program.

* * *

Identification

---

40. You would include the information contained in bracket A of Figure 4 in the ..... Division of your program.

* * *

Environment

---

41. The information in bracket B of Figure 4 should be included in the ..... Division of your program.

* * *

Environment

---

42. In the Data Division of your program you should include a description of the:

    a. records described in brackets C and D of Figure 4.

    b. output records, as illustrated in bracket F of Figure 4.

* * *

Both

---

43. The calculations indicated by bracket E in Figure 4 should be specified in the ..... Division.

* * *

Procedure

---

44. You should include the instructions to perform the input and output operations specified in the problem statement in Figure 4 in the ..... Division.

* * *

Procedure

SUMMARY:

You have now completed Lesson 1 in which you have learned to identify the divisions of a COBOL program according to their function:

Identification Division--identifies the program
Environment Division--identifies the input/output devices required and the computer to be used
Data Division--describes the records to be used in the program
Procedure Division--contains instructions for solving the data-processing problem

You have seen how a COBOL program is written on a standard coding form. You have learned to relate various information in a problem statement to corresponding entries in a COBOL source program.

Procedure Division

The Procedure Division contains the instructions needed to solve a problem. To accomplish this, several types of COBOL statements are used:

1. Input Operations - Beginning the Program.

The first step in building the Procedure Division is to make the records contained in the input files available for processing. Use of the OPEN verb establishes a line of communication with each file-input and output. A check is made to ascertain that each file is available for use, and the first record of the input file is brought into the buffer - a special area of internal storage. Other housekeeping tasks are also performed.

Next, a READ command makes the first record from each file available for processing. AT END and INVALID KEY phrases may be appended to the READ statement. These options will be discussed later.

2. Arithmetic Statements

The basic arithmetic operations are all accomplished by verbs specified to implement addition, subtraction, multiplication and division. The verb COMPUTE allows the programmer to incorporate more than one arithmetic operation in a single statement.

3. Conditional Statements

Some instructions examine data to determine whether a given condition is present, and depending on what is found, to carry out an appropriate course of action. The IF verb permits control of the program to be switched or branched to various locations within the program depending on the evaluation of one or more tested conditions. Possible error conditions pertaining to data being processed or to results emerging from computation of the data may be detected and handled through particular usages of the condition statements.

4. Data Manipulation Statements

Data items may be copied from one storage location to another through the use of the MOVE verb. The data being copied will exist in two places - the original location and the location to which the data is moved.

5. Output Operations

When all arithmetic and data-manipulation statements have been executed, the results will be output onto an external medium - printer listing, punched card, or magnetic disk. The output function is performed by means of the WRITE verb.

6. Procedure Branching Statements

Normally, COBOL instructions are processed sequentially, one at a time. The GO TO verb allows the programmer to deviate or branch away from the sequential processing of instructions. Such deviation permits the program to switch direction depending on a variety of circumstances, such as the nature of data being processed or the type of results derived from arithmetic operations. The PERFORM verb also allows alteration of program direction.

7. Ending the Program

After all data has been read, manipulated and results written onto an output device, the program must be ended. The CLOSE verb closes files that have been in use, and the STOP RUN verb initiates COBOL ending procedures after which the execution of the program is halted.

END OF LESSON 1

LESSON 2

LESSON 2 - BASIC INPUT-OUTPUT STATEMENTS; CODING FORMAT

INTRODUCTION

As a computer programmer you will find it useful to be able to display a message to the computer operator at a particular point during execution of your program. For example, you may want to provide the operator with a special instruction should an error be detected in the data being processed. You may want to include a provision in your program for requesting special information when a portion of the program has been executed and then for accepting a reply from the operator. Such messages and replies may be displayed and accepted through the console typewriter. In this lesson you will learn to include provisions in your program for displaying a message to the operator and accepting his reply.

Specific COBOL language features you will learn to use in this lesson are:

ACCEPT statement with the FROM CONSOLE option
DISPLAY statement with the UPON CONSOLE option
Working-Storage Section of the Data Division
Level number 77
PICTURE clause with picture character X
Division headers
Paragraph name in the Procedure Division

This lesson will require approximately three quarters of an hour.

1. Subsequent frames will often include card column captions such as these:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

ACCEPT NAME FROM CONSOLE.

The statement shown above allows the computer operator to key a name into storage through the console typewriter.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

ACCEPT DEPARTMENT FROM CONSOLE.

You might expect that the statement above allows:

a. a department number to be entered into storage through the console typewriter.

b. a value such as 201 or 516 to be keyed into storage.

* * *

Both

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

ACCEPT NAME FROM CONSOLE.

The statement above allows the operator to:

a. key in a name such as JOHN SMITH.

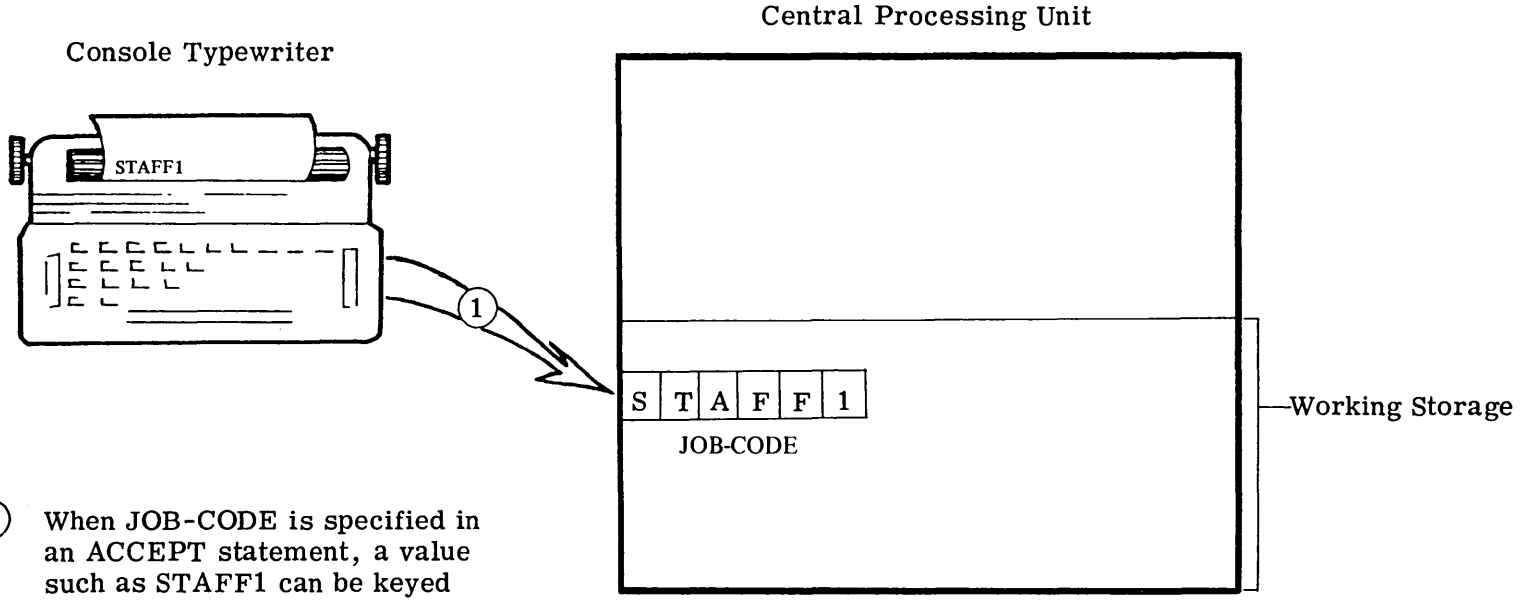
b. key in <u>only</u> the word NAME.

* * *

a

This diagram illustrates execution of the statement

```
8   12   16   20   24   28   32   36   40   44
    ACCEPT JOB-CODE FROM CONSOLE.
```



(1) When JOB-CODE is specified in an ACCEPT statement, a value such as STAFF1 can be keyed into the variable JOB-CODE through the console typewriter.

Figure 5

3. Figure 5 shows how data is keyed into storage using the ACCEPT statement. Read the explanation and look at the illustration in Figure 5. The data name JOB-CODE in Figure 5 is the name of a location in working-storage. This location is reserved to hold a value (in this case an actual job code). The location is called a variable because it may contain various values in the course of the program. According to Figure 5:

   a. STAFF1 is a value that may be keyed into the variable JOB-CODE through the console keyboard if JOB-CODE is specified in an ACCEPT statement.

   b. the value keyed in through the console keyboard will be stored in the working storage location specified in the ACCEPT statement.

* * *

Both
(When an ACCEPT statement with the FROM CONSOLE option is executed, a system generated message code followed by AWAITING REPLY is written on the console typewriter. Execution is then suspended until the same message code followed by a message is keyed in through the console keyboard. Execution is resumed and the message is stored in the working-storage location specified in the ACCEPT statement.)

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           ACCEPT NAME FROM CONSOLE.
```

The statement above would:

   a. allow a value of NAME such as JOHN HANCOCK to be keyed into a location in working storage.

   b. store the word NAME in a location in working storage.

* * *

a

---

5. Using the statement in Frame 4, write a statement that would cause today's date to be entered into a location in working storage through the console typewriter.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           ACCEPT DATE FROM CONSOLE.
```

---

6. Write statements in Area B of your COBOL coding form to allow the computer operator to enter a value of the following variables through the console typewriter.

   1) PAYMENTS-DUE

   2) CUSTOMER-NUMBER

* * *

1)

```
C   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ACCEPT PAYMENTS-DUE FROM CONSOLE.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ACCEPT CUSTOMER-NUMBER FROM CONSOLE.
```

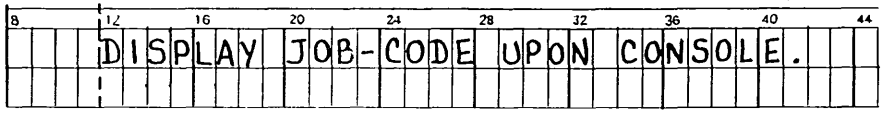
(Statements can begin anywhere in Area B.)

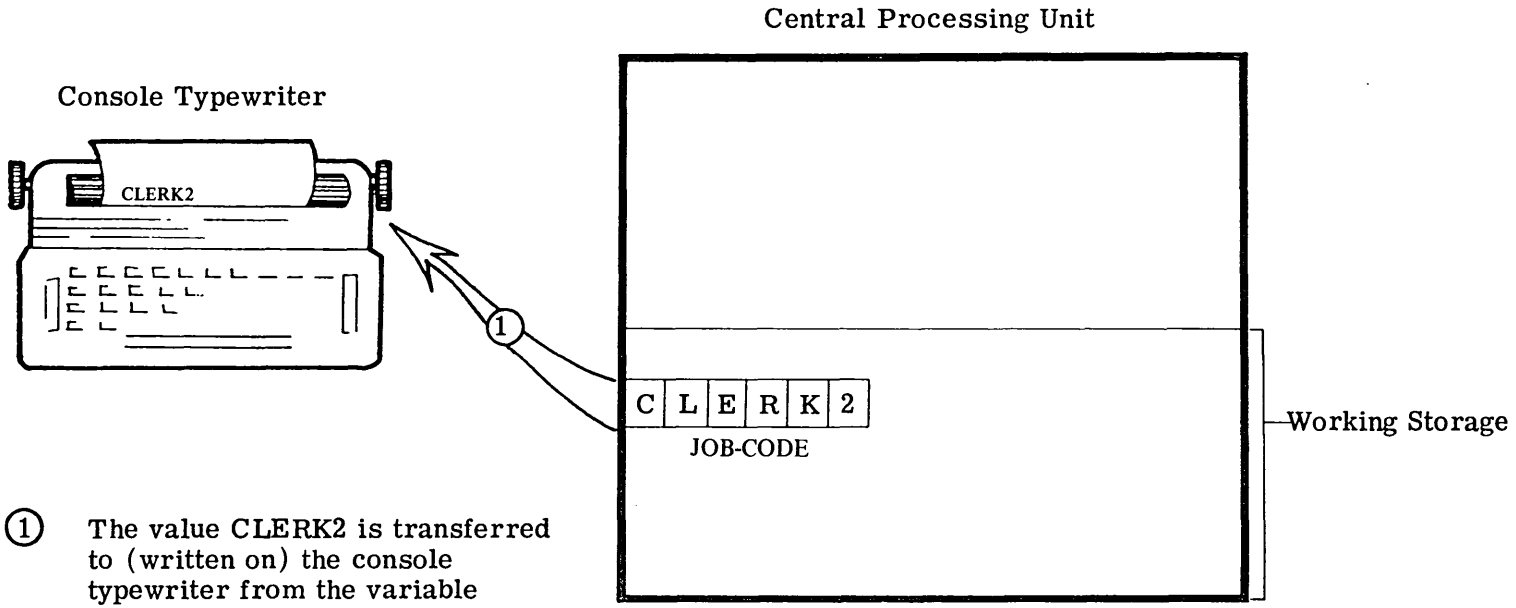
7.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ACCEPT JOB-CODE FROM CONSOLE.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY JOB-CODE UPON CONSOLE.
```

The ACCEPT statement above would allow a value of the variable JOB-CODE to be entered through the console typewriter. The DISPLAY statement would cause data to be written on the console typewriter, as illustrated in Figure 6. Read the explanation and look at the illustration in Figure 6.

This diagram illustrates execution of the statement





① The value CLERK2 is transferred to (written on) the console typewriter from the variable JOB-CODE

Figure 6

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DISPLAY NAME UPON CONSOLE.
```

According to Figure 6 the statement above would be used to:

a. allow a value of NAME to be entered into storage.

b. write a value of NAME on the console typewriter.

* * *

b

8.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DISPLAY PROGRAM-NAME UPON CONSOLE.
```

The statement above would be used to:

a. write the word PROGRAM-NAME on the console typewriter.

b. allow a value of PROGRAM-NAME to be entered into a variable in working storage through the console keyboard.

* * *

Neither (write a value of PROGRAM-NAME on the console typewriter.)

---

9. Which statement below would be used to write a value of a working-storage variable such as 12/21/68, 05/07/68, or 1/12/69 on the console typewriter?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DISPLAY DATE UPON CONSOLE.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DISPLAY 12/21/68 UPON CONSOLE.
```

* * *

a

---

10. Write statements to write a value of the following variables on the console typewriter.

    1) OPERATOR-NAME

    2) EXACT-TIME

                    *   *   *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY OPERATOR-NAME UPON CONSOLE.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY EXACT-TIME UPON CONSOLE.
```

---

11. A Working-Storage Section is not the same as the working-storage on the V2 disk pack. It is instead a data area in core.

    Defining a Working-Storage Section in the Data Division does not refer to working-storage in the disk pack.

    a. True.

    b. False.

                    *   *   *

a

12.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  JOB-CODE PICTURE XXXXXX.
```

Before a variable such as JOB-CODE can be specified in an ACCEPT statement (or a DISPLAY statement), a location in working-storage must be reserved for JOB-CODE. This is done by defining JOB-CODE in the Working-Storage Section of the Data Division as shown above.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  LAST-NAME PICTURE XXXXXXXXXX.
```

The entries above:

a. reserve a location in working storage for values of LAST-NAME.

b. define the variable LAST-NAME so that it can be specified in an ACCEPT statement

* * *

Both
(The number 77 before the data name LAST-NAME indicates that LAST-NAME will have values that are single data items. The level number 77 is always used for single data items, which are also called independent data items.)

---

13. Match each explanation with the correct portion of a COBOL program.

1)	Definition of a variable in the Working-Storage Section of the Data Division	a.	Allows a value to be keyed into a variable defined in working storage through the console typewriter
2)	ACCEPT statement	b.	Reserves a location in working storage
3)	DISPLAY statement	c.	Writes a value of a variable in working storage on the console typewriter

* * *

1) b
2) a
3) c

14.

COBOL Character Set
(in collating sequence, beginning with the highest value)

9 ... 0	(numbers)
Z ... A	(letters)
=	(equal sign)
" or '	(quotation mark, apostrophe, or single quotation mark)
,	(comma)
/	(slash, virgule, stroke)
-	(hyphen or minus symbol)
)	(right parenthesis)
*	(asterisk)
$	(currency symbol)
+	(plus symbol)
(	(left parenthesis)
.	(period or decimal point)
	(blank) (The notation frequently used to indicate a blank is ß.)

Figure 7

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 JOB-CODE PICTURE XXXXXX.
```

PICTURE clause

In the description of JOB-CODE above, the six X-characters in the PICTURE clause specifiy that values of JOB-CODE will be six characters long, although values transferred to JOB-CODE may be less than six characters long. Each X specifies that the character in that position may be any character in Figure 7. A value that could be transferred to JOB-CODE as it is defined in the Data Division entry above is:

a. STENO

b. CLERK3

c. ENGR#2

d. TECHØ4

* * *

a,b,d
(The word PICTURE is a necessary part of the PICTURE clause; c is wrong because # is not acceptable in an X picture.)

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  TIME PICTURE XXXXX.
```

A value that could be transferred to TIME as it is described in the above Data Division entry:

a. could be made up of any characters in Figure 7.

b. could be a maximum of five characters.

c. could be 12:35.

* * *

a,b
(Although it is possible that you will use additional characters from the EBCDIC character set as a programmer on the job, you will be using the COBOL character set in this course.)

---

16. Match the values with the appropriate variables described below:

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

77 PAY-CODE PICTURE XXXX.

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

77 PROBLEM PICTURE XXXXX.

3)

```
0   0    1    1    2    2    3    43   4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

77 PERSONNEL-CODE PICTURE XXX.

a. A/039

b. 620951

c. UPDT1

d. EG4

e. 9208

* * *

1) d,e
2) a,c,d,e
3) d

---

17. Figure 8 shows the correct format for writing Data Division entries. Follow the example in Figure 8 and try writing the Data Division entries to reserve a location in working storage for the variable PROGRAM-NAME. Values of PROGRAM-NAME will be eight characters long. When you have coded the entries, check them with each point in Figure 8 before looking at the confirmation below.

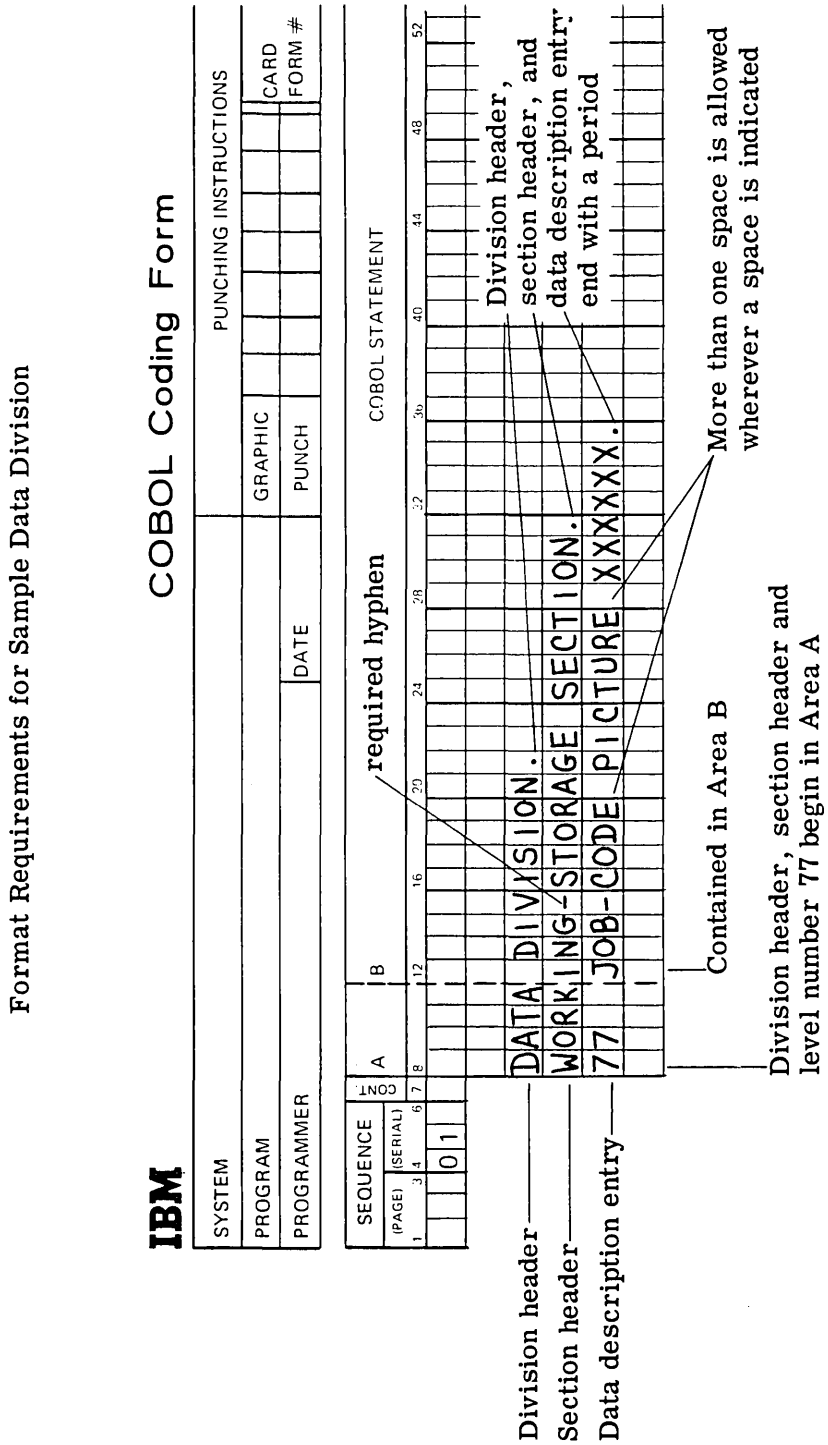


Figure 8

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  PROGRAM-NAME PICTURE XXXXXXXX.
```

---

18. Write the Data Division entries to describe the working-storage variable DEPARTMENT, which will contain values such as X109,A924, and Z125.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  DEPARTMENT PICTURE XXXX.
```

---

19. The note at the lower right in Figure 8 states that wherever a space is indicated:

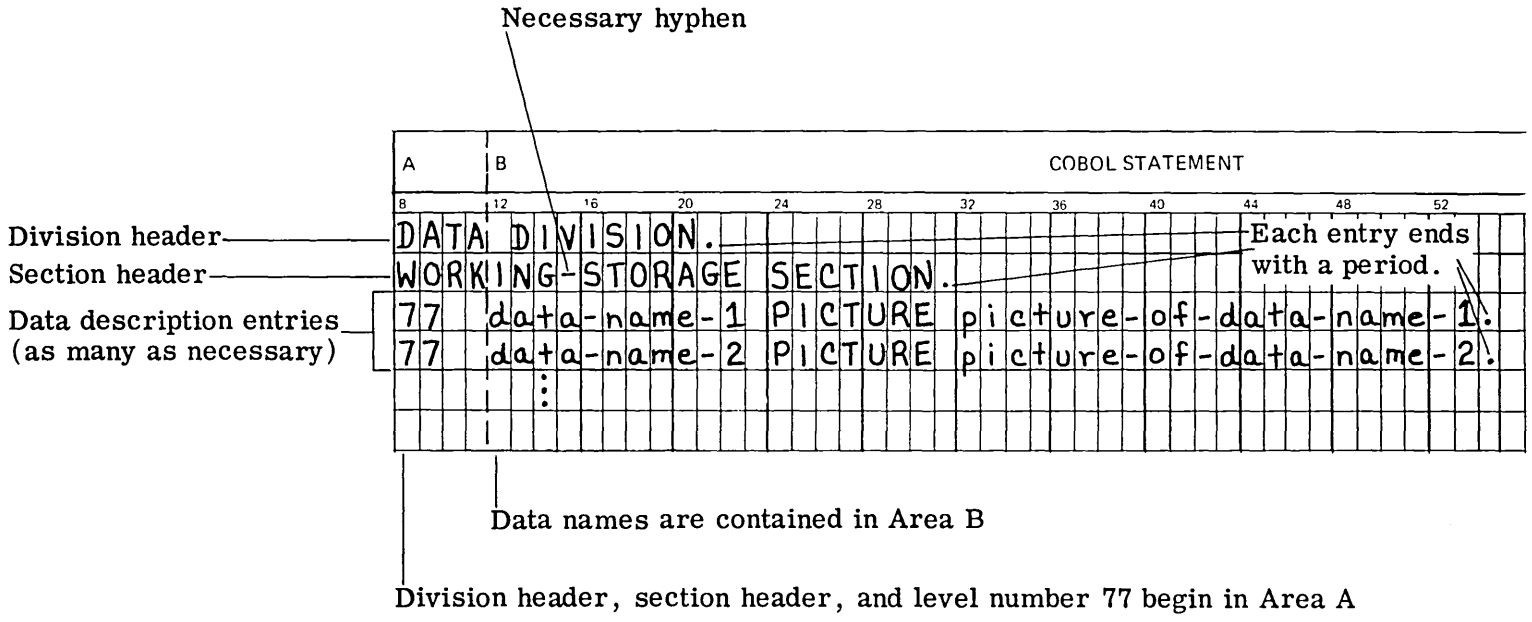
    a. it may be omitted.

    b. more than one space is allowed.

```
                          *       *       *
```

b

Guide for Coding the Data Division with Level 77 Entries in the Working-Storage Section



More than one space is allowed wherever a space is indicated.

Figure 9

20. Figure 9 is a general guide for coding the Data Division with level 77 entries in the Working-Storage Section. Figure 9 shows that the division header and the section header are written:

    a. for each level 77 entry.

    b. only once regardless of the number of level 77 entries.

* * *

b

---

21. Using Figure 9 as a guide, code the Data Division with entries to reserve storage for the working-storage variables LOAN-ACCOUNTS and PROGRAM-NUMBER. LOAN-ACCOUNTS is to have values such as 123, 486, and 019. PROGRAM-NUMBER is to have values such as 67, 91, and 02.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  LOAN-ACCOUNTS PICTURE XXX.
       77  PROGRAM-NUMBER PICTURE XX.
```

IBM AMERICAN NATIONAL STANDARD COBOL RESERVED WORDS

No word in the following list should appear as a programmer defined name.

The words below which are preceded by a • are not reserved words in 1130 COBOL, but should be avoided for compatibility with other American National Standard COBOL compilers.

ACCEPT
ACCESS
ACTUAL
ADD
• ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
• APPLY
ARE
AREA
AREAS
• ASCENDING
ASSIGN
AT
AUTHOR

• BASIS
BEFORE
• BEGINNING
BLANK
BLOCK
BY

CALL
• CANCEL
• CF
• CH
• CHANGED
CHARACTERS
• CLOCK-UNITS
CLOSE
COBOL
• CODE
• COLUMN
• COM-REG
COMMA
COMP
• COMP-1
• COMP-2
• COMP-3
COMP-4
COMPUTATIONAL
• COMPUTATIONAL-1
• COMPUTATIONAL-2
• COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONSOLE
• CONSTANT
CONTAINS
• CONTROL
• CONTROLS
COPY
• CORE-INDEX
• CORR
• CORRESPONDING
CSP
CURRENCY
• CURRENT-DATE
• CYL-INDEX
• CYL-OVERFLOW
C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12

DATA
DATE-COMPILED
• DATE-WRITTEN
• DE
• DEBUG
• DECIMAL-POINT
• DECLARATIVES
• DELETE
DEPENDING
• DESCENDING
• DETAIL
• DISP
DISPLAY
• DISPLAY-ST
• DISPLAY-n
DIVIDE
DIVISION
DOWN

• EJECT
ELSE
END
END-OF-PAGE
• ENDING
ENTER
• ENTRY
ENVIRONMENT
EOP
EQUAL
• EQUALS
ERROR
EVERY
EXAMINE
• EXCEEDS
• EXHIBIT
EXIT
• EXTENDED-SEARCH

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
• FINAL
FIRST
• FOOTING
FOR
FROM

- GENERATE
  GIVING
  GO
- GOBACK
  GREATER
- GROUP

- HEADING
  HIGH-VALUE
  HIGH-VALUES
- HOLD

  I-O
  I-O-CONTROL
- ID
  IDENTIFICATION
  IF
  IN
  INDEX
- INDEX-n
  INDEXED
- INDICATE
- INITIATE
  INPUT
  INPUT-OUTPUT
- INSERT
  INSTALLATION
  INTO
  INVALID
  IS

  JUST
  JUSTIFIED

  KEY
- KEYS

  LABEL
- LABEL-RETURN
- LAST
  LEADING
- LEAVE
  LEFT
  LESS
- LIBRARY
- LIMIT
- LIMITS
- LINAGE
- LINAGE-COUNTER
- LINE
- LINE-COUNTER
  LINES
  LINKAGE
  LOCK
  LOW-VALUE
  LOW-VALUES
- LOWER-BOUND
- LOWER-BOUNDS

- MASTER-INDEX
  MEMORY
  MODE
  MODULES
- MORE-LABELS
  MOVE
  MULTIPLE
  MULTIPLY

- NAMED
  NEGATIVE
  NEXT
  NO
- NOMINAL
  NOT
  NOTE
- NSTD-REELS
- NUMBER
  NUMERIC

  OBJECT-COMPUTER
- OBJECT-PROGRAM
  OCCURS
  OF
  OFF
- OH
  OMITTED
  ON
  OPEN
- OPTIONAL
  OR
- OTHERWISE
  OUTPUT
- OV
- OVERFLOW

- PAGE
- PAGE-COUNTER
  PERFORM
- PF
- PH
  PIC
  PICTURE
- PLUS
- POSITION
- POSITIONING
  POSITIVE
- PREPARED
- PRINT-SWITCH
- PRIORITY
  PROCEDURE
  PROCEED
- PROCESS
  PROCESSING
  PROGRAM
  PROGRAM-ID

  QUOTE
  QUOTES

  RANDOM
- RANGE
- RD
  READ
  READY
  RECORD
- RECORD-OVERFLOW
- RECORDING
  RECORDS
  REDEFINES
- REEL
- RELEASE
- REMAINDER
  REMARKS
- RENAMES
- REORG-CRITERIA
  REPLACING
- REPORT
- REPORTING
- REPORTS
- REREAD
  RERUN
  RESERVE
  RESET
- RETURN
- RETURN-CODE
- REVERSED
- REWIND
- REWRITE
- RF
- RH
  RIGHT
  ROUNDED
  RUN

- SA
- SAME
- SD
- SEARCH
- SECTION
- SECURITY
- SEEK
- SEGMENT-LIMIT
- SELECT
- SELECTED
- SENTENCE
- SEQUENTIAL
- SET
- SIGN
- SIZE
- SKIP1
- SKIP2
- SKIP3
- SORT
- SORT-CORE-SIZE
- SORT-FILE-SIZE
- SORT-MODE-SIZE
- SORT-RETURN
- SOURCE
- SOURCE-COMPUTER
- SPACE
- SPACES
- SPECIAL-NAMES
- STANDARD
- START
- STATUS
- STOP
- SUBTRACT
- SUM
- SUPERVISOR
- SUPPRESS
- SUSPEND
- SW0
- SW1
- SW2
- SW3
- SW4
- SW5
- SW6
- SW7
- SW8
- SW9
- SW10
- SW11
- SW12
- SW13
- SW14
- SW15
- SYNC
- SYNCHRONIZED
- SYSIN
- SYSIPT
- SYSLST
- SYSOUT
- SYSPCH
- SYSPUNCH
- S01
- S02

- TALLY
- TALLYING
- TAPE
- TERMINATE
- THAN
- THEN
- THROUGH
- THRU
- TIME-OF-DAY
- TIMES
- TO
- TOTALED
- TOTALING
- TRACE
- TRACK
- TRACK-AREA
- TRACK-LIMIT
- TRACKS
- TRANSFORM
- TYPE

- UNEQUAL
- UNIT
- UNTIL
- UP
- UPON
- UPPER-BOUND
- UPPER-BOUNDS
- UPSI-0
- UPSI-1
- UPSI-2
- UPSI-3
- UPSI-4
- UPSI-5
- UPSI-6
- UPSI-7
- USAGE
- USE
- USING

- VALUE
- VALUES
- VARYING

- WHEN
- WITH
- WORDS
- WORKING-STORAGE
- WRITE
- WRITE-ONLY
- WRITE-VERIFY

- ZERO
- ZEROES
- ZEROS

Figure 10

The words in uppercase letters in the coding guide Figure 10 are COBOL reserved words. They are reserved for a specific meaning in the COBOL language, and they may not be used as names in a COBOL program. Words not on this list may be used to indicate names or specifications supplied by the programmer using the COBOL language. They are often called user names or user-supplied words. User names such as JOB-CODE and DEPARTMENT are called data names. Data names may be any combination of digits, letters and hyphens with a maximum length of 30. The data name must contain at least one letter, and the initial and final characters must not be hyphens. In addition to fitting these rules, data names must not be COBOL reserved words. A list of COBOL reserved words is given in Figure 10. You may refer to this list whenever you are coding to verify that names you are supplying in your program are not COBOL reserved words.

The Procedure Division contains instructions that direct the data-processing activities of the computer. If a programmer wishes to display a message to the operator during execution of his program, he will code a DISPLAY statement in the Procedure Division. If he wants to allow the operator to reply through the console typewriter, he will code an ACCEPT statement in the Procedure Division. The order in which he codes the statements in this division will depend on the order of the data-processing activities or the logic of his problem solution.

Format Requirements for Sample Procedure Division

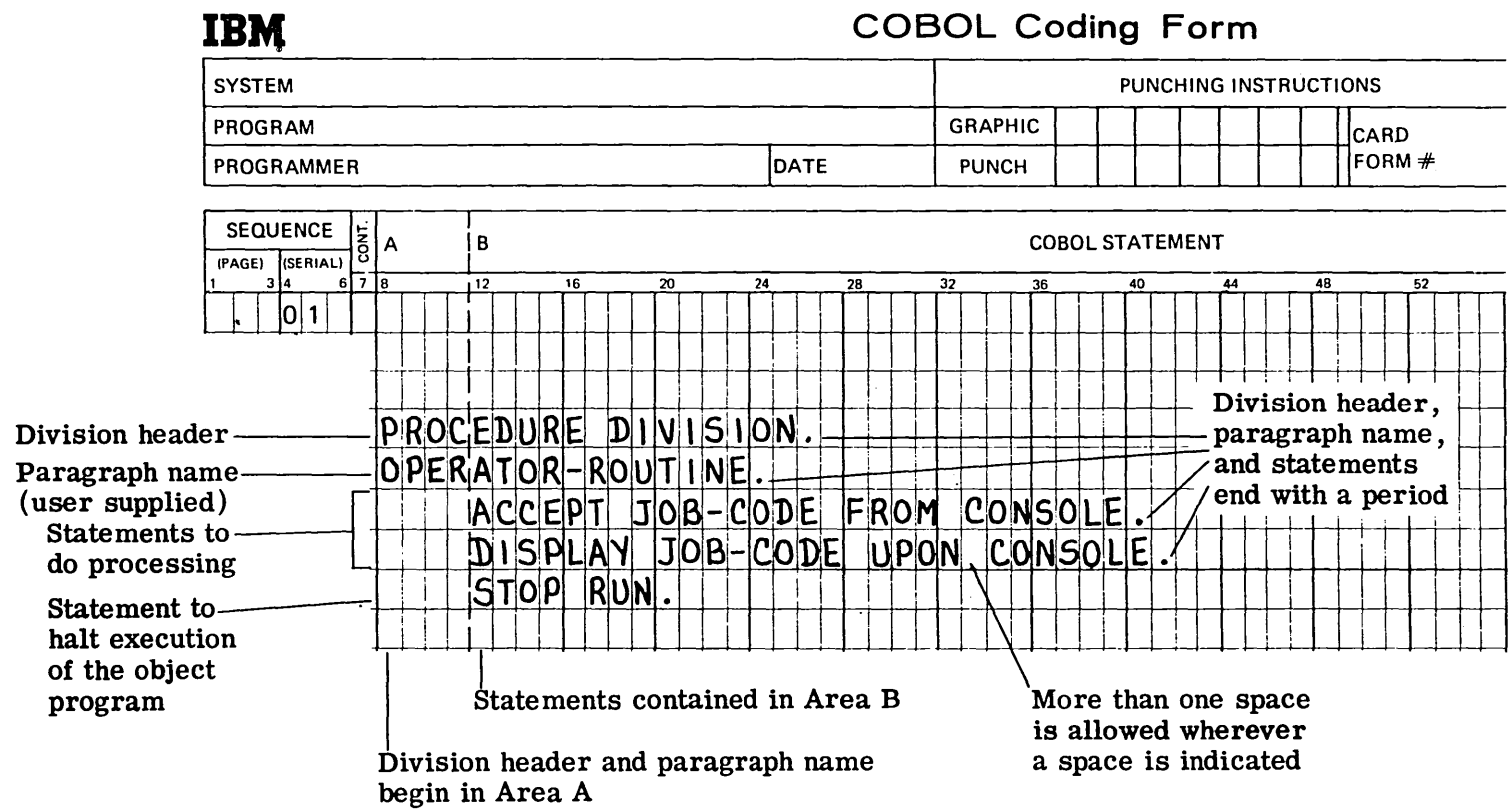


Figure 11

22. Although there is no standard order for statements in the Procedure Division, there are some format requirements. Figure 11 shows a sample Procedure Division containing ACCEPT and DISPLAY statements. The format requirements for this example are indicated in the figure. Follow the example in Figure 11 and determine which of the following is written correctly.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
           ACCEPT JOB-CODE FROM CONSOLE.
           DISPLAY JOB-CODE UPON CONSOLE.
           STOP RUN.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       MAIN-SEQUENCE.
       ACCEPT JOB-CODE FROM CONSOLE.
       DISPLAY JOB-CODE UPON CONSOLE.
       STOP RUN.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       MAIN-SEQUENCE.
           ACCEPT JOB-CODE FROM CONSOLE.
           DISPLAY JOB-CODE UPON CONSOLE.
           STOP RUN.
```

* * *

c
(Example a has no paragraph name. The statements in example b should begin in Area B.)

---

23. In the Procedure Division of a COBOL program, statements that are logically related are grouped into paragraphs. In a program as short as that shown in Figure 11 all the statements in the Procedure Division might be grouped into one paragraph. Every program must have at least one paragraph in the Procedure Division, and each paragraph must be identified by a paragraph name preceding the first statement. In the example in Figure 11:

a. the group of statements in the Procedure Division is identified by the name OPERATOR-ROUTINE.

b. a paragraph name is not required for a program as short as this one.

* * *

a

---

24. A paragraph name is chosen by the programmer. Which is correct?

    a. The paragraph in Figure 11 could have a name other than OPERATOR-ROUTINE.

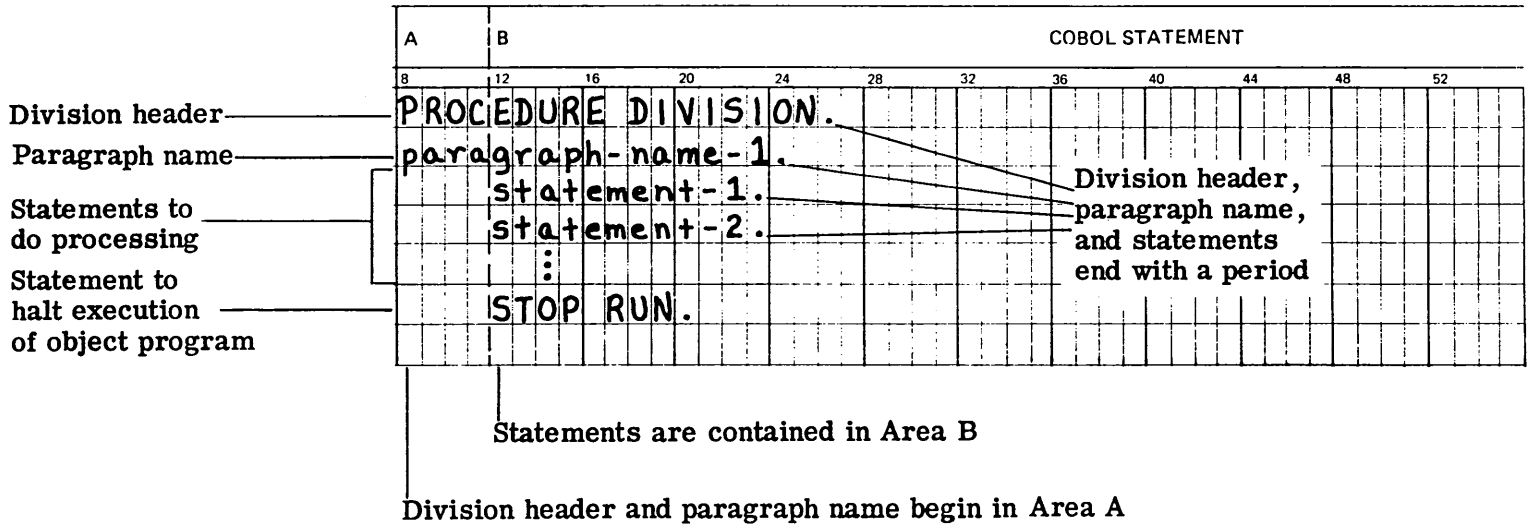
    b. The first paragraph in a program is always named OPERATOR-ROUTINE.

* * *

a
(A paragraph name may be any combination of digits, letters and hyphens with a maximum length of 30. The initial and final characters must not be hyphens. In addition to fitting these rules, paragraph names must not be COBOL reserved words.)

Figure 12

Guide for Coding the Procedure Division



25. Figure 12 is a general guide for coding the Procedure Division. Following the guide, try to write the Procedure Division entries for the paragraph SEQUENCE-1. The statements in this paragraph are to:

    1) allow a value of CODE-DATA to be entered into working storage through the console typewriter.

    2) write a value of CODE-DATA on the console typewriter.

    3) halt execution of the object program.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
       PROCEDURE DIVISION.
       SEQUENCE-1.
           ACCEPT CODE-DATA FROM CONSOLE.
           DISPLAY CODE-DATA UPON CONSOLE.
           STOP RUN.
```

(Although coding in this text will be shown in this format for readability, statements may be coded as shown below. Statements must be separated by one or more blanks and they must be contained in Area B. Any statement may be broken wherever a blank appears and continued on the next line.)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       SEQUENCE-1. ACCEPT CODE-DATA FROM
           CONSOLE. DISPLAY CODE-DATA UPON
           CONSOLE. STOP RUN.
```

26.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY JOB-CODE UPON CONSOLE.
```

The statement above will:

a. write the job code that is stored in working storage on the console typewriter.

b. write a value of the variable JOB-CODE on the console typewriter.

* * *

Both

SUMMARY:

You have now completed Lesson 2 in which you have learned coding for the Data and Procedure Divisions of 1130 COBOL, and the procedure of transferring data to and from the computer through the console typewriter.

END OF LESSON 2

LESSON 3

LESSON 3 - BASIC STANDARD CODING ENTRIES

INTRODUCTION

Lesson 3 will teach you certain standard coding entries for the Data, Identification and Environment Divisions. You will also learn the usage of both numeric and non-numeric literals, which are often needed for printed output. The STOP RUN statement is included since it is the statement used to conclude programmed procedures.

Upon the completion of this lesson, you will have a clearer understanding of what elements are needed to comprise a complete - though simple - program.

This lesson will require approximately three quarters of an hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

DISPLAY 'ENTER DATE' UPON CONSOLE.

In a statement such as the one above, the exact message to be written is specified in the DISPLAY statement. Since ENTER DATE is stored with the DISPLAY statement itself as part of the object coding and is not a variable name, you might expect that:

a. the programmer need not specify an area in working storage for the message.

b. no Data Division entries must be written for the message to be displayed.

* * *

Both

---

2. Match the result with the statement that will cause it.

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

DISPLAY DATE UPON CONSOLE.

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

DISPLAY 'DATE' UPON CONSOLE.

a. The word DATE will be written on the console typewriter.

b. The word 'DATE' will be written on the console typewriter.

c. A value of the variable DATE will be written on the console typewriter.

* * *

1) c
2) a

---

3. Quotation marks in a DISPLAY statement indicate that:

a. the value of the variable is to be written.

b. the message enclosed within quotation marks is to be written.

* * *

b

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY 2468 UPON CONSOLE.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0

           DISPLAY '2468' UPON CONSOLE.
```

Any combination of characters in Figure 7, with the exception of a quotation mark which would be recognized as an end of a message, may be enclosed in quotation marks and specified in a DISPLAY statement. Any combination of digits may be specified in a DISPLAY statement without quotation marks. A combination of digits without quotation marks will be recognized as a numeric literal by the COBOL compiler and will be stored in a certain form as part of the statement itself. A combination of characters in Figure 7 enclosed in quotation marks will be recognized as a nonnumeric literal and will be stored in another form. Either statement above will result in 2468 being written on the console typewriter. The compiler would recognize 256 as a:

a. numeric literal if it were written as 256.

b. nonnumeric literal if it were written as '256'

* * *

Both
(The form in which a literal is stored determines how it may be used by the program. In a subsequent lesson you will learn to specify numeric literals in computations as well as DISPLAY statements.)

---

5. Match each example with the correct term.

1) Numeric literal

2) Nonnumeric literal

a. 'ENTER OPERATOR CODE'

b. 8000

c. '8000'

* * *

1) b
2) a,c

---

6. Write what would be written by each statement.

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY 5090 UPON CONSOLE.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY 'ENTER NUMBER OF ACCOUNTS'
               UPON CONSOLE.
```

* * *

1) 5090
2) ENTER NUMBER OF ACCOUNTS

---

7. In order to write a literal on the console typewriter the programmer must:

   a. reserve a location in working storage for the literal.

   b. always enclose the literal in quotation marks.

* * *

Neither
(Numeric literals do not require quotation marks.)

---

8. ENTER NUMBER OF PAYMENTS DUE.

   Which statement would write the message above on the console typewriter?

   a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY
               'ENTER NUMBER OF PAYMENTS DUE.'
               UPON CONSOLE.
```

   b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY ENTER NUMBER OF PAYMENTS DUE.
               UPON CONSOLE.
```

* * *

a
(Statement b would result in an error message at compile time.)

---

9. Write a statement to write the following message on the console typewriter.

ENTER PROGRAM-NAME.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY 'ENTER PROGRAM NAME.'
               UPON CONSOLE.
```

---

10. Write a statement to write whatever value has been stored in the variable PROGRAM-NAME on the console typewriter.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DISPLAY PROGRAM-NAME UPON CONSOLE.
```

---

11. The programmer must define a location in working storage for:

    a. a literal.

    b. a variable.

    c. a word or words enclosed in quotation marks.

    d. numbers not enclosed in quotation marks.

```
                          *       *       *
```

b

When a message specified as a literal in a DISPLAY statement is too long to fit into a single line on a coding form, the programmer may use a continuation line as explained in the sections titled CONTINUATION OF NONNUMERIC LITERALS and CONTINUATION OF WORDS AND NUMERIC LITERALS in the Language Specifications Manual.

The next sequence of frames will provide you with opportunities to practice coding the Data Division and Procedure Division entries that you have learned up to this point. Whenever you are asked to code, you may refer to the general guides in your Language Specifications Manual.

---

12. Write the Data Division entries for LOAN-ACCOUNTS which will contain values such as 129, 131, 010 and 063.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  LOAN-ACCOUNTS PICTURE XXX.
```

(The data name and its PICTURE clause may be written anywhere in Area B. The level number 77 must be in Area A.)

---

13. The order of the statements in the Procedure Division depends on the logic of the program. If you wish to display a message to the operator and to accept his reply, you should code:

    a. an ACCEPT statement first.

    b. a DISPLAY statement first.

* * *

b

---

14. Now, on the same coding sheet, write the Procedure Division entries for the paragraph SEQUENCE-1 to perform the following steps:

    a. Write the message
       ENTER NUMBER OF LOAN-ACCOUNTS DUE.
       on the console typewriter.

    b. Allow the number of loan accounts that are due to be keyed into LOAN-ACCOUNTS.

    c. Write the value of LOAN-ACCOUNTS on the console typewriter.

    (Remember to write a STOP RUN statement at the end of the program to halt execution.)

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       SEQUENCE-1.
           DISPLAY
           'ENTER NUMBER OF LOAN-ACCOUNTS DUE.'
               UPON CONSOLE.
           ACCEPT LOAN-ACCOUNTS FROM CONSOLE.
           DISPLAY LOAN-ACCOUNTS UPON CONSOLE.
           STOP RUN.
```

[The DISPLAY statement above can be coded entirely on one line or as shown above. Statements can be broken wherever a space occurs (except within a nonnumeric literal) and continued anywhere in Area B. With the exception of continuing a nonnumeric literal, no continuation character is required in column 7.]

---

15. Write all the necessary Procedure Division entries for the paragraph MAIN-SEQUENCE, in which the value of PROGRAM-NAME is to be entered into storage through the console typewriter.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       MAIN-SEQUENCE.
           ACCEPT PROGRAM-NAME FROM CONSOLE.
           STOP RUN.
```

---

16. Figure 13 describes a problem to be solved. As a COBOL programmer, you probably will never be given such a simple problem. However, this problem will give you an opportunity to practice some entries that you have learned and that you might use in a more realistic problem later. Working from the problem statement in Figure 13, write the Data Division and the Procedure Division for the solution on a new coding sheet.

Problem Statement

1) In the paragraph BEGIN, the following message is to be written on the console typewriter:

    KEY IN OPERATION-CODE.

2) A value of OPERATION-CODE is to be entered into storage through the console keyboard. Values of OPERATION-CODE have a form such as:

    106A, 509X, or 287Q.

3) The value of OPERATION-CODE is to be written on the console typewriter.

Figure 13

```
                        *       *       *
0   0   1   1   2   2   3   3   4   4   5   5   6   6   7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 OPERATION-CODE PICTURE XXXX.
       PROCEDURE DIVISION.
       BEGIN.
           DISPLAY 'KEY IN OPERATION-CODE.'
               UPON CONSOLE.
           ACCEPT OPERATION-CODE FROM CONSOLE.
           DISPLAY OPERATION-CODE UPON CONSOLE.
           STOP RUN.
```

In the remaining sequence of frames you will learn to code the Identification Division and the Environment Division, and then you will code a complete COBOL program.

17.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. FILE-UPDATE.
```

Figure 14

The above figure shows the minimum requirements for an Identification Division of a COBOL program. As the division header implies, this division identifies the program. In the example above, the name of the program is:

a. FILE-UPDATE

b. PROGRAM-ID. FILE-UPDATE.

* * *

a
(A program name may be any combination of digits, letters, and hyphens with a maximum length of 30. The initial character must be alphabetic, and the final character must not be a hyphen. In addition to fitting these rules, program names must not be COBOL reserved words. Although a program name may have a maximum length of 30, only the first five characters are used for identification. Therefore, the first five characters of a program name should not be duplicated in any other program name.)

---

18. The IDENTIFICATION DIVISION has 6 entries, only the PROGRAM-ID is mandatory. The others are optional. The following example shows all six of them.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. TESTING.
       AUTHOR. JOHN PROGRAMMER.
       INSTALLATION. MARYLAND USA.
       DATE-WRITTEN. MARCH 29,1970.
       SECURITY. A-14-1669.
       REMARKS. THIS PROGRAM IS A SAMPLE PROGRAM
                FOR IBM 1130 COBOL.
```

Guide for Coding the Identification Division

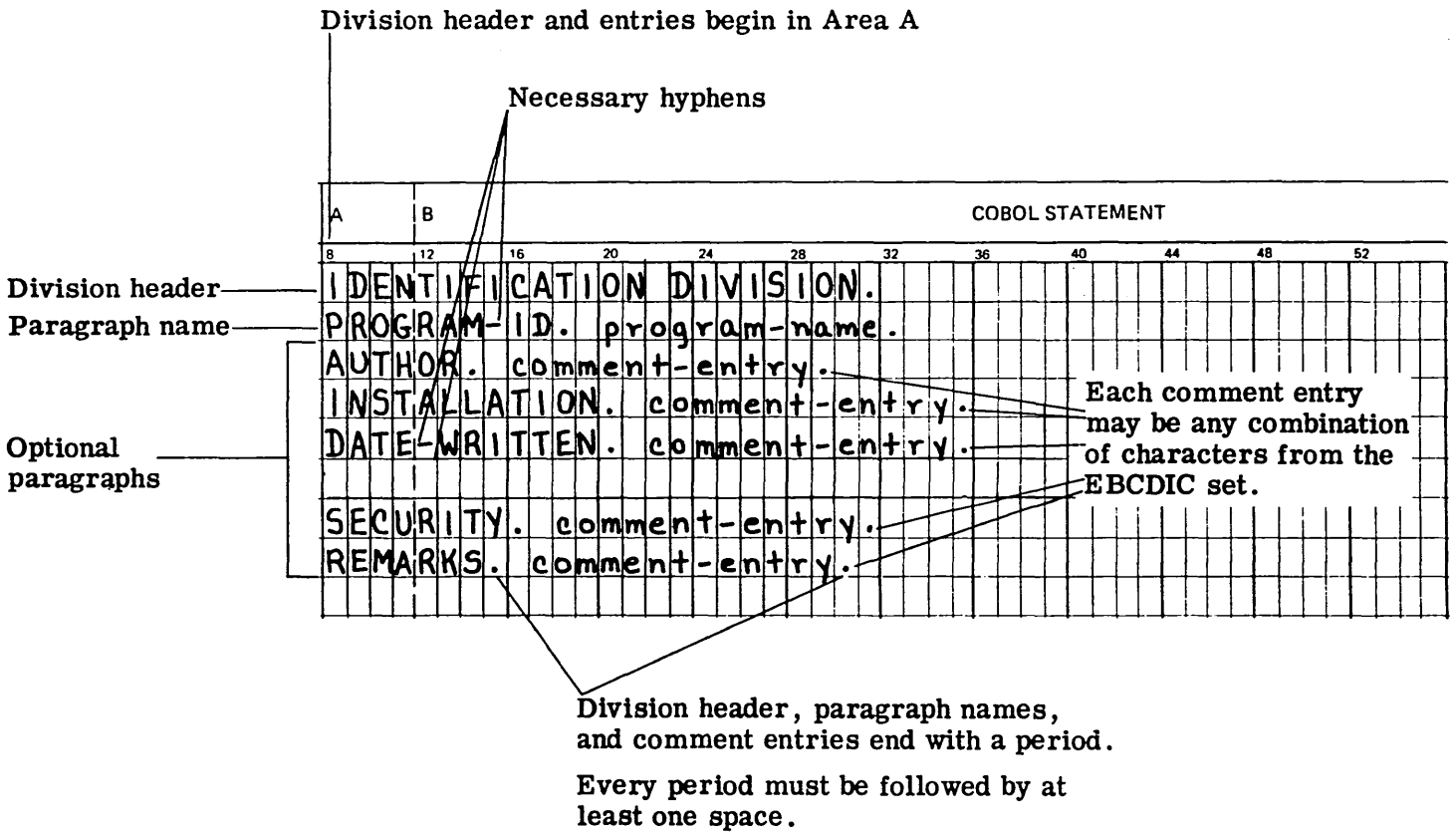


Figure 15

19. Figure 15 is a general guide for coding the Identification Division. Use Figure 15 to determine which of the following examples is written correctly.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. FILE-UPDATE.
       DATE-WRITTEN.  10/16/68.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       AUTHOR. SMITH.
       DATE WRITTEN. 02/31/68.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. SALES-ANALYSIS.
```

* * *

a,c

---

20. Use Figure 15 to determine which of the following examples is written correctly.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID.SALES ANALYSIS.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION
       PROGRAM-ID. CARD-TO-TAPE
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID - PERSONNEL-MASTER.
```

* * *

None of these
(In a the space following a period is missing. In b the period following the division header is missing. In c the hyphen is misplaced and a period is missing.)

---

21. Follow the coding guide in Figure 15 and try to write the Identification Division entries for a program called PAYROLL. Code only the required entries.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. PAYROLL.
```

Format Requirements for Sample Environment Division

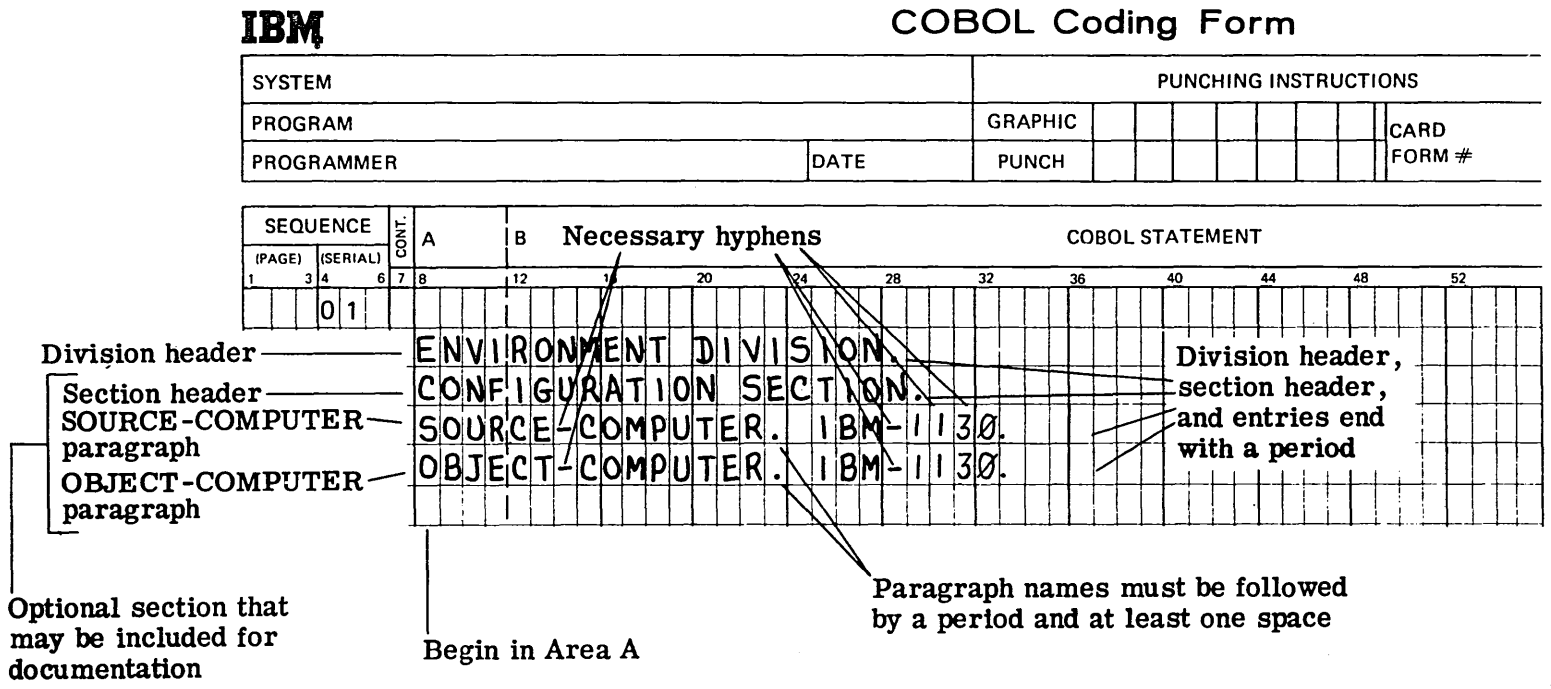


Figure 16

22. Figure 16 shows a sample portion of the Environment Division called the Configuration Section. It specifies the computer configuration used to compile the program as well as the one used to execute the program. For example, the SOURCE-COMPUTER paragraph in Figure 16 indicates that the source program is to be compiled on an IBM 1130 computer. The OBJECT-COMPUTER paragraph in Figure 16 indicates that the object program is also to be executed on an IBM 1130.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
```

In the example above, the Configuration Section indicates that:

a. the source program is to be compiled on an IBM-1130 computer.

b. the object program is to be executed on an IBM-1130 computer.

* * *

Both

---

23. A program called PAYROLL is to be compiled on an IBM-1130 computer. It is to be executed on an IBM-1130 computer. Use Figure 16 to determine which of the following Environment Divisions contains a correct Configuration Section for PAYROLL.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER IBM-1130.
       OBJECT-COMPUTER IBM-1130.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
```

* * *

b (a is wrong because periods are missing in the SOURCE-COMPUTER and OBJECT-COMPUTER clauses.)

---

24. Write the Identification and Environment Divisions for the program COMMISSION-CALCULATION, which will be compiled and executed on an IBM-1130 computer.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. COMMISSION-CALCULATION.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
```

---

25. Write the Identification, Environment, and Data Division for the program DATA-LIST, in which values such as SALES and TAXES will be read into and written from the variable PROGRAM-NAME. DATA-LIST will be compiled and executed on an IBM-1130 computer.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. DATA-LIST.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  PROGRAM-NAME PICTURE XXXXX.
```

---

26. Code the Data and Procedure Divisions to provide for the following:

    1) In paragraph MESSAGE-ROUTINE, the message
       ENTER NUMBER OF PAYMENTS DUE.
       is to be written on the console typewriter.

    2) The number of payments due is to be keyed into the variable PAYMENTS-DUE.

    3) PAYMENTS-DUE will have values such as 106,235, and 084.

    4) The message END-JOB is to be written on the console typewriter.

                        *   *   *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  PAYMENTS-DUE PICTURE XXX.
       PROCEDURE DIVISION.
       MESSAGE-ROUTINE.
           DISPLAY
           'ENTER NUMBER OF PAYMENTS DUE.'
               UPON CONSOLE.
           ACCEPT PAYMENTS-DUE FROM CONSOLE.
           DISPLAY 'END-JOB' UPON CONSOLE.
           STOP RUN.
```

---

27. The solution to the problem described below will include entries that you will code in many of the programs that you will be writing as a programmer on the job. Code all divisions for the program UPDATING to do the following:

    1) In paragraph SEQUENCE-1, write the following message on the console typewriter:
    ENTERØTODAYSØDATE.
    (Since an embedded quotation mark would be recognized as the end of a nonnumeric literal, it must be replaced by a blank.)

    2) Allow a value of DATE to be entered into storage through the console keyboard in a form such as:
    06/21/68, 12/01/68, or 03/09/69.

    3) Write the value of DATE on the console typewriter.

    UPDATING will be compiled and executed on an IBM-1130.

                        *   *   *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.                    (44)
       PROGRAM-ID. UPDATING.
       ENVIRONMENT DIVISION.                       (49)
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       DATA DIVISION.                              (12)
       WORKING-STORAGE SECTION.
       77  DATE PICTURE XXXXXXXX.
       PROCEDURE DIVISION.                         (22)
       SEQUENCE-1.
           DISPLAY 'ENTER TODAY S DATE.'
               UPON CONSOLE.
           ACCEPT DATE FROM CONSOLE.                (1)
           DISPLAY DATE UPON CONSOLE.
           STOP RUN.                               (22)
```

(The number in parentheses at the end of each entry identifies the frame in which the use of that entry was introduced. Your solution may appear different but be correct provided you have followed the rules for placement of entries and for separation and breaking of statements.)

SUMMARY:

You have now completed Lesson 3 in which you have learned coding for three divisions in a COBOL program and more about the simple procedure of transferring data to and from the computer through the console typewriter. Although the ACCEPT and DISPLAY statements provide one way to transfer data to and from the computer, they are usually used for low-volume input and output such as codes or messages.

A larger volume of data would require a more efficient input or output device such as a card reader, printer, or disk drive. You will learn to use COBOL language features to process larger volumes of data in subsequent lessons.

END OF LESSON 3

LESSON 4

LESSON 4 - INTRODUCTION TO DATA FILES

INTRODUCTION

In many data-processing activities related data items are grouped into a record of data. For example, the data items relating to a single customer may be grouped into a record for that customer. The record may be punched into a card, printed in a report, or stored on magnetic tape or disk. Processing a file of such records would involve processing a large volume of data. Although you will not learn to process a complete file in this lesson, you will learn to code the Data Division entries for using record variables.

Specific COBOL language features you will learn to use in this lesson are:

Level numbers 01 and 02.
Repetition factor in the PICTURE clause.
ACCEPT statement

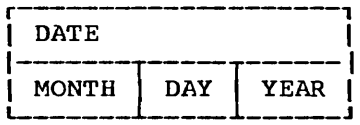
This lesson will require approximately one hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

77 DATE PICTURE XXXXXXXX.

The entry above describes a variable in working storage whose values throughout execution of the program will be single data items. A single data item, a data item that is not subdivided further, is called an elementary item. A variable whose values throughout execution of the program will be elementary items, then, is an elementary variable.



In the diagram above, the variable DATE has been subdivided into the variables MONTH, DAY, and YEAR. The values of MONTH, DAY, and YEAR will be single data items. In this example, an elementary variable is:

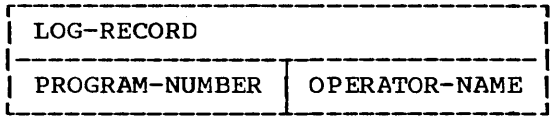
a. DATE.

b. MONTH,DAY, or YEAR.

* * *

b

2.



Several related elementary variables may be grouped together as a group variable. The variables may be referred to individually or as a group. The elementary variables PROGRAM-NUMBER and OPERATOR-NAME in the diagram above have been grouped together as the variable LOG-RECORD. LOG-RECORD is:

a. an elementary variable.

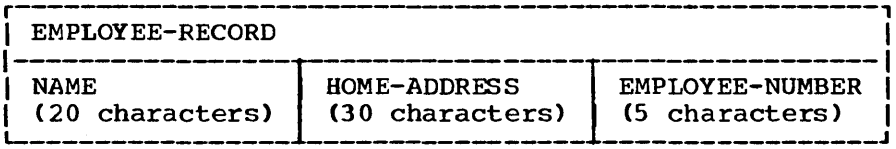
b. a group variable.

* * *

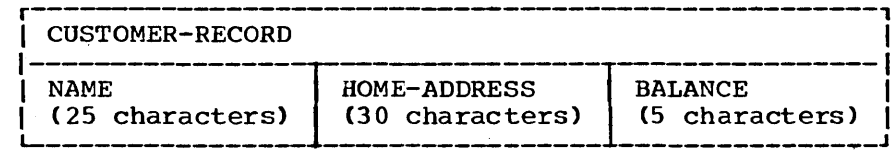
b

3.

A.



B.



A number of related elementary variables may be grouped together into a group variable. A group variable is usually called a record variable. For instance, in diagram A, the variables whose values may refer to one employee are grouped into a record variable for a single employee. Diagram B shows:

a. a grouping of related variables whose values refer to one customer.

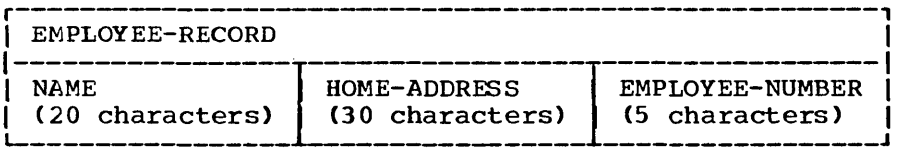
b. a grouping of four elementary items.

c. a record variable whose values refer to one customer.

* * *

a,c

4.



In a program, a reference can be made to the record variable or to any of the elementary variables within the record variable. In the example in the diagram above, a reference could be made to:

a. all the elementary variables with the record variable name EMPLOYEE-RECORD.

b. any of the elementary variables, such as NAME, HOME-ADDRESS, or EMPLOYEE-NUMBER.

* * *

Both

---

5. Throughout execution of a program, a variable in storage:

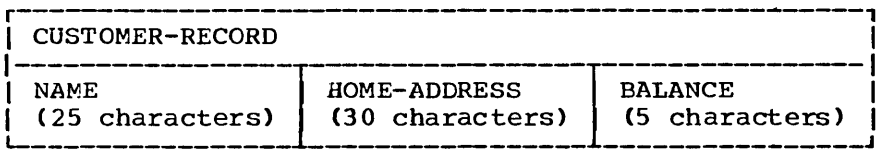
   a. may have various values.

   b. has a constant value.

                    *   *   *

a

6.



The group of data items that are the values of the elementary variables in a record variable constitute a record.

If a value of each of the elementary variables in the record variable in the diagram above were punched into a card, the card would contain:

   a. data for one customer.

   b. a record.

                    *   *   *

Both

---

7. A record variable:

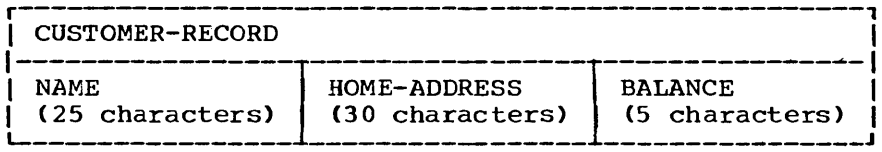
   a. may have various records as values throughout execution of a program.

   b. has a constant record value throughout execution of a program.

                    *   *   *

a

8.



Values of NAME, HOME-ADDRESS, and BALANCE in the record variable illustrated above have been punched into a separate card for each customer.

Throughout execution of a program that processes these cards:

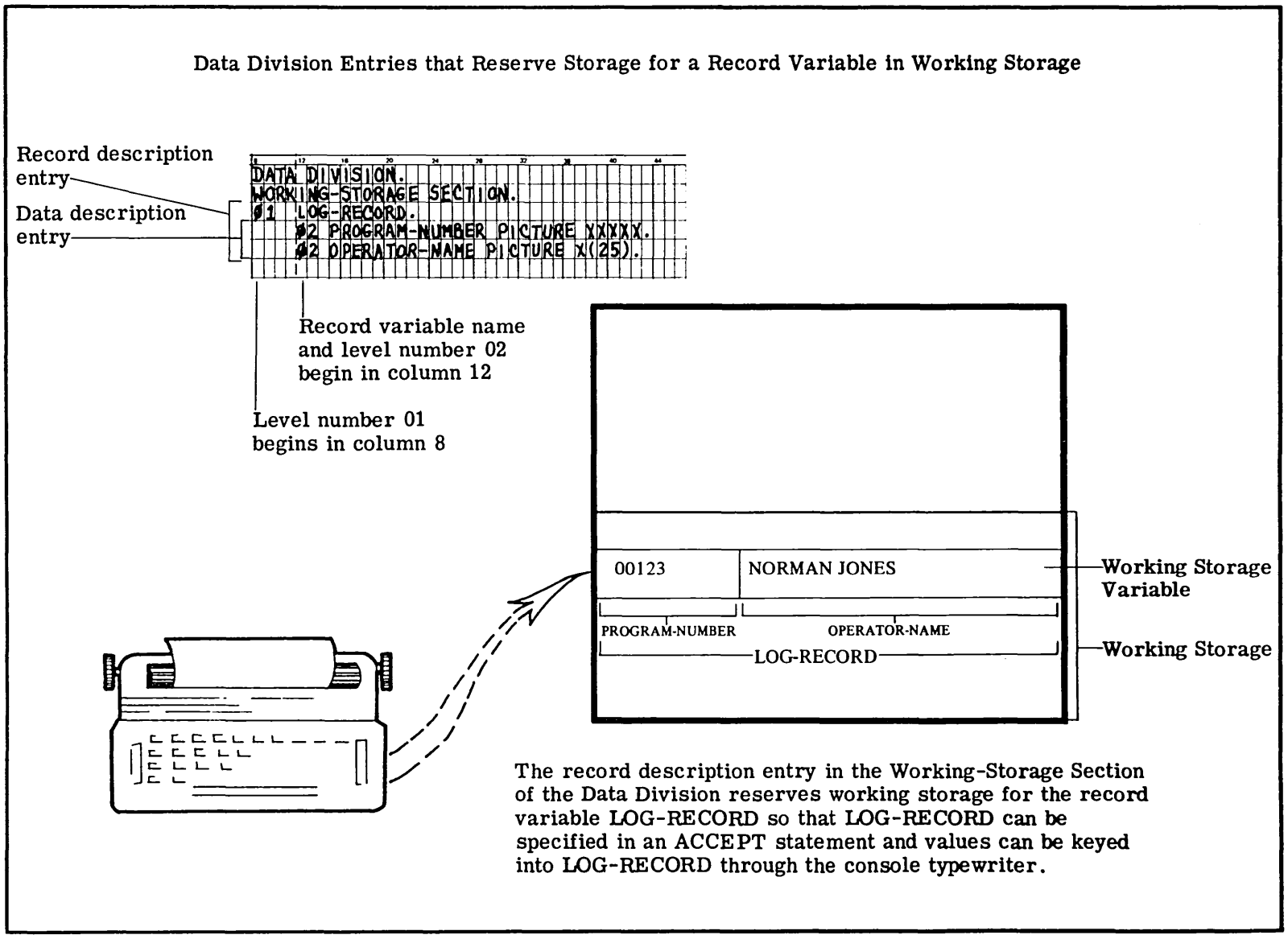
a. The record variable CUSTOMER-RECORD will contain various records of data.

b. the elementary variables in CUSTOMER-RECORD will contain values from various cards.

* * *

Both

Data Division Entries that Reserve Storage for a Record Variable in Working Storage



The record description entry in the Working-Storage Section of the Data Division reserves working storage for the record variable LOG-RECORD so that LOG-RECORD can be specified in an ACCEPT statement and values can be keyed into LOG-RECORD through the console typewriter.

Figure 17

9. Figure 17 shows how storage is reserved for a record variable. Read the explanation and look at the illustration in Figure 17. The variables PROGRAM-NUMBER and OPERATOR-NAME are grouped into the record variable LOG-RECORD which is:

   a. a variable containing a record of data.

   b. a location in storage that will contain a record of data.

* * *

Both

---

10. A record that is a value of LOG-RECORD shown in Figure 17 will consist of:

   a. two elementary values.

   b. values for both PROGRAM-NUMBER and OPERATOR-NAME.

* * *

Both

---

11. If values of the variables in LOG-RECORD are to be keyed in through the console typewriter, storage for these variables would be reserved with entries in:

   a. the Configuration Section of the Environment Division.

   b. the Working-Storage Section of the Data Division.

* * *

b

---

12. Figure 17 shows the Data Division entries that reserve storage for the record variable LOG-RECORD. In these entries:

   a. only the elementary variables are described with the PICTURE clause.

   b. an entry with the level number 01 contains no PICTURE clause if it is further subdivided.

* * *

Both
(Group items do not contain a PICTURE clause; elementary items do.)

---

13. According to Figure 17:

    a. an entry for a record variable is indicated by the level number 01.

    b. an entry for an elementary variable is indicated by the level number 02.

* * *

Both
[If a programmer wishes to treat elementary variables as independent variables he defines them as level 77 in working storage. If he wishes to treat them as part of a record variable, he defines them as level 02 (or 03 or lower, depending on the levels of subdivision) in the record description entry.]

---

14. In Figure 17, you can see that:

    a. a level 01 entry begins in column 8.

    b. a level 02 entry begins in column 12.

* * *

Both

---

15. Figure 17 shows that:

    a. a record description entry usually consists of both level 01 and level 02 entries.

    b. a data description entry consists of an entry for an elementary variable.

* * *

Both

---

16. Match the bracketed portions of the Data Division entries below with the correct type of entry.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        DATA DIVISION.
        WORKING-STORAGE SECTION.
    ┌   01  CODE-DATA.
1)  │  ┌    02  DATE PICTURE XXXXXXXX.
    └ 2)└    02  JOB-CODE PICTURE XXXXXX.
```

    a. Data description entries

    b. Record description entry

* * *

1) b
2) a

17.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CODE-DATA.
           02  DATE PICTURE XXXXXXXX.
           02  JOB-CODE PICTURE XXXXXX.
```

The Data Division entries above:

a. reserve an area in working storage for the record variable CODE-DATA.

b. reserve locations in storage for the elementary variables DATE and JOB-CODE.

* * *

Both

---

18. In Figure 17 the PICTURE clause for OPERATOR-NAME specifies that values of the variable will be 25 characters long. The number enclosed in parentheses is called a repetition factor. A PICTURE clause for PROGRAM-NUMBER could also be written with a repetition factor. Code the data item description entry for the variable PROGRAM-NUMBER using the repetition factor.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02 PROGRAM-NUMBER PICTURE X(5).
```

Guide for Coding the Data Division with Level 01 and
Level 02 Entries in the Working-Storage Section

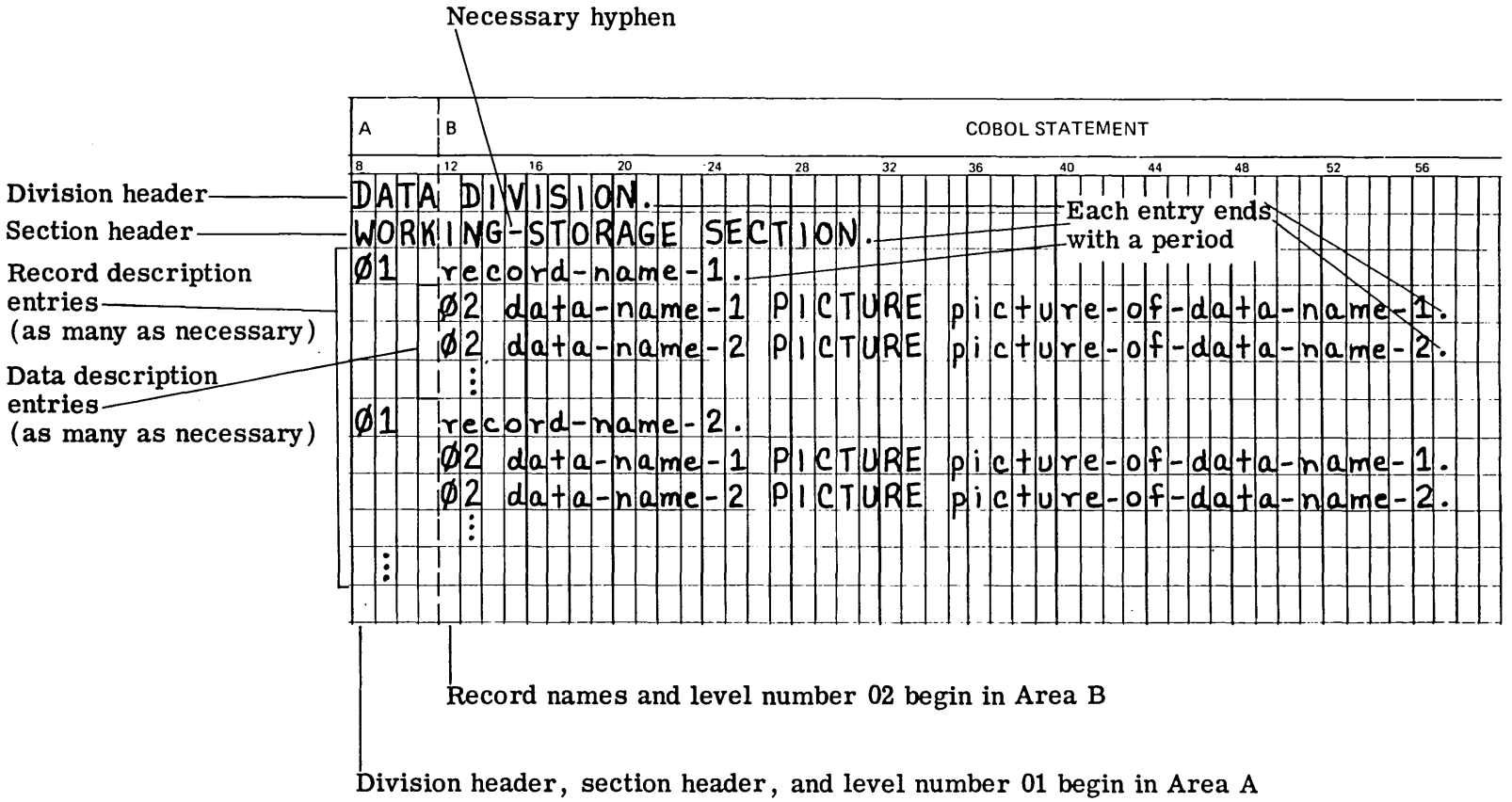


Figure 18

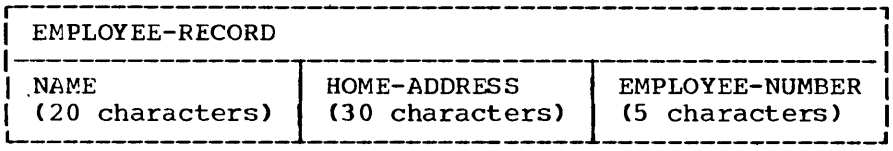


Figure 18 is a guide for coding the Data Division with level 01 and level 02 entries in the Working-Storage Section.

Using the guide in Figure 18, determine which Data Division entries for the record variable diagramed above are correct.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  EMPLOYEE-RECORD PICTURE X(55).
           02  NAME PICTURE X(25).
           02  HOME-ADDRESS PICTURE X(30).
           02  EMPLOYEE-NUMBER PICTURE XXXXX.
```

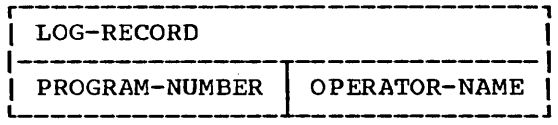
b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  EMPLOYEE-RECORD.
           02  NAME PICTURE X(20).
           02  HOME-ADDRESS PICTURE X(30).
           02  EMPLOYEE-NUMBER PICTURE X(5).
```

* * *

b
(a is incorrect because the level 01 entry contains a PICTURE clause for a group variable.)

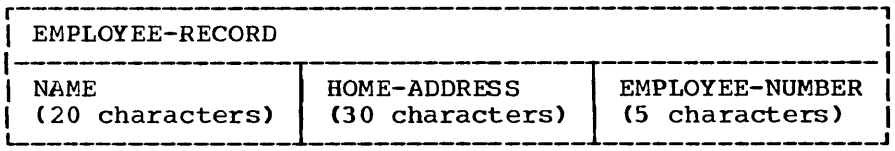
20.




```
0     0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    ACCEPT LOG-RECORD FROM CONSOLE.
```

The statement above will allow values of PROGRAM-NUMBER and OPERATOR-NAME to be keyed into the working-storage variable LOG-RECORD through the console typewriter.




```
0     0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    ACCEPT EMPLOYEE-RECORD FROM CONSOLE.
```

This statement will allow:

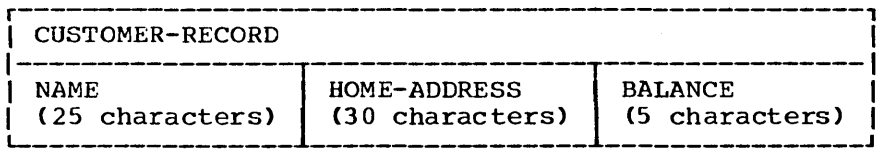
a. values of the elementary variables in EMPLOYEE-RECORD to be entered into working storage through the console typewriter.

b. values of NAME, HOME-ADDRESS, and EMPLOYEE-NUMBER to be keyed into working storage through the console typewriter.

* * *

Both

21.



Using Figure 18 as a guide, write the Data Division entries for the working-storage variable CUSTOMER-RECORD illustrated above. Then write the Procedure Division entries for the paragraph SEQUENCE-1 to allow values to be keyed into the variable through the console typewriter and to halt execution.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  CUSTOMER-RECORD.
           02  NAME PICTURE X(25).
           02  HOME-ADDRESS PICTURE X(30).
           02  BALANCE PICTURE X(5).

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       SEQUENCE-1.
           ACCEPT CUSTOMER-RECORD FROM CONSOLE.
           STOP RUN.
```

---

22. The ACCEPT statement may be used for low-volume input from a card reader as well as from the console typewriter. If the FROM option is not specified, the card reader is assumed to be the device.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ACCEPT CUSTOMER-RECORD.
```

would transmit values to a working-storage variable:

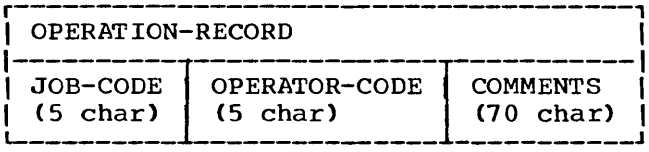
a. from a punched card.

b. through a card reader.

* * *

Both

23.



Write the Data Division entries to reserve working storage for the record variable OPERATION-RECORD illustrated above. Then write the Procedure Division entries for the paragraph MAIN-SEQUENCE to transmit values to OPERATION-RECORD through the card reader and to halt execution.

```
                         *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  OPERATION-RECORD.
           02  JOB-CODE PICTURE X(5).
           02  OPERATOR-CODE PICTURE X(5).
           02  COMMENTS PICTURE X(70).


       PROCEDURE DIVISION.
       MAIN-SEQUENCE.
           ACCEPT OPERATION-RECORD.
           STOP RUN.
```

SUMMARY:

The COBOL language features you have learned to use thus far can be used for low-volume data such as operator messages and replies. Processing files of records that involve a large volume of data, however, requires certain entries in the Environment and Data Divisions as well as certain statements in the Procedure Division. You will learn to code these in the following lesson.

END OF LESSON 4

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 5

LESSON 5 - INTRODUCTION TO FILE PROCESSING

INTRODUCTION

The COBOL language features you have learned to use thus far can be used for low-volume data such as operator messages or replies. Processing files of records that involve a large volume of data, however, requires certain entries in the Environment and Data Divisions as well as certain statements in the Procedure Division. Although the ACCEPT statement can be used to transmit values in a card record to the computer, it is not used to process a file of card records.

You have already learned to code the Configuration Section of the Environment Division in which the programmer specifies the equipment on which the program is to be compiled and executed. In this lesson you will learn to code the Input-Output Section of the Environment Division in which you specify the equipment, such as a printer or a disk drive, that is to be used for a file during execution of a program.

Specific COBOL language features you will learn to use in this lesson are:

Input-Output Section of the Environment Division
SELECT Clause
ASSIGN Clause
File Section of the Data Division
FD Entry
LABEL RECORDS Clause
OPEN Statement
CLOSE Statement
MOVE Statement
WRITE Statement

This lesson will require approximately one hour.

1. A collection of records stored on an external medium such as cards, disk or printer pages is known as a file. Which of these might be a file?

   a. A printed report in which each line contains a customer record

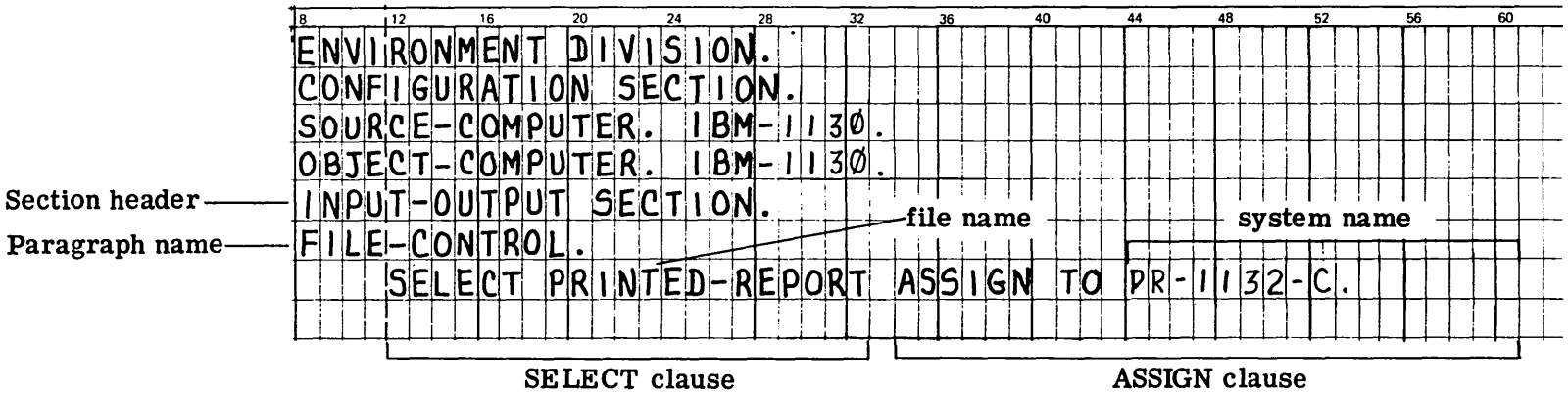
   b. A printed report in which each line contains an employee record

* * *

Both

Figure 19

Environment Division with Sample Input-Output Section



2. A programmer uses a file name such as LIST-OF-CUSTOMERS to refer to a file in his program. He must link this file name to the equipment to be used for the file in the Input-Output Section of the Environment Division. Figure 19 shows a sample Input-Output Section. The file name in Figure 19 is:

   a. PRINTED-REPORT.

   b. specified in the SELECT clause.

* * *

Both

---

3. The ASSIGN clause in Figure 19 specifies a:

   a. 1132 printer.

   b. 1442 reader.

* * *

a

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
         .
         .
         .
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT LIST-OF-CUSTOMERS
               ASSIGN TO PR-1132-C.
```

The entries above:

a. link the file named LIST-OF-CUSTOMERS to an output device.

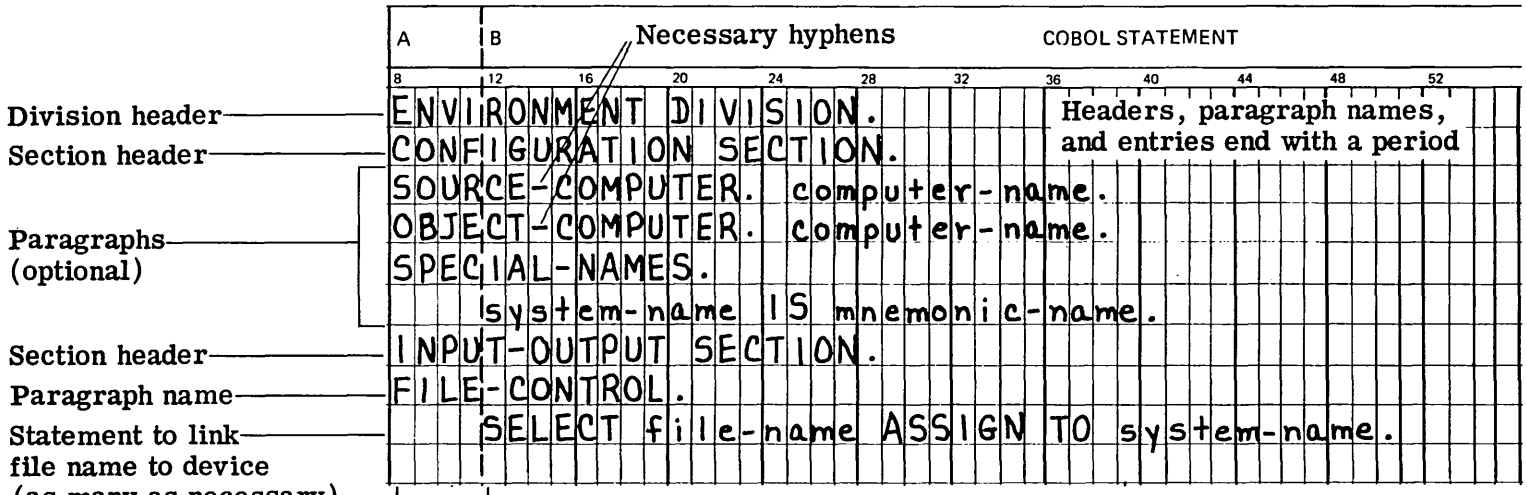
b. specify that LIST-OF-CUSTOMERS will be on an 1132 printer.

c. describe the format of records within the file LIST-OF-CUSTOMERS.

* * *

a,b

Guide for Coding the Environment Division with the Configuration Section and the Input-Output Section Containing the SELECT and ASSIGN Clauses



More than one space is allowed whenever a space is indicated.

Figure 20

5. Figure 20 is a guide for coding the Environment Division with the Configuration Section and the Input-Output Section. The input or output device to be used for a file is specified by the system name in the ASSIGN clause as shown in Figure 20. The ASSIGN Clause Guide in Figure 21 specifies how the system name is to be written for users of the 1130 System. For practice in using the ASSIGN Clause Guide for a printer file, find the printer device name used for the 1132 printer.

   All files used in a program must be assigned to an external medium. That assignment is accomplished by means of the system-name. For non-disk devices, system-name has the following form:

RD-1442	for 1442-6/7, used in this program for reading only.
PU-1442	for 1442-6/7, used in this program for punching only.
RP-1442	for 1442-6/7, used in this program for reading and punching.
PO-1442	for 1442-5 (a punch-only device).
RD-2501	for 2501 card reader.
PR-1132	for 1132, where no carriage control is to be used in this program.
PR-1132-C	for 1132, where carriage control is to be used in this program.
PR-1403	for 1403, where no carriage control is to be used in this program.
PR-1403-C	for 1403, where carriage control is to be used in this program.

   For a disk file, the form of system-name is somewhat different. Three facts must be specified in this name:

   1) The file number of the file (to be equated with an actual file by means of an *FILES supervisor control record at XEQ time.)

   2) The number of record slots (to be) allocated for the file on disk.

   3) Whether the file is to use a shared disk buffer during execution, or its own unique disk buffer.

   The form of system-name for a disk file is:

   DF-FILENUMBER-numberofrecords (-X)

   where:

   filenumber is the number of the file to be equated at XEQ time; the number must be in the range 1 thru 32767 and be written without preceding zeros.

   numberofrecords is the number of record slots (to be) allocated for the file; it must be a number in the range 1 thru 32767, written without preceding zeros.

   -X specifies that the file is to have its own unique disk buffer; if -X is not specified, the shared disk buffer will be utilized.

Disk Drives on 1130

Device or Drive No.	Device Location
0	CPU Resident
1	2310 1st Drive
2	2310 2nd Drive
3	2310 3rd Drive
4	2310 4th Drive

Figure 21

---

6. Using Figures 20 and 21 as guides, write the Environment Division entries for a program that will be compiled and executed on an IBM 1130 computer. The program will create a file called LIST-OF-EMPLOYEES on an 1132 printer, using carriage control.

```
                        *       *       *
0   0   1   1   2   2   3   3   4   4   5   5   6   6   7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT LIST-OF-EMPLOYEES
               ASSIGN TO PR-1132-C.
```

When a -C appears next to print-unit number, carriage control is to be used in this program.

---

7. The ASSIGN Clause Guide in Figure 21 specifies how the system name is to be written for users of the 1130 System.

Using Figures 20 and 21 as guides, write the Input-Output Section for a printer file called PRINTED-REPORT.

```
                        *       *       *
0   0   1   1   2   2   3   3   4   4   5   5   6   6   7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C.
```

---

8. Write the necessary Environment Division entries, including both the Configuration Section and Input-Output Section, for a program that will:

   1) be compiled and run on an IBM 1130 computer.

   2) create a file called CUSTOMER-FILE on the 1132 printer using carriage control.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-1130.
OBJECT-COMPUTER. IBM-1130.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CUSTOMER-FILE
        ASSIGN TO PR-1132-C.
```

(The ASSIGN clause is part of the SELECT entry and may be written below it.)

Storage for an output area must be reserved in the File Section of the Data Division in order for a record (values of the variable PRINT-RECORD) to be transmitted to a line of a printed report.

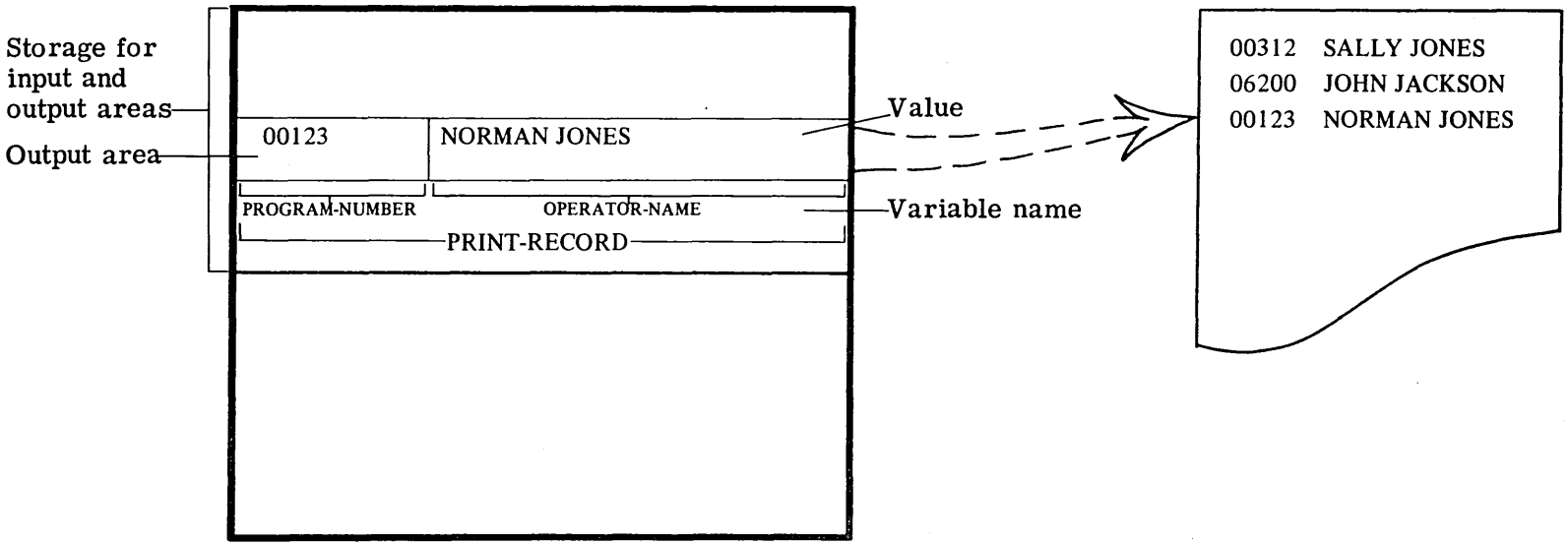


Figure 22

9. According to Figure 22, records on the printer file are values transmitted from the:

   a. record variable LOG-RECORD.

   b. variables in an output area.

   c. variables in PRINT-RECORD.

* * *

b,c

---

10. Various values of a record variable are to be written on a printer. Figure 22 shows values printed on a page from a record variable. From this you can see that:

   a. one record is printed on one line of the printer.

   b. different values of the record variable will be printed on different lines.

* * *

Both

---

11. When a file is being used in a COBOL program, it must be described in a file description entry in the Data Division. The entry on line 03 in Figure 23 names the file being used as PRINTED-REPORT.

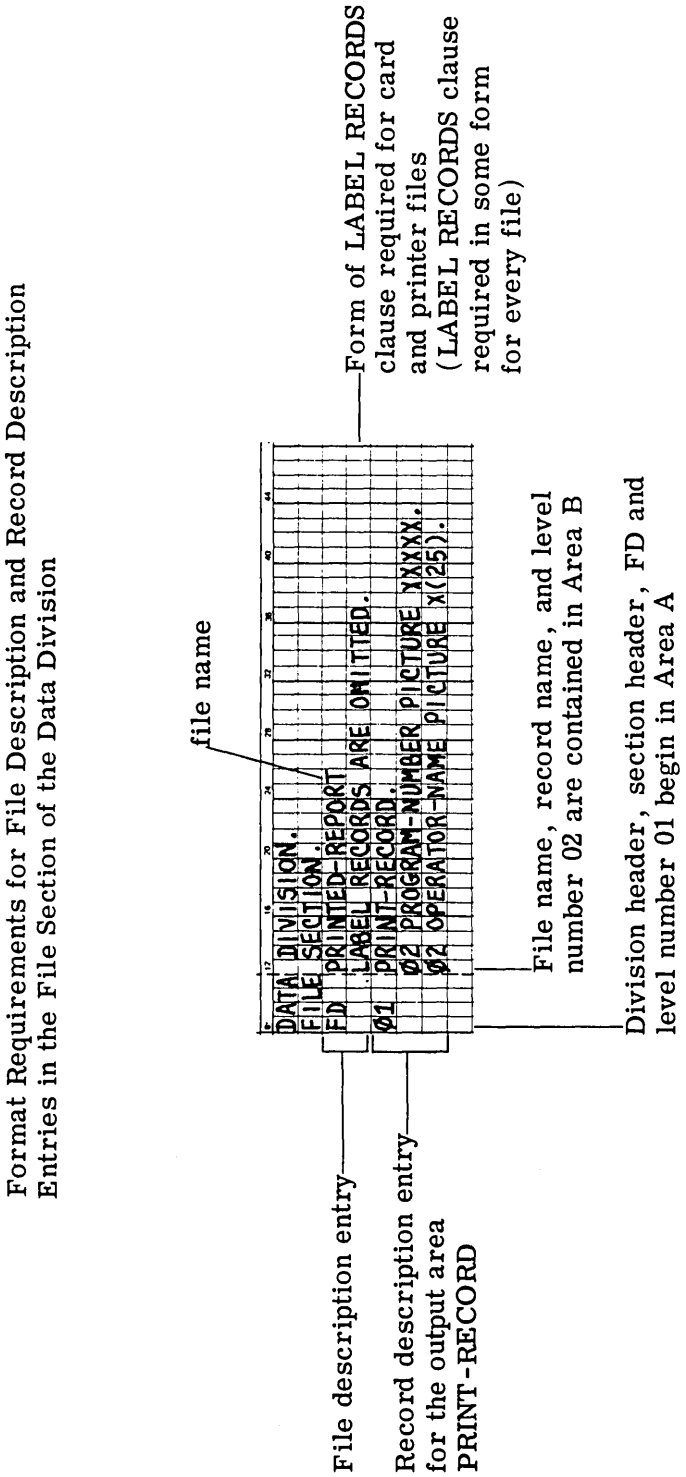


Figure 23

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       FD  PRINTED-REPORT
```

The entry above:

a. designates that the file being used is a printer file.

b. indicates that the name of the file being used is PRINTED-REPORT.

* * *

b

---

12. In the FD entry the programmer must specify whether the file contains records used to label the file in addition to the records of data. He specifies this in a LABEL RECORDS clause. The LABEL RECORDS clause on line 04 of Figure 23 indicates that the file PRINTED-REPORT has no records used to label the file. This form of the LABEL RECORDS clause is used in the FD entry for every card and printer file. Following the example in Figure 23, write the division and section headers and the file description entry for the printer file PRINT-OUT.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       DATA DIVISION.
       FILE SECTION.
       FD  PRINT-OUT
           LABEL RECORDS ARE OMITTED.
```

(Since the LABEL RECORDS clause is part of the FD entry, a period follows the clause, not the file name. Card and printer files do not require label records.)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DATA RECORD IS record-name
           RECORD CONTAINS integer CHARACTERS
```

The clauses shown above may be used in a file description (FD) entry at any time. These clauses are never required and serve only as documentation to assist in reading the program. The DATA RECORD clause is used to name the record associated with the file, while the RECORD CONTAINS clause is used to specifiy the number of characters in each record.

13.

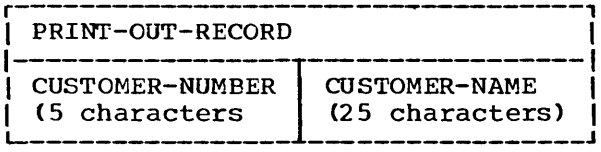


Figure 23 shows a record description entry following the file description entry. This entry describes the output area from which records will be transmitted to the printer file. The programmer has associated the output area PRINT-RECORD with the file PRINTED-REPORT by coding the file description entry followed by the record description entry in the File Section of the Data Division. Follow the example in Figure 23 and code the Data Division entries to specify that the file PRINTER-OUTPUT is to contain records written from the output area illustrated above.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  PRINTER-OUTPUT
           LABEL RECORDS ARE OMITTED.
       01  PRINT-OUT-RECORD.
           02  CUSTOMER-NUMBER PICTURE X(5).
           02  CUSTOMER-NAME PICTURE X(25).
```

---

14. Data Division entries in the File Section reserve:

    a. locations in working storage.

    b. an output area.

* * *

b

---

15. A record description entry following the FD entry describes variables from which values can be transmitted to either the printer or the console typewriter. Values transmitted to a file (a large volume of data) would be transmitted to:

    a. the printer.

    b. the console typewriter.

* * *

a

---

16. You have learned to code the Data Division entries describing a record variable in the Working-Storage Section and an output area in the File Section. The statements in the Procedure Division are used to process data from the record variables described in the Data Division. Before a file may be used, however, it must be prepared for processing. This is done at the beginning of the Procedure Division with an OPEN statement as shown in Figure 24. Write a statement to prepare the printer file PRINT-FILE for processing.

Sample Procedure Division with Entries for Writing a Record into an Output File

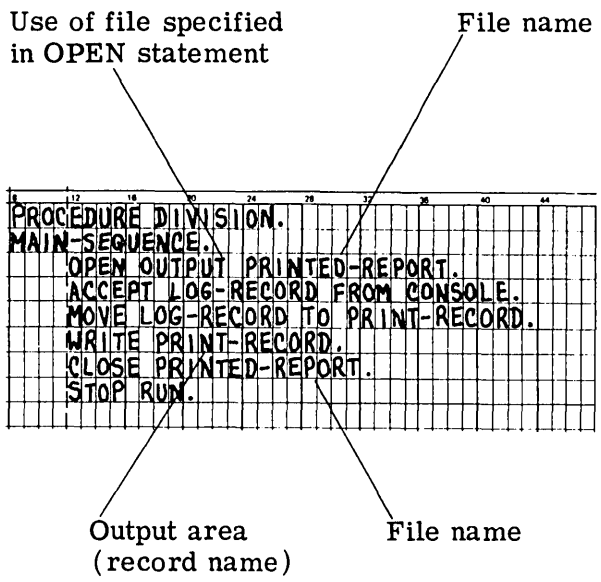


Figure 24

```
                                *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           OPEN OUTPUT PRINT-FILE.
```

(With the exception of paragraph names, statements in the Procedure Division must be contained in Area B.)

---

17. An OPEN statement must be executed before the use of:

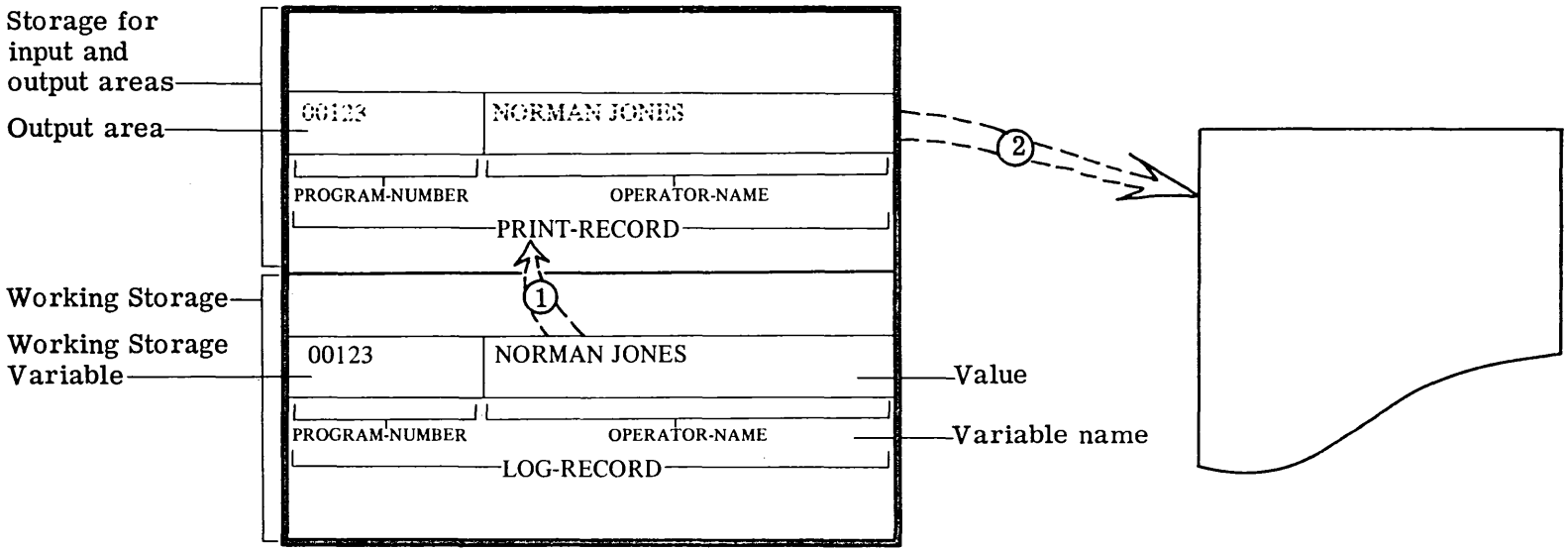
    a. a file on the printer.

    b. the console typewriter.

                                *   *   *

a

This diagram illustrates execution of the MOVE and WRITE statements. These statements can be used to write a record in a working storage variable into a printer output file.



① Values of the elementary variables in LOG-RECORD are to be transferred to the output area PRINT-RECORD by a MOVE statement

② Values of the elementary variables in PRINT-RECORD are to be transmitted to the printer output file by a WRITE statement specifying PRINT-RECORD

Figure 25

18. Figure 25 shows a record in:

    a. working storage.

    b. the output area PRINT-RECORD.

                    *   *   *

a

---

19. Figure 25 shows that in order for the record in working storage to be transmitted to the printer file, the record in LOG-RECORD must be moved to PRINT-RECORD by a ..... statement and then transmitted to the printer by a ..... statement.

                    *   *   *

MOVE,WRITE

---

20. When the MOVE statement in Figure 25 is executed, the value of the variable LOG-RECORD is copied into the output area PRINT-RECORD. The value of LOG-RECORD remains the same, but any previous value of PRINT-RECORD is destroyed.

    The value of the variable NAME is to be transmitted to the variable NAME-FOR-OUTPUT. NAME contains the value SMITH; NAME-FOR-OUTPUT presently contains the value JONES.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MOVE NAME TO NAME-FOR-OUTPUT.
```

    After the execution of the statement above:

    a. NAME and NAME-FOR-OUTPUT will both contain the value JONES.

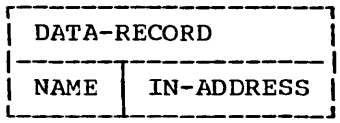
    b. NAME will contain blanks, and NAME-FOR-OUTPUT will contain the value SMITH.

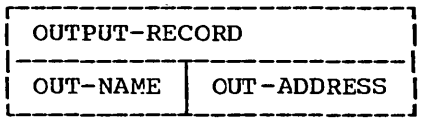
    c. NAME and NAME-FOR-OUTPUT will both contain the value SMITH.

                    *   *   *

c

21.





The variables specified in a MOVE statement may be either record variables or elementary variables. Values of one record variable may be moved to another record variable by a single MOVE statement, or they may be moved as elementary values by more than one statement. Which of the coding segments below would transfer values of the variables in DATA-RECORD, shown above, to the variables in OUTPUT-RECORD?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE DATA-RECORD TO OUTPUT-RECORD.
```

b.

```
0   0    1    1    12   2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE NAME TO OUT-NAME.
           MOVE IN-ADDRESS TO OUT-ADDRESS.
```

* * *

Either

Literals as well as variables can be specified in MOVE statements as shown in the statements below.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE 'GOOD' TO CREDIT-RATING.
           MOVE 1000 TO MAXIMUM-CREDIT.
```

---

22. When the file has been opened and data has been moved into an output area, a record can be written on the file. The WRITE statement is an instruction to write a record on a file. The WRITE statement in Figure 24 will cause the record in the output area PRINT-RECORD to be written on the printer. The statement

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE DATA-RECORD
```

would cause:

a. values in the output area called DATA-RECORD to be printed.

b. a record to be written.

* * *

Both

---

23. When all records have been written into a file, processing must be terminated by a CLOSE statement for that file. The CLOSE statement in Figure 24 terminates processing for the file called PRINTED-REPORT. The statement

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           CLOSE PRINTED-REPORT
```

would be:

a. coded so that it would be executed after the last record has been written into the file called PRINTED-REPORT.

b. used to prepare the file PRINTED-REPORT for processing.

* * *

a

---

24. Figure 24 shows that the word OUTPUT used to specify the use of the file is coded in the statement to:

a. prepare the file PRINTED-REPORT for processing.

b. terminate processing of the file PRINTED-REPORT.

* * *

a

---

25. Follow the example in Figure 24 and write the Procedure Division entries, in a paragraph called SEQUENCE-OF-STEPS, to do the following:

    1) Prepare an output printer file called LIST-OF-CUSTOMERS for processing.

    2) Key values into the record variable CUSTOMER-RECORD.

    3) Transfer values from CUSTOMER-RECORD to an output area called PRINT-RECORD.

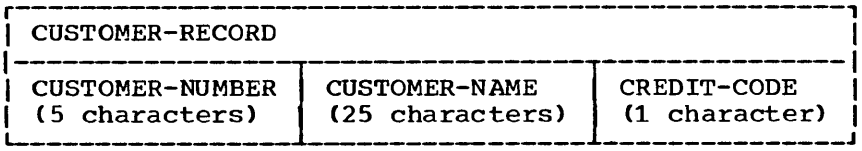
    4) Write a record from PRINT-RECORD on the printer.

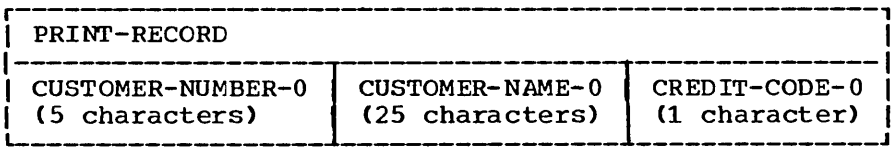
    5) Terminate processing of the output file.

    6) Halt execution of the object program.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       SEQUENCE-OF-STEPS.
           OPEN OUTPUT LIST-OF-CUSTOMERS.
           ACCEPT CUSTOMER-RECORD FROM CONSOLE.
           MOVE CUSTOMER-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           CLOSE LIST-OF-CUSTOMERS.
           STOP RUN.
```

26.

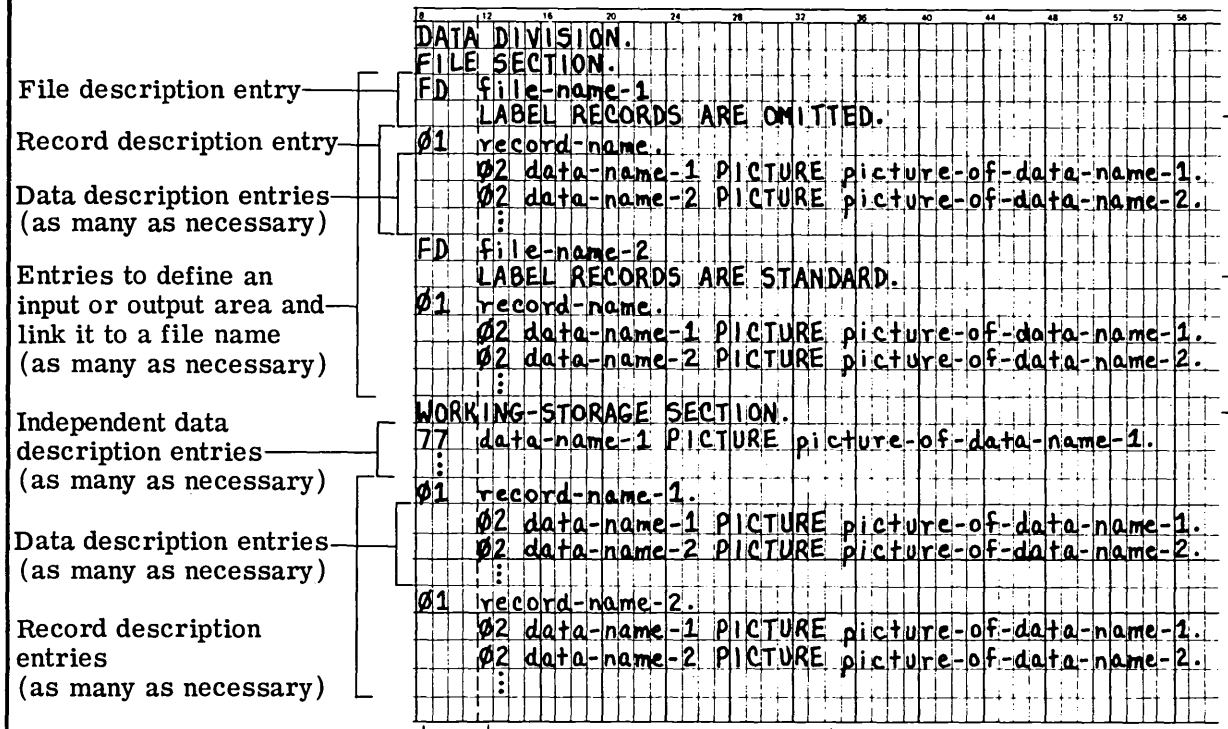




The record variable CUSTOMER-RECORD illustrated above is to contain values transmitted from the card reader. The values will then be transferred to the variables in PRINT-RECORD, also illustrated above. Records contained in the output area PRINT-RECORD will be written into the file LIST-OF-CUSTOMERS.

Figure 26 is a guide for coding the Data Division with the File Section and the Working-Storage Section. Follow the guide in Figure 26 and code the Data Division entries for the record variables described above.

Guide for Coding the Data Division with the
File Section and the Working-Storage Section



FILE SECTION must precede WORKING-STORAGE SECTION if both are included in a program. Within the Working-Storage Section, any level 77 entries must precede any record description entries.

Figure 26

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  LIST-OF-CUSTOMERS
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD.
           02  CUSTOMER-NUMBER-O picture XXXXX.
           02  CUSTOMER-NAME-O PICTURE X(25).
           02  CREDIT-CODE-O PICTURE X.
       WORKING-STORAGE SECTION.
       01  CUSTOMER-RECORD.
           02  CUSTOMER-NUMBER PICTURE XXXXX.
           02  CUSTOMER-NAME PICTURE X(25).
           02  CREDIT-CODE PICTURE X.
```

(The File Section always precedes the Working-Storage Section in the Data Division.)

---

27. Code the Procedure Division entries for the problem in the preceding frame. Use PRINTING-ROUTINE as a paragraph name. You may use Figure 16 as a guide.

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PRINTING-ROUTINE.
           OPEN OUTPUT LIST-OF-CUSTOMERS.
           ACCEPT CUSTOMER-RECORD.
           MOVE CUSTOMER-RECORD
               TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           CLOSE LIST-OF-CUSTOMERS.
           STOP RUN.
```

---

28. Match the functional description(s) with the appropriate portion of a COBOL program.

1)	The record description entry following the FD entry in the File Section of the Data Division	a.	Associates the file name with the equipment to be used for the file
2)	OPEN statement in the Procedure Division	b.	Defines the output area from which records will be transmitted to the file
3)	CLOSE statement in the Procedure Division	c.	Prepares the file for processing
4)	SELECT and ASSIGN clauses in the FILE-CONTROL paragraph of the Input-Output Section of the Environment Division	d.	Terminates processing of the file
		e.	Specifies the use of the file as OUTPUT

* * *

1) b
2) c,e
3) d
4) a

---

29. Read the problem statement in Figure 27. Then on a new coding sheet, write the Identification and Environment Division entries for the problem.

Problem Statement

EMPLOYEE-RECORD			
EMPLOYEE-NUMBER (4)	NAME (20)	ADDRESS-I (30)	MARITAL-STATUS (1)

PRINT-RECORD			
EMPLOYEE-NUMBER-O (4)	NAME-O (20)	ADDRESS-O (30)	MARITAL-STATUS-O (1)

1) A program called PRINT-DATA is to be compiled and run on an IBM-1130 computer.

2) An output file EMPLOYEE-DATA-LIST is to be created with a printer.

3) Values of the variables in EMPLOYEE-RECORD (described above) are to be transmitted from the card reader.

4) Values in EMPLOYEE-RECORD are to be transferred to the variables in the output area PRINT-RECORD.

5) A record from PRINT-RECORD is to be written on a printer file called EMPLOYEE-DATA-LIST.

6) The message END-OF-JOB is to be written on the console typewriter.

7) The output file must be prepared for processing before data is written and closed before execution is terminated.

8) Statements in the Procedure Division are to be grouped in the paragraph MAIN-SEQUENCE.

Figure 27

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. PRINT-DATA.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.                (25)
       FILE-CONTROL.
           SELECT EMPLOYEE-DATA-LIST
               ASSIGN TO PR-1132-C.         (28)
```

(The numbers in parentheses to the right of the coding indicate the frames in which the entries were introduced.)

---

30. Write the Data and Procedure Division entries to solve the problem in Figure 27. (Continue on the coding sheet used in the previous frame.)

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  EMPLOYEE-DATA-LIST                        (34)
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD.                            (13)
           02  EMPLOYEE-NUMBER-O PICTURE X(4).
           02  NAME-O PICTURE X(20).
           02  ADDRESS-O PICTURE X(30).
           02  MARITAL-STATUS-O PICTURE X.
       WORKING-STORAGE SECTION.
       01  EMPLOYEE-RECORD.                         (13)
           02  EMPLOYEE-NUMBER PICTURE X(4).
           02  NAME PICTURE X(20).
           02  ADDRESS-I PICTURE X(30).
           02  MARITAL-STATUS PICTURE X.
       PROCEDURE DIVISION.
       MAIN-SEQUENCE.
           OPEN OUTPUT EMPLOYEE-DATA-LIST.           (39)
           ACCEPT EMPLOYEE-RECORD.
           MOVE EMPLOYEE-RECORD                      (42)
               TO PRINT-RECORD.
           WRITE PRINT-RECORD.                       (42)
           DISPLAY 'END OF JOB' UPON CONSOLE.
           CLOSE EMPLOYEE-DATA-LIST.                 (46)
           STOP RUN.
```

(The CLOSE statement could have been coded on the same line as the DISPLAY statement. Your solution may appear different but be correct provided you have followed the rules for placement of entries and for separation and breaking of statements.)

SUMMARY:

You have now completed Lesson 5 in which you have learned COBOL statements and entries necessary to write data records into an output printer file. The function of each of these entries is summarized in the following illustration.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        IDENTIFICATION DIVISION.
        PROGRAM-ID. PRINT-DATA.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. IBM-1130.
        OBJECT-COMPUTER. IBM-1130.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
(1)         SELECT EMPLOYEE-DATA-LIST
            ASSIGN TO PR-1132.
        DATA DIVISION.
        FILE SECTION.
(2)     FD EMPLOYEE-DATA-LIST
            LABEL RECORDS ARE OMITTED.
        01  PRINT-RECORD.
(3)         02  EMPLOYEE-NUMBER-O PICTURE X(4).
            02  NAME-O PICTURE X(20).
            02  ADDRESS-O PICTURE X(30).
            02  MARITAL-STATUS-O PICTURE X.
        WORKING-STORAGE SECTION.
        01  EMPLOYEE-RECORD.
            02  EMPLOYEE-NUMBER PICTURE X(4).
            02  NAME PICTURE X(20).
            02  ADDRESS-I PICTURE X(30).
            02  MARITAL-STATUS PICTURE X.
        PROCEDURE DIVISION.
        MAIN-SEQUENCE.
(4)         OPEN OUTPUT EMPLOYEE-DATA-LIST.
            ACCEPT EMPLOYEE-RECORD.
            MOVE EMPLOYEE-RECORD
                TO PRINT-RECORD.
(5)         WRITE PRINT-RECORD.
            DISPLAY 'END OF JOB' UPON CONSOLE.
(6)         CLOSE EMPLOYEE-DATA-LIST.
            STOP RUN.
```

(1) Links file name to equipment to be used for the file
(2) Specifies whether file contains records used to label the file
(3) Defines an output area and associates it with file name
(4) Prepares file for processing and specifies its use as output
(5) Write record into file associated with PRINT-RECORD in the File Section
(6) Terminates processing of the file

The ACCEPT statement which is used for low-volume data such as operator messages and replies, has been used in this lesson to transmit records of data from the card reader to working storage record variables. It has been used to allow you to practice writing statements for output on a printer file. In the next lesson you will learn to code the entries to transmit data from an input file to an input area which will be defined in the File Section.

END OF LESSON 5

LESSON 6

LESSON 6 - CARD FILE PROCESSING AND BRANCHING

INTRODUCTION

In previous lessons the input data used in the COBOL programs you wrote was keyed in through the console typewriter or accepted from the card reader as a single card. Usually input data will be contained in records on an external medium called a file. The input file may be on punched cards, or disk. The punched card is a widely used medium in data processing. Processing data that is recorded in a punched card input file is a more efficient method than accepting single cards or records from the card reader, which would be used only for low-volume data.

In this lesson you will learn to process input data from a card file. You will also learn to branch to another point in a COBOL program and to repeat a series of steps.

Specific COBOL language features you will learn to use in this lesson are:

PIC abbreviation
FILLER item in an input area
INPUT option of the OPEN statement
READ statement
GO TO statement

This lesson will require approximately three quarters of an hour.

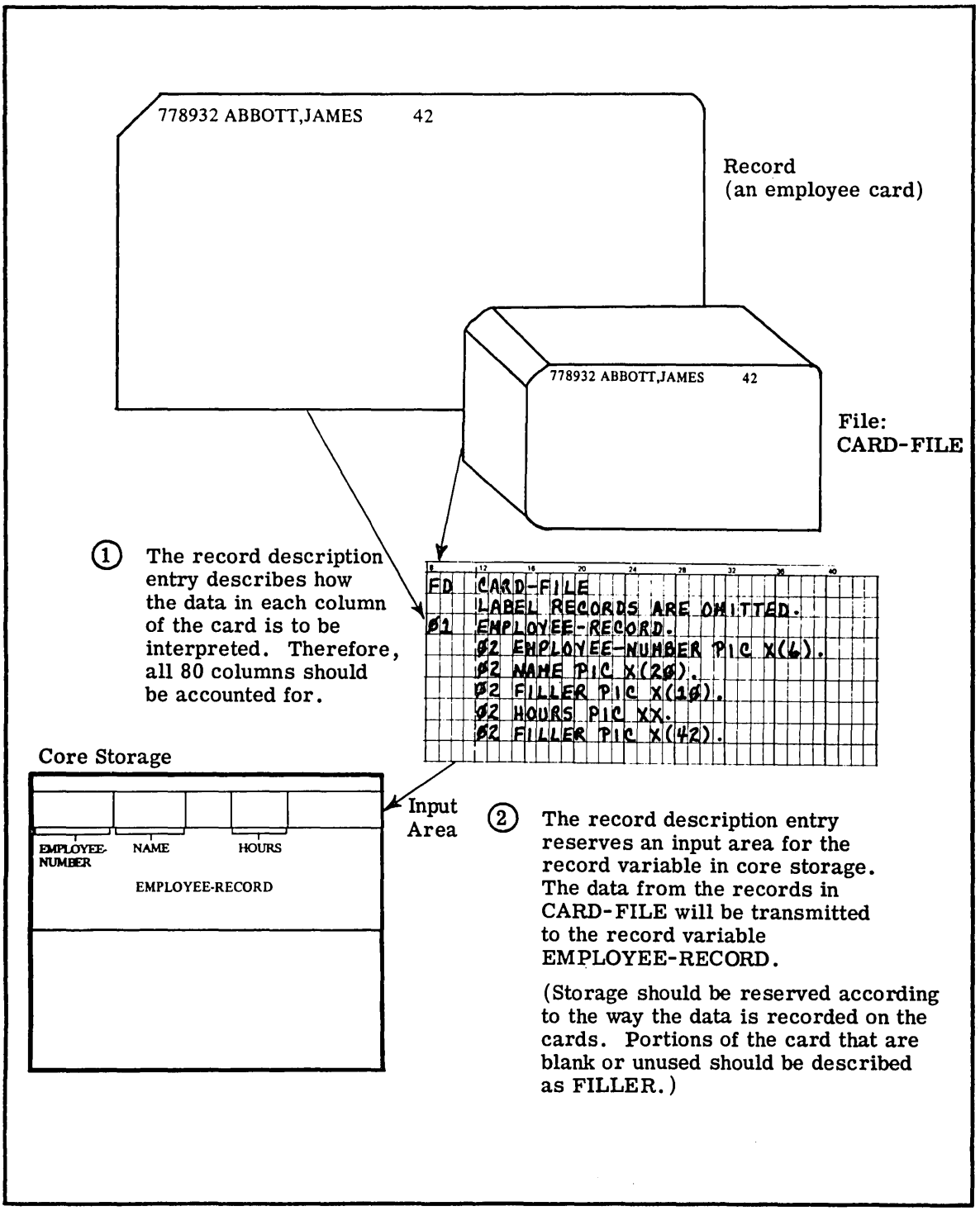

778932 ABBOTT,JAMES 42
Record
(an employee card)
778932 ABBOTT,JAMES 42
File:
CARD-FILE
1 The record description entry describes how the data in each column of the card is to be interpreted. Therefore, all 80 columns should be accounted for.
FD CARD-FILE
LABEL RECORDS ARE OMITTED.
01 EMPLOYEE-RECORD.
02 EMPLOYEE-NUMBER PIC X(4).
02 NAME PIC X(20).
02 FILLER PIC X(10).
02 HOURS PIC XX.
02 FILLER PIC X(42).
Core Storage
Input Area
EMPLOYEE-NUMBER
NAME
HOURS
EMPLOYEE-RECORD
2 The record description entry reserves an input area for the record variable in core storage. The data from the records in CARD-FILE will be transmitted to the record variable EMPLOYEE-RECORD.
(Storage should be reserved according to the way the data is recorded on the cards. Portions of the card that are blank or unused should be described as FILLER.)

Figure 28

1. Figure 28 shows a deck of cards. This deck of cards is a file. According to Figure 28, the file shown is:

   a. CARD-FILE

   b. EMPLOYEE-RECORD

* * *

a

---

2. Each card in a card file is a separate record. Figure 28 shows an illustration of a single card, or record, from the file CARD-FILE. Each card in the file CARD-FILE is:

   a. an employee card.

   b. a record.

* * *

Both

---

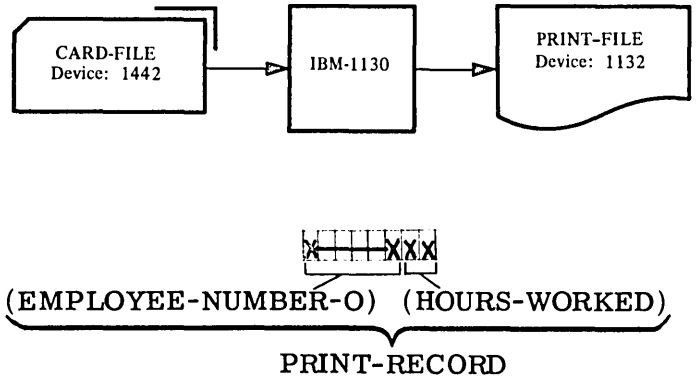
3. An entire deck of cards is a ..... . Each card within the deck is a ..... .

* * *

file
record

Read the problem statement in Figure 29. In the following frame sequence you will code this problem in COBOL. Continue to code on the same coding form as you build the program.

As a COBOL programmer, you will probably be required to write programs to process data that is recorded on punched cards. It is also a common practice to list on the printer the data, or a portion of the data, recorded on punched cards.



The flowchart above is a system flowchart. It gives you necessary information about the equipment to be used during execution of the program LISTING, described below.

The file CARD-FILE, whose records are illustrated in Figure 28, is to be processed in the following way:

1) The record is to be read into the input area EMPLOYEE-RECORD.

2) The employee-number and the hours worked are to be moved to an output record.

3) The output record is to be printed in the following form:

X----XXX

(EMPLOYEE-NUMBER-O) (HOURS-WORKED)

PRINT-RECORD

Figure 29

---

4. The program is to be called LISTING. Write the Identification Division for this program.

* * *

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LISTING.
```

---

5. The next division to be coded is the Environment Division. The computer to be used for compilation and execution is shown in the system flow chart in Figure 29. Code the Configuration Section of the Environment Division. (Remember to include the appropriate division and section headers.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
```

---

6. The program in the previous lesson required only one file, a printer output file. Consequently, only one SELECT clause and one ASSIGN clause were required. In the problem statement in Figure 29 two files are specified. Therefore the program to solve this problem will contain:

   a. two SELECT clauses.

   b. one ASSIGN clause.

* * *

a
(An ASSIGN clause is required for each SELECT clause.)

---

7. The SELECT clause for a card input file is just like the SELECT clause for a printer output file. The appropriate parts of the ASSIGN clause for a card input file are shown in Figure 21. Refer to Figures 20 and 21 and code the Input-Output Section of the Environment Division for the problem in Figure 29. The file names and input and output devices are shown in the system flow chart.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CARD-FILE ASSIGN TO
               RD-1442.
           SELECT PRINTFILE ASSIGN TO
               PR-1132-C.
```

---

8. The next division of the program to be coded is the Data Division. The File Section must contain an FD entry for each file used in the program. In the program you are writing for the problem statement in Figure 29, there should be:

   a. two FD entries.

   b. an FD entry for CARD-FILE.

   c. an FD entry for EMPLOYEE-RECORD.

   d. an FD entry for PRINTFILE.

* * *

a,b,d

---

9. Figure 28 shows the FD entry for the file CARD-FILE. The clause which is shown must always be present. An FD entry always requires a ..... clause.

* * *

LABEL RECORDS (Write the Data Division and File Section headers on your coding form, and then copy the FD entry in Figure 28 into your program. Label records are always omitted from a card file.)

---

10. Figure 28 also shows that the record description entry reserves an area in storage that will contain values from the records in CARD-FILE. The record variable that will contain values from the records in CARD-FILE is:

    a. EMPLOYEE-RECORD.

    b. CARD-FILE.

    c. described in a record description entry.

* * *

a,c
(PIC is a reserved word and a valid abbreviation for PICTURE; these two reserved words are equivalent.)

---

11. The record description entry contains data description entries for elementary variables within the record variable. The elementary variables will each contain specific data from specified fields in the records in CARD-FILE. The pictures associated with the variables determine the size of the field on the card that is to be transmitted to each variable. The data description entries in Figure 28 show that:

    a. six characters from each record will be transmitted to the variable EMPLOYEE-NUMBER.

    b. 10 characters from each record will be transmitted to the variable NAME.

* * *

a

---

12. The record description entry indicates the order in which data is to be transmitted to the record variable. That is, the first field on a card will be transmitted to the first elementary variable named in the record description entry, the second field to the second elementary variable, and so on. Refer to Figure 28 and determine which of the following is true.

    a. The data in the field in columns 1-6 will be transmitted to the variable EMPLOYEE-NUMBER.

    b. The data in the field in columns 7-26 will be transmitted to the variable NAME.

* * *

Both

---

13. The third data description entry in Figure 28 specifies the COBOL reserved word FILLER. The field on the sample card that corresponds to this data description entry:

    a. is blank.

    b. contains characters and digits.

* * *

a

---

14. Some files are used for more than one program and contain data that may not be needed in every program in which the file is used. A FILLER item can never be referred to in a program. Consequently, you would use FILLER in a data description entry when:

    a. the corresponding field in the record contains no data.

    b. the data contained in the field is not to be referred to in the program.

* * *

Either

---

15. In the record description entry in Figure 28, FILLER is used after NAME because:

    a. the data contained in that field is not to be referred to in the program.

    b. there are 10 columns following NAME in the input record that contain no data.

* * *

b

---

16. According to the explanation identified by 1 Figure 28, FILLER is used after HOURS in the record description entry because:

    a. no data is punched into the columns described.

    b. all 80 columns on the card should be accounted for in the record description entry for a card file.

* * *

Both

---

17. The PICTURE clause is used with FILLER just as it is with an elementary-variable name. The PICTURE clause used with FILLER should indicate the:

    a. number of blanks in the related field.

    b. length of the field that is not to be referred to in the program.

* * *

Either

---

18. Although values of NAME are punched into the cards, NAME is not referred to in the program for the problem statement in Figure 29. Therefore FILLER could be used in place of NAME in the record description entry. Write a record description entry for the file in Figure 28 using FILLER in place of NAME. (Do not use the coding form on which you are coding the program for the problem statement in Figure 29.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  EMPLOYEE-RECORD.
           02 EMPLOYEE-NUMBER PIC X(6).
           02 FILLER PIC X(30).
           02 HOURS PIC XX.
           02 FILLER PIC X(42).
```

---

19. The explanation identified by 2 in Figure 28 explains that the record description entry causes:

    a. data to be stored in core.

    b. storage to be reserved in core for the data from the file.

* * *

b

---

20. The record description entry associates the record variable with the file from which data is to be transmitted to the variable. Consequently, you might expect that a record description entry for records from a file in a COBOL program:

    a. must be placed immediately following the FD entry for the file with which it is to be associated.

    b. can be placed anywhere in the Data Division.

* * *

a

---

21. When records on punched cards are processed as a file, the record description entry appears in the ..... Section immediately following the FD entry for its associated file. When records on punched cards are to be accepted from the card reader as single records by an ACCEPT statement, the record description entry appears in the ..... Section.

* * *

File
Working-Storage

---

22. Complete the Data Division of your program by writing the FD entry and the record description entry for the output file described in Figure 29.

```
                         *       *       *
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  PRINTFILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD.
           02 EMPLOYEE-NUMBER-0 PIC X(6).
           02 HOURS-WORKED PIC XX.
```

Although user-supplied words are provided for you in this course, you will form your own as a programmer. Figure 30 shows rules for forming all user-supplied words that are included in the course.

The final division of your COBOL program is the Procedure Division. Figure 31 defines the content and terminating symbols of valid Procedure Division entries. All statements that you have seen thus far have ended with periods, since they were treated as sentences. Statements will continue to be shown ending with periods, but keep in mind that the period is not a required part of a statement.

Rules for Forming User-Supplied Words

User-supplied Name	Use	Number of Characters	Type of Characters from Figure 1	Restrictions
file name	name a file	1 to 30	at least one must be alphabetic; no spaces; cannot begin or end with hyphen	name must be unique
record name	name a record			name must be unique or qualifiable
data name	name a data item			
condition name	name a value of data item			name must be unique
paragraph name	name a paragraph	1 to 30	no alphabetic characters required no spaces; cannot begin or end with hyphen	name must be unique
program name	name a program			first five characters must be unique
library name	name a library entry			

Figure 30

Valid Procedure Division Entries

TERM	DEFINITION	
	CONTENT	TERMINATING SYMBOL(S)
statement	a basic valid combination of words and symbols used in the Procedure Division	a space, a comma followed by a space, or a period followed by a space (If a period is used, the statement is also a sentence.)
sentence	a sequence of one or more statements	a period followed by a space
paragraph	a sequence of one or more sentences, the first one being preceded by a paragraph name	another paragraph name or the end of the program

Figure 31

23. You know that an output file must be opened before you can refer to data in that file. The same is true for an input file. The name of the input file is preceded by the reserved word INPUT in the OPEN statement. Which of the following would be a correct OPEN statement for the input file FILEIN?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           OPEN FILEIN.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           OPEN FILEIN INPUT.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           OPEN INPUT FILEIN.
```

* * *

c

24.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
OPEN INPUT CARD1 OUTPUT PRINTER CARD2.
```

A single OPEN statement can open more than one file. For example, the statement above will cause CARD1 to be opened as an input file and PRINTER and CARD2 to be opened as output files. This demonstrates that:

a. the reserved word INPUT and/or OUTPUT needs to be named only once in an OPEN statement.

b. all the files whose names follow the reserved word INPUT (without an intervening OUTPUT) in an OPEN statement will be opened as input files.

* * *

Both
(The reserved words INPUT and OUTPUT may be used only once in an OPEN statement.)

---

25. Include in your program the appropriate OPEN statement for the files described in Figure 29. The division header for the Procedure Division and a paragraph name (use BEGIN) must also be included.

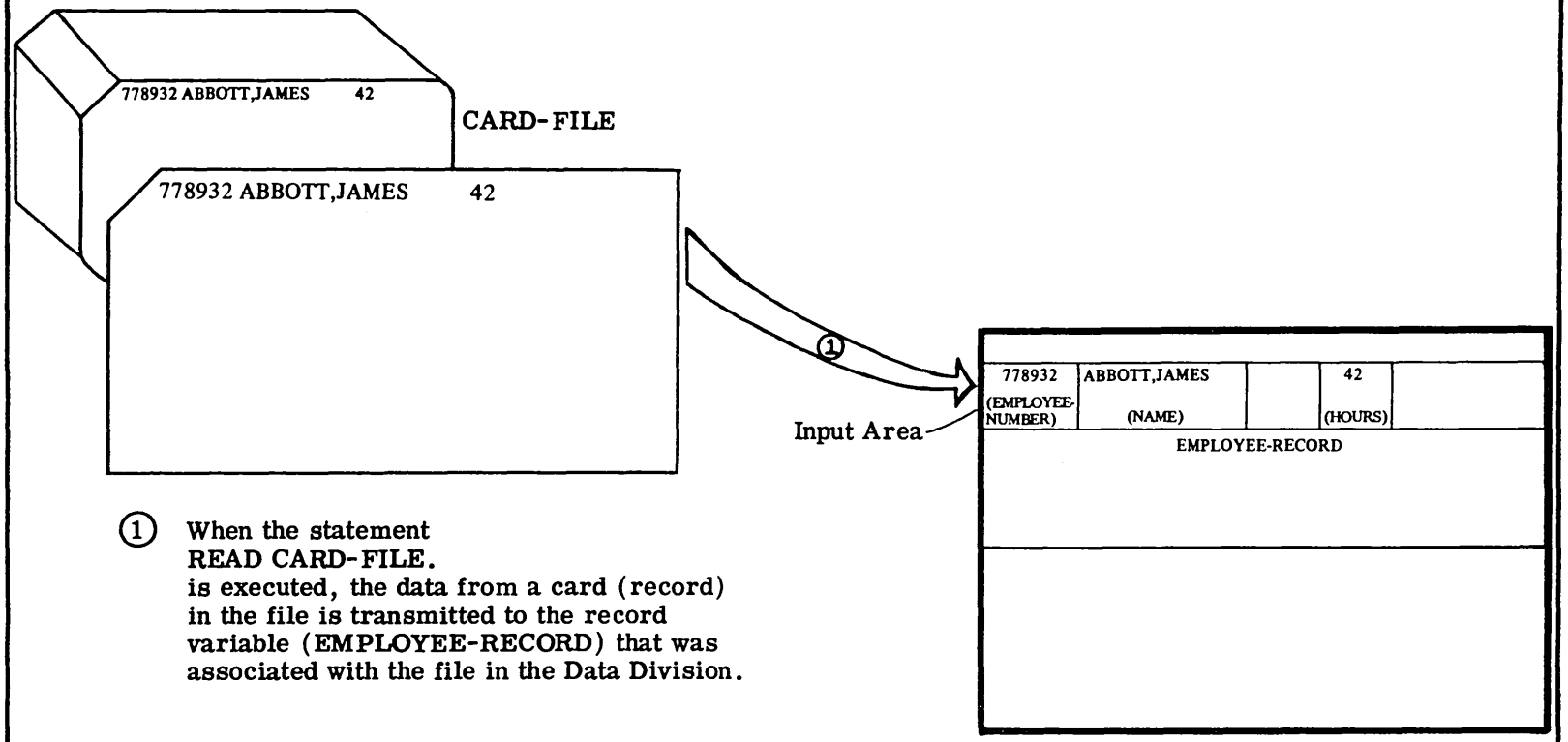
* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT CARD-FILE
         OUTPUT PRINTFILE.
```

An ACCEPT statement is used to instruct the computer to accept data punched into a single card or keyed in from the console. To be able to use the data from a card file, a READ statement must be executed for each record in the file.

This figure illustrates the execution of the statement
READ CARD-FILE.



(1) When the statement
READ CARD-FILE.
is executed, the data from a card (record) in the file is transmitted to the record variable (EMPLOYEE-RECORD) that was associated with the file in the Data Division.

Figure 32

26. The text identified by 1 in Figure 32 explains that the READ statement causes:

    a. data from a card in the file to be transmitted to the input area in core storage set up by the record description entry for the file.

    b. storage to be reserved in core for information from the cards.

* * *

a

27.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

READ file-name

The simple form of the READ statement is shown above. (No terminating period is shown in the form above, since a period is not a required part of the statement.)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

READ CUSTOMER-FILE.

The statement above would cause the next record in the file CUSTOMER-FILE to be stored in the record variable associated with the file. Information contained in that record can subsequently be referred to in the program by references to the appropriate variables. A READ statement:

a. must be executed before data from a record in a file can be processed.

b. causes data to be stored as specified in the Working-Storage Section.

* * *

a

---

28. Write a statement to cause data from a record in the file CARD-FILE to be stored in the record variable associated with the file. (Do not use the coding form on which you are coding the program for the problem statement in Figure 29.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

READ CARD-FILE.

---

29. According to the problem statement in Figure 29 the operations necessary to complete the Procedure Division are:

    a. ACCEPT

    b. READ

    c. MOVE

    d. WRITE

    e. DISPLAY

* * *

b,c,d

---

30. On a separate coding form, write the necessary COBOL statements to:

    1) read the data from a card in CARD-FILE.

    2) move the necessary data as indicated in Figure 29.

    3) print the output record as indicated in Figure 29.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          READ CARD-FILE.
          MOVE EMPLOYEE-NUMBER TO
              EMPLOYEE-NUMBER-0.
          MOVE HOURS TO HOURS-WORKED.
          WRITE PRINT-RECORD.
```

---

31. The statements you have just written will cause one record to be read from CARD-FILE and one record to be printed in PRINTFILE. A file always contains more than one record. In order to complete the data-processing task, the entire file must be processed. That is, every card in the file must be read and the necessary steps performed to process it. This could be specified by:

    a. repeating the steps you have just written

    b. executing a READ statement (and other necessary steps) for each record in the file.

* * *

Either

32.

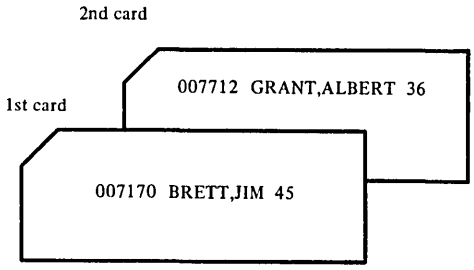


Figure 33

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

  203       READ CARD-FILE.
  204       MOVE EMPLOYEE-NUMBER TO
  205           EMPLOYEE-NUMBER-0.
  206       MOVE HOURS TO HOURS-WORKED.
  207       WRITE PRINT-RECORD.
  208       READ CARD-FILE.
  209       MOVE EMPLOYEE-NUMBER TO
  210           EMPLOYEE-NUMBER-0.
  211       MOVE HOURS TO HOURS-WORKED.
  212       WRITE PRINT-RECORD.
  213
```

The first time a READ statement for a card file is executed, the data from the first card in the file is transmitted to the appropriate input area. Execution of the next READ statement for that file will cause the data from the second card in the file to be stored in the same input area, thus destroying the data from the first record. Using the data descriptions in your program and the cards shown from the input file, you can see that after execution of statement 203 above, the variable NAME will contain .....; after execution of statement 208, NAME will contain ..... .

* * *

```
BRETT,JIM
GRANT,ALBERT
```

33.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

  203       READ CARD-FILE.
  204       MOVE EMPLOYEE-NUMBER TO
  205           EMPLOYEE-NUMBER-O.
  206       MOVE HOURS TO HOURS-WORKED.
  207       WRITE PRINT-RECORD.
  208       READ CARD-FILE.
  209       MOVE EMPLOYEE-NUMBER TO
  210           EMPLOYEE-NUMBER-0.
  211       MOVE HOURS TO HOURS-WORKED.
  212       WRITE PRINT-RECORD.
  213
```

Using the data shown in the previous frame, write the information that will be printed when:

1) statement 207 is executed

2) statement 212 is executed

* * *

1) 00717045 (Employee-Number and Hours)
2) 00771236

---

34. It would be impractical to write all the necessary statements for every card in a file. You might not know how many cards are in the file. Therefore, it would probably be best to:

a. read only one card in the file

b. reexecute statements that process a single card for every card in the file

* * *

b

---

35. Statements in the Procedure Division of a COBOL program are normally executed in the order in which they occur. You might assume that this pattern of sequential execution could be altered by a:

a. branching statement

b. statement causing a transfer of control to another point in the program

* * *

Either

---

36. A COBOL statement used to transfer control to another point in the program is the GO TO statement.

This flowchart illustrates a loop, which is the repetition of a series of steps in a program.

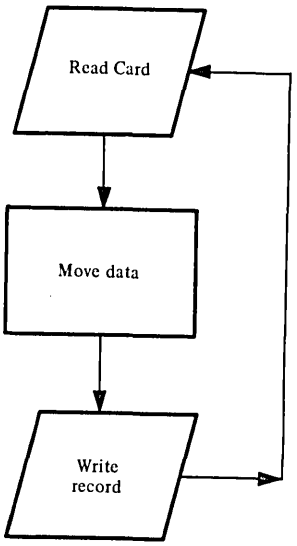


Figure 34

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

GO TO paragraph-name.

The GO TO statement of the form shown above can be used for looping. When a GO TO statement is executed, control transfers to the given paragraph name and execution continues from that point.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN OUTPUT LIST-FILE.
       PROC1.
           ACCEPT NAME FROM CONSOLE.
           MOVE NAME TO OUTPUT-NAME.
           WRITE OUTPUT-RECORD.
           GO TO PROC1.
           CLOSE LIST-FILE.
           STOP RUN.
```

Refer to the Procedure Division above and determine which of the following statements is true.

a. When the GO TO statement is executed, control will transfer to the point where PROC1 appears and all the statements following PROC1 up to the GO TO statement will be reexecuted.

b. The statements shown will be executed once and then execution will stop.

c. If the GO TO statement were placed immediately following the ACCEPT statement, nothing would be printed in the output file.

* * *

a,c

37.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT
               OUTPUT LISTING.
           READ STUDENT.
           MOVE STUDENT-NAME TO LIST-NAME.
           WRITE LISTING-RECORD.
           GO TO BEGIN.
           CLOSE STUDENT LISTING.
           STOP RUN.
```

When the Procedure Division above is executed, the OPEN statement will be executed each time control is transferred by the GO TO statement. An OPEN statement for a file can be executed only once in a program (unless the file is to be closed and then reopened). You know that a paragraph name can precede any statement in the Procedure Division. Using the paragraph name PARA1, rewrite the Procedure Division above specifying that the OPEN statement will be executed only once.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT
               OUTPUT LISTING.
       PARA1.
           READ STUDENT.
           MOVE STUDENT-NAME TO LIST-NAME.
           WRITE LISTING-RECORD.
           GO TO PARA1.
           CLOSE STUDENT LISTING.
           STOP RUN.
```

Alternate solutions:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT
               OUTPUT LISTING.
       PARA1.
           READ STUDENT
           MOVE STUDENT-NAME TO LIST-NAME
           WRITE LISTING-RECORD
           GO TO PARA1
           CLOSE STUDENT LISTING
           STOP RUN.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT
                OUTPUT LISTING.
       PARA1.
           READ STUDENT,
           MOVE STUDENT-NAME TO LIST-NAME,
           WRITE LISTING-RECORD,
           GO TO PARA1,
           CLOSE STUDENT LISTING,
           STOP RUN.
```

By refering to Figure 31, you can see that either of the Procedure Divisions above would also be a correct solution to this frame. Paragraph BEGIN contains only one statement, which must end in a period since a paragraph must contain at least one sentence. Paragraph PARA1 contains six sentences in the original solution and one sentence in the additional solutions. Two,three, four or five sentences in paragraph PARA1 could also be correct.

SUMMARY:

In the preceeding portion of this lesson you have learned to:

1) set up an input area in storage for data contained in an input file.

2) read data from punched cards in an input file.

3) specify that a series of steps in a program is to be repeated so that the entire input file can be processed.

END OF LESSON 6

LESSON 7

LESSON 7 - USE OF RECORD VARIABLES

INTRODUCTION

This lesson will give you increased flexibility in programming techniques by showing you what to do at the end of a program and how to use record variables at levels lower than 01 and 02.

Specific COBOL language features you will learn to use in this lesson are:

AT END option of the READ statement
Level number 03
Level number 04
Qualified names

This lesson will require approximately three quarters of an hour.

There is another situation you must provide for when you are writing a program using card input file. You must specify the action to be taken when the last card in the file has been read.

---

1. The last card in a card file is a card with a special code on it which is understood by the computer to mean "this is the end of the file." Therefore, when a program is processing an input card file, the computer will know that the last card has been processed when:

   a. there are no more cards.

   b. it reads a card with a special end-of-file code.

* * *

b

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

READ file-name AT END imperative-sentence

Although the computer knows when it has reached the end of the file, it doesn't know what to do. You, as a programmer, must use the AT END option of the READ statement to tell the computer what to do when all the records in an input file have been processed. A READ statement with an AT END option has the form shown above. An imperative sentence is one or more imperative statements, ended by a period and followed by a blank.

AT END Option

The AT END option must be specified for all files in the sequential access mode. If, during the execution of a READ statement, the logical end of the file is reached, control is passed to the imperative-statement specified in the AT END phrase. After execution of the imperative statement associated with the AT END phrase, a READ statement for that file must not be given without a prior execution of a CLOSE statement and an OPEN statement for the same file.

If, during the processing of a multivolume disk file in the sequential access mode, end-of-volume is recognized on a READ, the following actions are carried out:

1. A volume switch is made. If the MULTIPLE UNIT option was specified for this file, and if, in order to make the volume switch, it is necessary for the operator to make a physical cartridge change, he is instructed at this time to do so.

2. When the volume switch is complete, the first data record of the new volume is made available.

Determine which of the following could be included in the AT END option.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

            MOVE TOTAL TO TOTAL1 STOP RUN.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

            GO TO PARA2.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

            CLOSE INFILE STOP RUN.
```

* * *

Any of these
(As shown in c an imperative sentence consisting of a string of imperative statements may be included in the AT END option.)

---

3. In order to determine what should be included in the AT END option, you must decide what is to be done (as specified in the problem statement) when the entire input file has been processed. Because all files used in a program should be closed by a CLOSE statement, it would be logical to include a ..... statement in the AT END option.

* * *

CLOSE

---

4. If the program is complete when the entire input file has been processed, you might also include:

   a. STOP RUN in the AT END option.

   b. a MOVE statement in the AT END option.

* * *

a

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

CLOSE file-name file-name...

The CLOSE statement for an input file is just like a CLOSE statement for an output file. Each file in a program may have a separate CLOSE statement, or a single CLOSE statement may close several files, both input and output. In this case, the CLOSE statement has the format shown above. (Do not use the coding form on which you are coding the program for the problem statement in Figure 29.)

1) Write two sentences to close the files STUDENT and LISTING.

2) Write a single sentence to close the files STUDENT and LISTING.

* * *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
CLOSE STUDENT.
CLOSE LISTING.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
CLOSE STUDENT LISTING.
```

6.

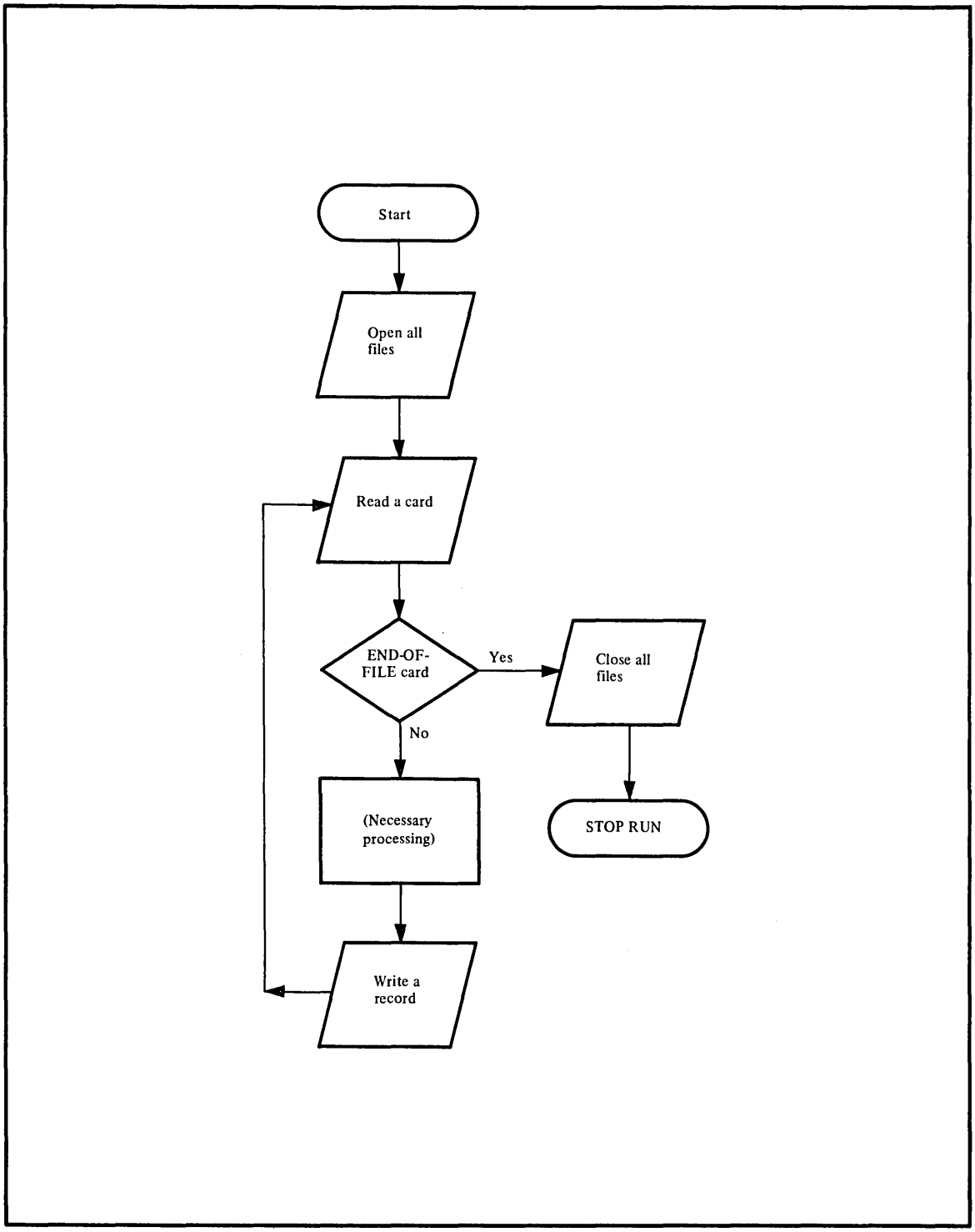

Start
Open all files
Read a card
END-OF-FILE card
Yes
Close all files
No
(Necessary processing)
STOP RUN
Write a record

Figure 35

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PROC1.
           OPEN INPUT STUDENT
                OUTPUT LISTING.
       PARA1.
           READ STUDENT.
           MOVE STUDENT-NAME TO LIST-NAME.
           WRITE LISTING-RECORD.
           GO TO PARA1.
           CLOSE STUDENT LISTING.
           STOP RUN.
```

Rewrite the READ statement in the coding above to specify the action described in the flow chart. The AT END option is used to specify the decision and subsequent action. (Do not use the coding form on which you are coding the program for the problem statement in Figure 29.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ STUDENT AT END
               CLOSE STUDENT LISTING
               STOP RUN.
```

Alternate coding:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ STUDENT AT END CLOSE STUDENT
               LISTING STOP RUN.
```

(If you had coded a period after LISTING, the STOP RUN statement would be executed after any execution of the READ statement.)

---

7. There may be more than one READ statement in a program for a single file. An AT END option must be specified in each READ statement for every input file. Select the statement(s) below that specify the correct use of the AT END option.

   a. Every READ statement must include the AT END option

   b. If a program includes only one READ statement, that statement must contain an AT END option.

   c. If a program contains more than one READ statement for a single file, only one of the statements may include the AT END option.

* * *

b
(An AT END option must be specified in each READ statement unless another option that you will learn to use later is specified.)

8.

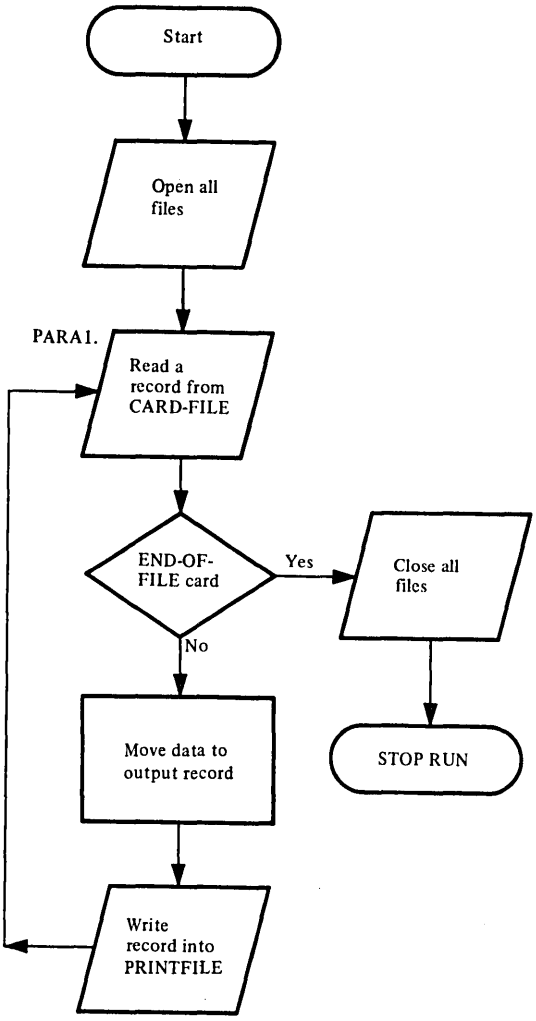


Figure 36

Continue on the coding form on which you have been coding the problem statement in Figure 29. Using the flow chart above, complete the Procedure Division for the program. (Remember that you must read a file and write a record.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PARA1.
           READ CARD-FILE AT END
               CLOSE CARD-FILE PRINTFILE
               STOP RUN.
           MOVE EMPLOYEE-NUMBER
               TO EMPLOYEE-NUMBER-O.
           MOVE HOURS TO HOURS-WORKED.
           WRITE PRINT-RECORD.
           GC TO PARA1.
```

---

9. Often, for easier reading, a programmer will place the statements necessary to complete his program and terminate execution at the end of the program instead of in the AT END option. In order to do this, the programmer must:

   a. include a GO TO statement in the AT END option.

   b. precede the completion statements by a paragraph name.

* * *

Both

---

10. Rewrite the READ statement through the GO TO statement for the problem statement in Figure 29 to include the completion statements at the end of the program. Use the paragraph name FINISH.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PARA1.
           READ CARD-FILE
               AT END GO TO FINISH.
           MOVE EMPLOYEE-NUMBER TO
               EMPLOYEE-NUMBER-0.
           MOVE HOURS TO HOURS-WORKED.
           WRITE PRINT-RECORD.
           GO TO PARA1.
       FINISH.
           CLOSE CARD-FILE PRINTFILE.
           STOP RUN.
```

(This coding will produce the same effect as the coding in the AT END option.)

11.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  RECORD1.
           02 PURCHASE PIC X(4).
           02 ITEM-NUMBER PIC X(6).
```

You know that level 01 record variables can be broken into level 02 elementary variables. In the record description entry above:

a. RECORD1 is the name of a file.

b. a reference to RECORD1 is a reference to PURCHASE and ITEM-NUMBER.

* * *

b

12.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  STUDENT-RECORD.
           02 MONTH PIC XX.
           02 DAY PIC XX.
           02 YEAR PIC XX.
           02 STUDENT-NUMBER PIC X(8).
           02 STREET PIC X(15).
           02 CITY PIC X(10).
           02 STATE PIC X(4).
```

According to the record description entry above:

a. any of the seven level 02 elementary variables can be referred to individually.

b. all seven level 02 elementary variables are grouped together to form STUDENT-RECORD.

* * *

Both

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  SALES.
           02 AVERAGE PIC 9999.
           02 HIGH PIC 9999.
           02 LOW PIC 9999.
```

(The 9's shown in Level 02 pictures signify numeric characters).

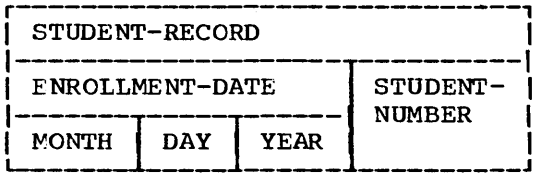
Match each description with the appropriate variable defined in the record description entry above.

1) SALES	a.	Elementary variable
2) AVERAGE	b.	Group variable
	c.	Variable that is further subdivided

* * *

1) b,c
2) a

14.



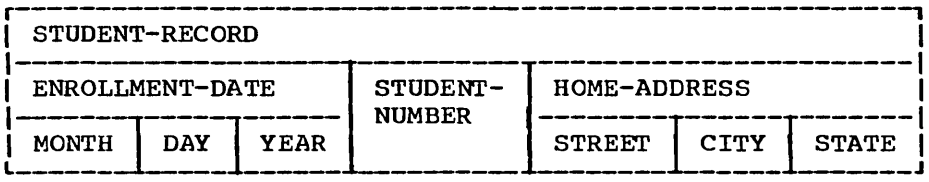

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  STUDENT-RECORD.
           02  ENROLLMENT-DATE.
               03  MONTH
               03  DAY
               03  YEAR
           02  STUDENT-NUMBER
```

If you were using a record like the one described in a preceding example, it is likely that you would want to refer to MONTH, DAY and YEAR at the same time, perhaps as DATE, or ENROLLMENT-DATE. The same is true for STREET, CITY and STATE, which could be referred to as ADDRESS.

COBOL enables you to group single data items within a record. In the example above, MONTH, DAY, and YEAR have been grouped together into the group variable ENROLLMENT-DATE. The block above shows the logical structuring of the record. ENROLLMENT-DATE is a level 02 group item. MONTH, DAY and YEAR are level 03 elementary variables and can be referred to separately, or collectively as ENROLLMENT-DATE, or together with STUDENT-NUMBER as STUDENT-RECORD.

Select the correct statement(s) for the following example.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  STUDENT-RECORD.
           02  ENROLLMENT-DATE.
               03  MONTH
               03  DAY
               03  YEAR
           02  STUDENT-NUMBER
           02  HOME-ADDRESS
               03  STREET
               03  CITY
               03  STATE
```



a. STREET, CITY and STATE are level 03 elementary variables.

b. HOME-ADDRESS is a level 02 elementary variable.

c. STREET, CITY and STATE can be referenced individually, collectively by HOME-ADDRESS, or together with STUDENT-NUMBER and ENROLLMENT-DATE by STUDENT-RECORD.

* * *

a,c
(HOME-ADDRESS is a level 02 group variable.)

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
01  DEPT-RECORD.
    02  DEPT-NUMBER
    02  SALES.
        03  PERIOD1
        03  PERIOD2
        03  PERIOD3
    02  LISTSALES
    02  COMMISSION.
        03  PERIOD-1
        03  PERIOD-2
        03  PERIOD-3
```

The level number of a data description entry never determines whether the variable is a group or elementary variable. If an item is broken into smaller items, it is a group variable. If an item in a record description entry is not broken into smaller items, it is an elementary variable. Referring to the record description entry in the segment of coding shown above, list the names of the:

1) group variables

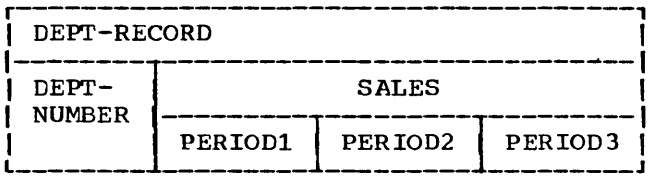
2) elementary variables

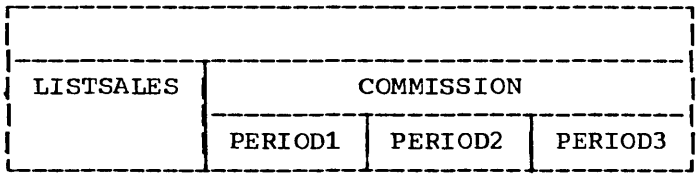
* * *

1) DEPT-RECORD (the record itself), SALES, COMMISSION

2) DEPT-NUMBER, PERIOD1, PERIOD2, PERIOD3, LISTSALES, PERIOD-1, PERIOD-2, PERIOD-3

16.





Only elementary variables have values. Group variables merely allow the programmer to refer to a group of values. This implies that the PICTURE clause will be included in the data description entry for:

a. group variables.

b. elementary variables.

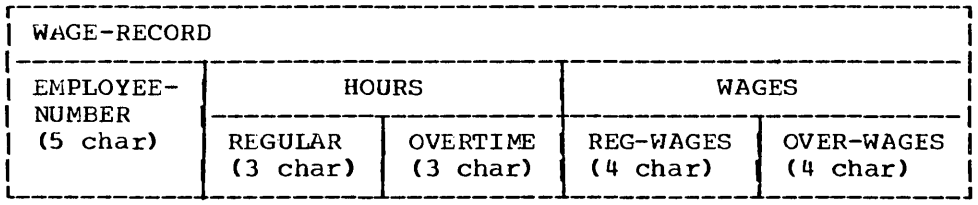
c. SALES in the record above.

d. LISTSALES in the record above.

* * *

b,d

17.



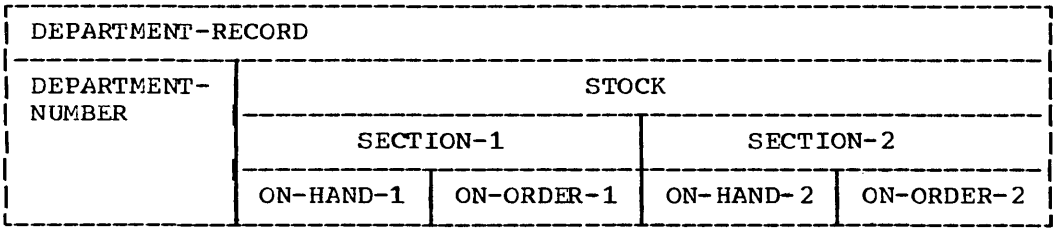
Write a record description entry for the record variable illustrated above. The record variable WAGE-RECORD is to be associated with an input card file.

* * *

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
01  WAGE-RECORD.
    02 EMPLOYEE-NUMBER PIC X(5).
    02 HOURS.
       03 REGULAR PIC X(3).
       03 OVERTIME PIC X(3).
    02 WAGES.
       03 REG-WAGES PIC X(4).
       03 OVER-WAGES PIC X(4).
```

18.



A record variable may contain as many as 49 levels. The diagram above illustrates a record which contains ..... levels.

* * *

four

---

19. The data entries for level 04 variables are written immediately following the level 03 variable of which they are a part. Write the record description entry for the record illustrated in the previous frame. DEPARTMENT-NUMBER contains two characters and each of the remaining elementary variables contains six characters.

```
                         *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  DEPARTMENT-RECORD.
           02 DEPARTMENT-NUMBER PIC XX.
           02 STOCK.
              03 SECTION-1.
                 04 ON-HAND-1 PIC X(6).
                 04 ON-ORDER-1 PIC X(6).
              03 SECTION-2.
                 04 ON-HAND-2 PIC X(6).
                 04 ON-ORDER-2 PIC X(6).
```

(Record variables may be subdivided to level 49. Level number 01 must begin in Area A. Level numbers 02 through 49 may begin in either Area A or Area B. All associated data names and their descriptions must be contained in Area B. In this text, level numbers will be indented for readability as shown above.)

20.

CUSTOMER-RECORD					
CUSTOMER-NUMBER (5 char)	PAYMENTS		BALANCE		
	HI (5 char)	LOW (5 char)	PRESENT (7 char)	HI (7 char)	LOW (7 char)

Write the record description entry for the record described above.

```
                         *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  CUSTOMER-RECORD.
           02 CUSTOMER-NUMBER PIC X(5).
           02 PAYMENTS.
              03 HI PIC X(5).
              03 LOW PIC X(5).
           02 BALANCE.
              03 PRESENT PIC X(7).
              03 HI PIC X(7).
              03 LOW PIC X(7).
```

---

21. The record variable described in the coding for the previous frame contains two data items called LOW and two called HI. Consequently, a reference to HI or LOW in a program would be ambiguous; the computer would have no way of knowing which variable was being referred to. A unique quality about each HI, however, is that:

    a. one is an elementary variable within the group variable PAYMENTS and the other is an elementary variable within BALANCE.

    b. they appear in different records.

* * *

a

---

22. In order to reference the variable HI in a program using the file associated with the record variable CUSTOMER-RECORD, HI must be made unique to avoid ambiguity. HI can be made unique by giving the group item of which it is a part: HI OF PAYMENTS and HI OF BALANCE. Show how the two variables called LOW could be made unique.

* * *

LOW OF PAYMENTS and LOW OF BALANCE
(It is also correct to use IN in place of OF.)

23.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  MASTER-FILE
           LABEL RECORDS ARE OMITTED.
       01  EMPLOYEE.
           02 NAME.
              03 SUR
              03 GIVEN


0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  EMPLOYEE-FILE
           LABEL RECORDS ARE OMITTED.
       01  EMPLOYEE-RECORD.
           02 NAME.
              03 SUR
              03 GIVEN
```

Forming a unique name as shown in the previous frame is called qualification. If a name is not in itself unique, it must be qualified by as many group items in which it is contained as are necessary to make it unique. If the two files described above were used in a program, SUR would have to be qualified each time a reference was made to it. SUR, qualified to the fullest extent, would be

SUR OF NAME OF EMPLOYEE OF MASTER-FILE

and

SUR OF NAME OF EMPLOYEE-RECORD OF EMPLOYEE-FILE.

Because a name must be qualified only to the extent necessary to make it unique, a correct reference to SUR could be:

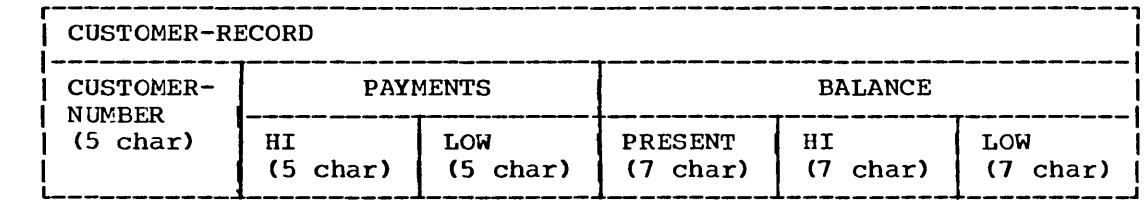
a. SUR OF NAME

b. SUR OF NAME OF EMPLOYEE

* * *

b

24.




```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
FD  LISTFILE
    LABEL RECORDS ARE OMITTED.
01  STOCK.
    02  DEPT.
        03  PORTION PIC X(4).
        03  MARK PIC X(4).
    02  INVOICE NUMBER.
```

The name following the reserved word OF in a qualified name is called a qualifier. The qualifiers must appear in hierarchical order. For example, HI in the record above could not be qualified as HI OF CUSTOMER-RECORD because OF BALANCE or OF PAYMENTS must be included. Nor could it be qualified as HI OF CUSTOMER-RECORD OF PAYMENTS because the qualifiers are out of order. A correct qualified name for MARK in the above coding would be:

a. MARK OF STOCK

b. MARK OF PORTION OF DEPT

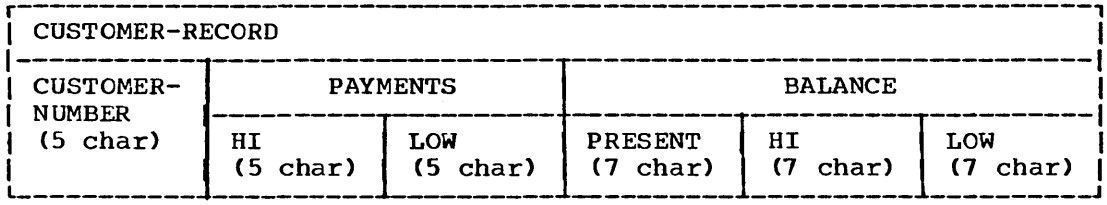
c. MARK OF LISTFILE OF DEPT

d. MARK OF DEPT OF LISTFILE

* * *

None of these
(Proper qualified names for MARK would be:
MARK OF DEPT,
MARK OF DEPT OF STOCK, or
MARK OF DEPT OF STOCK OF LISTFILE.)

25.



The record variable CUSTOMER-RECORD is associated with the file CUSTOMER-FILE. Show how the two variables LOW in the record illustrated above could be qualified.

1) to the fullest extent

2) only to the extent necessary to make them unique

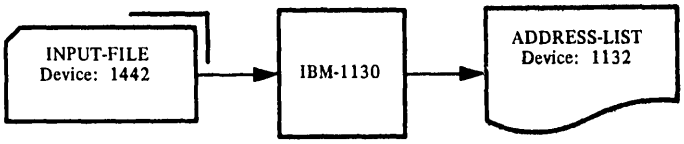
* * *

1) LOW OF PAYMENTS OF CUSTOMER-RECORD
   OF CUSTOMER-FILE
   LOW OF BALANCE OF CUSTOMER-RECORD
   OF CUSTOMER-FILE

2) LOW OF PAYMENTS
   LOW OF BALANCE

---

26. Data-processing problems often require that data in the input file be listed on the printer. Listing the data in the input file may be a portion of a complex program or may be an entire program in itself. The problem described in Figure 37 consists of listing the records in the input file. Read the problem statement and code the program ADDRESS-LISTING to produce the listing.



The system flow chart above shows the files and equipment to be used in this program to produce a listing of the data in the card file INPUT-FILE. The forms of records in INPUT-FILE and ADDRESS-LIST are illustrated below.

INPUT-RECORD			
NAME		HOME-ADDRESS	
SUR	GIVEN		(blank)
(12 characters)	(8 characters)	(40 characters)	(20 characters)

OUTPUT-RECORD	
LAST-NAME	ADDRESS-O
(12 characters)	(40 characters)

The program flow chart shows the order of operations for the Procedure Division.

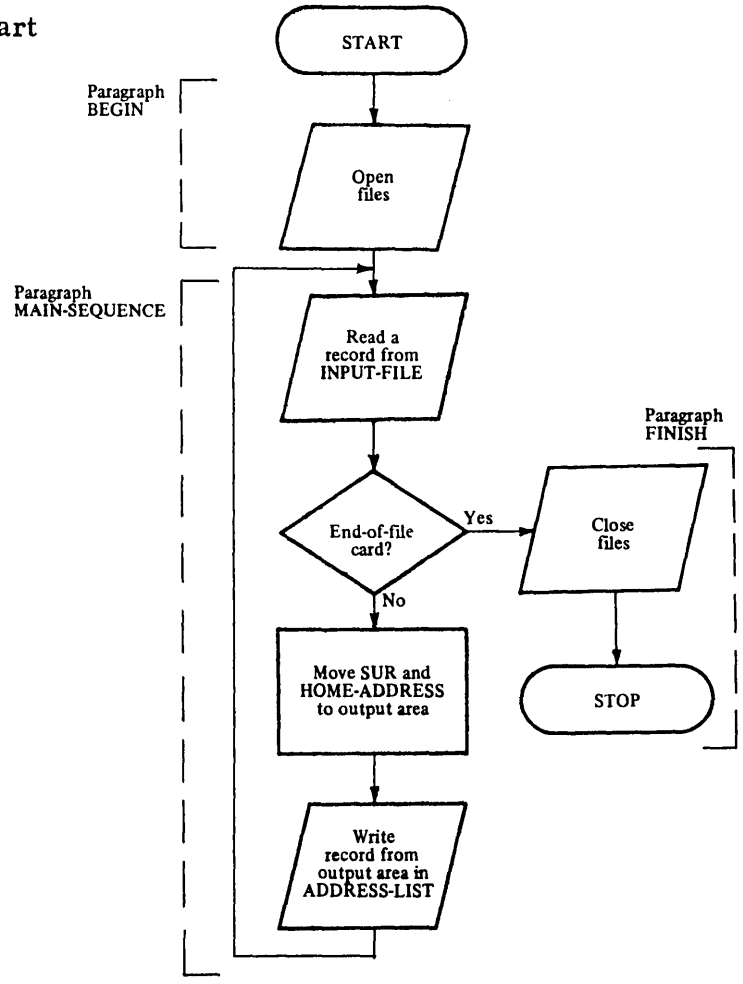


Figure 37

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. ADDRESS-LISTING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE ASSIGN TO
               RD-1442.
           SELECT ADDRESS-LIST ASSIGN TO
               PR-1132.
       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE
           LABEL RECORDS ARE OMITTED.
       01  INPUT-RECORD.
           02  NAME.
               03  SUR PIC X(12).                          (10,51)
               03  GIVEN PIC X(8).
           02  HOME-ADDRESS PIC X(40).
           02  FILLER PIC X(20).                             (13)
       FD  ADDRESS-LIST
           LABEL RECORDS ARE OMITTED.
       01  OUTPUT-RECORD.
           02  LAST NAME PIC X(12).
           02  ADDRESS-O PIC X(40).
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT INPUT-FILE
               OUTPUT ADDRESS-LIST.
       MAIN-SEQUENCE.
           READ INPUT-FILE                                  (26)
               AT END GO TO FINISH.
           MOVE SUR TO LAST-NAME.
           MOVE HOME-ADDRESS TO ADDRESS-O.
           WRITE OUTPUT-RECORD.
           GO TO MAIN-SEQUENCE.                             (36)
       FINISH.
           CLOSE INPUT-FILE ADDRESS-LIST.
           STOP RUN.
```

SUMMARY:

You have now completed Lesson 7 in which you learned COBOL statements and entries necessary to read data records from an input card file. The function of each of these entries is described on the following page.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. ADDRESS-LISTING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM 1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
(1)        SELECT INPUT-FILE ASSIGN TO
               RD-1442.
           SELECT ADDRESS-LIST ASSIGN TO
               PR-1132.
       DATA DIVISION.
       FILE SECTION.
(2)    FD  INPUT-FILE
           LABEL RECORDS ARE OMITTED.
       01  INPUT-RECORD.
           02  NAME.
(3)            03  SUR PIC X(12).
               03  GIVEN PIC X(8).
           02  HOME-ADDRESS PIC X(40).
           02  FILLER PIC X(20).
       FD  ADDRESS-LIST
           LABEL RECORDS ARE OMITTED.
       01  OUTPUT-RECORD.
           02  LAST-NAME PIC X(12).
           02  ADDRESS-O PIC X(40).
       PROCEDURE DIVISION.
       BEGIN.
(4)        OPEN INPUT INPUT-FILE
               OUTPUT ADDRESS-LIST.
       MAIN-SEQUENCE.
(5)        READ INPUT-FILE
               AT END GO TO FINISH.
           MOVE SUR TO LAST-NAME.
           MOVE HOME-ADDRESS TO ADDRESS-O.
           WRITE OUTPUT-RECORD.
           GO TO MAIN-SEQUENCE.
       FINISH.
(6)        CLOSE INPUT FILE ADDRESS-LIST.
           STOP RUN.
```

(1) Links file name to equipment to be used for the file.

(2) Specifies whether file contains records used to label the file.

(3) Defines an input area and associates it with a file name.

(4) Prepares file for processing and specifies its use as input.

(5) Reads a record from the file into the input area associated with the file in the File Section.

(6) Terminates processing of the file.

In this lesson you also learned to group elementary variables into level 02 and level 03 group variables and to qualify names when necessary. You learned to specify looping or a transfer of control to another point in the program. These programming techniques will be expanded in following lessons.

END OF LESSON 7

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 8

LESSON 8 - HORIZONTAL SPACING

INTRODUCTION

As a programmer you will be required to prepare printed reports with specified page titles and headings. You will also be required to follow a specified format for titles, headings, and data items. In this lesson you will learn to control horizontal spacing of titles, headings, and data items in records in a printer file.

Specific COBOL language features that you will learn to use are:

- FILLER items to insert blanks preceding, between, and following data items in a record in an output file
- VALUE clause
- ZEROS figurative constant
- Maximum size of output area for printer file

This lesson will require approximately one hour.

1. You learned to specify a record variable of 80 characters, the same size as a card record in an input file. If a record in an output file is to be a line of 120 or less characters on an 1132 printer, you would expect to specify an output area:

   a. the same size as the record in the output file.

   b. of 120 characters or less.

* * *

Both (Carriage control: if the maximum of 124 printer positions is to be used, you must actually specify an output area of 125 characters and provide for the first position to be unused. A carriage control character for the printer will be generated by the compiler in the first position of the output area. If no carriage control is specified in the ASSIGN clause, one must specify only a 120 character output area. No carriage control character is required.)

---

2. You used a FILLER item in a record variable for a portion of a card record that was blank, or that contained no data to be transmitted. If a record in an output area is to be transmitted to a printer line, you would expect to use a FILLER item in the output area for a portion of the printer line:

   a. that is to be blank.

   b. to which no data is to be transmitted.

* * *

Both

---

3. You know that the value of a variable is undefined until it is given a value in the program. If no specific value has been given, the variable:

   a. may contain a value remaining from previous processing.

   b. will contain blanks.

* * *

a

---

4. Thus far you have learned that a value may be given to a variable by:

   a. moving a value to the variable from another variable with a MOVE statement.

   b. transmitting a value to the variable from an input file with a READ statement.

* * *

Both

---

5. If a value has not been given to a FILLER item in an output area before a record is transmitted from the output area to a printer line, the value that will be printed from the FILLER item is:

   a. made up of blanks.

   b. whatever value remains from previous processing.

* * *

b

---

6. A FILLER item in an output area:

   a. is used for a portion of a printer line that is to be blank.

   b. should be given an initial value of blanks.

* * *

Both

This figure illustrates the execution of the statement

MOVE INPUT-RECORD TO OUTPUT-RECORD

used to move a value of a shorter variable to a longer variable.

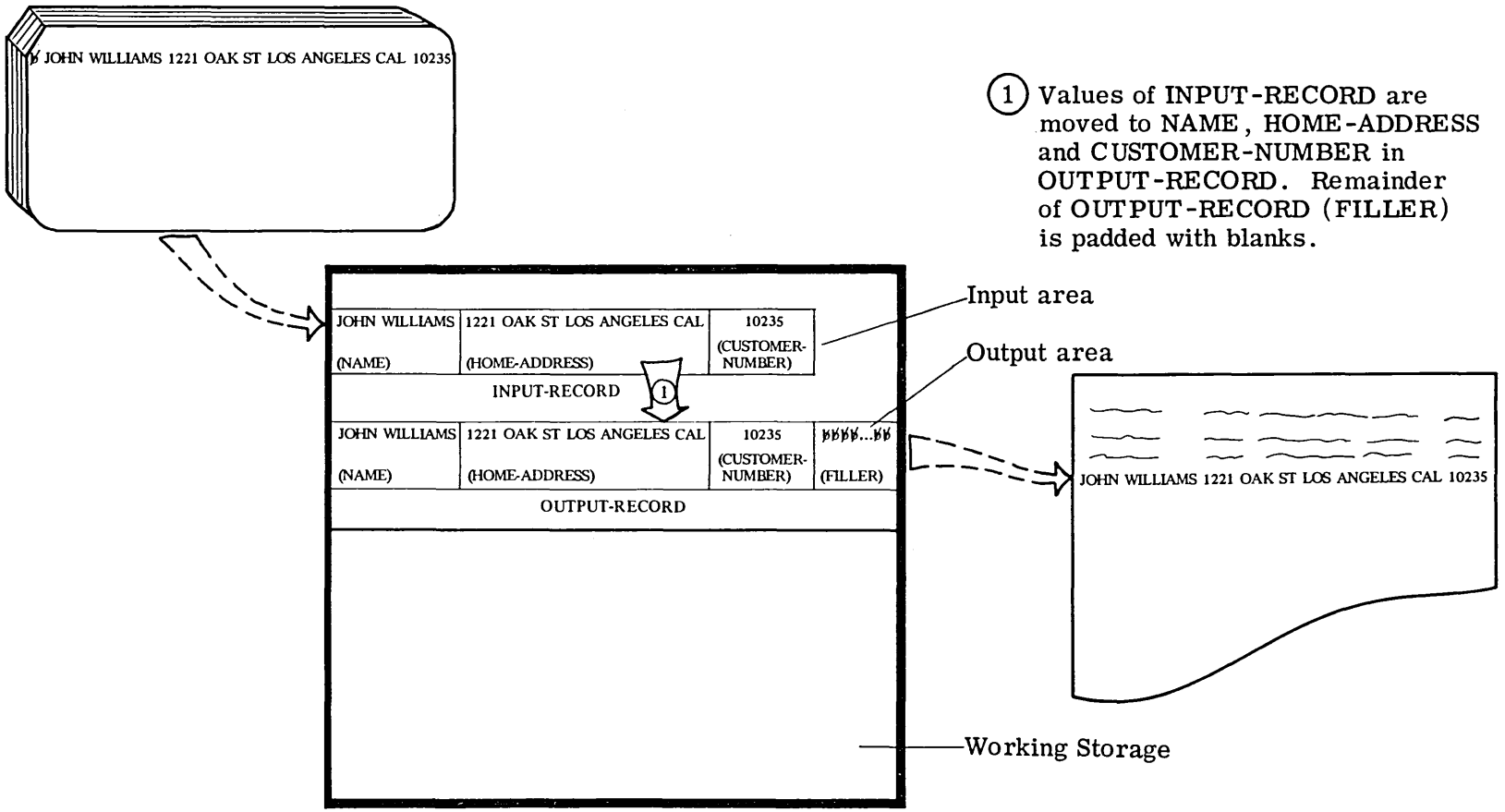


Figure 38

7. Figure 38 illustrates the execution of a MOVE statement. Read the explanation and look at the illustration in Figure 38. Arrow 1 indicates that the statement is used to move values from:

   a. working storage to an output area.

   b. an input area to an output area.

* * *

b

---

8. According to the explanation for Arrow 1 in Figure 38, the MOVE statement moves:

   a. values of INPUT-RECORD to NAME, HOME-ADDRESS, and CUSTOMER-NUMBER in OUTPUT-RECORD.

   b. a value of INPUT-RECORD to FILLER in OUTPUT-RECORD.

* * *

a

---

9. According to the explanation for Arrow 1 in Figure 38, the MOVE statement:

   a. moves a value to FILLER in OUTPUT-RECORD from INPUT-RECORD.

   b. causes FILLER in OUTPUT-RECORD to be padded with blanks.

* * *

b

---

10. Figure 38 shows that when values are moved from a record variable to a larger output area, FILLER in the output area:

    a. is unaffected.

    b. contains whatever value remains from previous processing.

* * *

Neither (is padded with blanks. When values of a record variable are moved to a shorter output area, the record is truncated on the right to the length of the output area.)

---

11. In Figure 38 the FILLER item is used in OUTPUT-RECORD to fill the output area to:

    a. the size of a record in the output file.

    b. the length of a line on a printer.

* * *

Both

12.

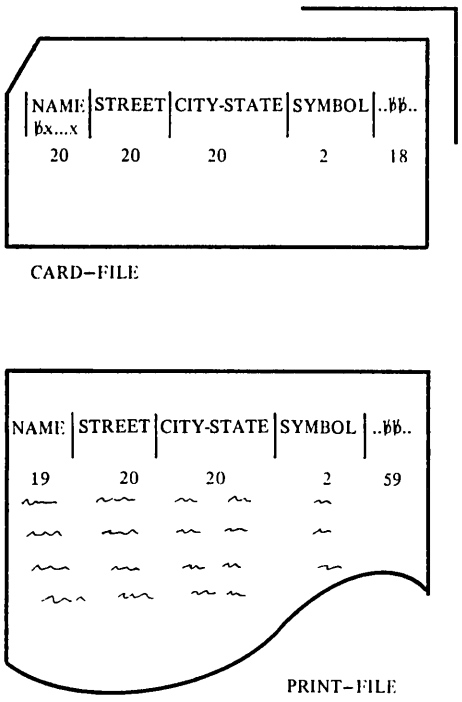


Figure 39

Preparation of a listing of records in a card (or disk) file is a common operation in any data-processing center. The following problem will give you an opportunity to practice coding the entries you have learned as you write the Data and Procedure Divisions for a program to prepare a listing.

The customer records for a department store are card records of the type shown above. Each record in CARD-FILE is to be printed on one line of PRINT-FILE using an 1132 printer (which has 120 columns per line.) Although the first column in each card record is blank, this is not a common practice. It has been done for this problem to provide for the carriage control character that will be generated as the first character in the output area. While 19 positions are shown for NAME in PRINT-FILE, you must specify 20 positions for NAME in the output area for PRINT-FILE. (The output area is to be 121 positions.) The first position in the output area will contain a printer carriage control character; only the characters in positions 2 through 20 will be printed, in regard to NAME.

Using CUSTOMER-RECORD and PRINT-RECORD as the input and output areas, respectively, write the:

1) Data Division entries that would be required in this problem including the necessary headers, the FD entries, and the record description entries.

2) Procedure Division entries including the:

   a. division header.

   b. paragraph BEGIN containing a statement to prepare the files for processing.

   c. paragraph MAINSEQUENCE containing statements to transfer a record from the card file to the printer file, an instruction to transfer control to MAINSEQUENCE to process the next record and to print a record, and an instruction to transfer control to the paragraph FINISH when all records in CARD-FILE have been processed.

   d. paragraph FINISH containing statements to close files and stop execution.

* * *

1)

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CUSTOMER-RECORD.
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY-STATE PIC X(20).
           02  SYMBOL PIC X(2).
           02  FILLER PIC X(18).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD.
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY-STATE PIC X(20).
           02  SYMBOL PIC X(2).
           02  FILLER PIC X(59).
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT CARD-FILE
               OUTPUT PRINT-FILE.
       MAINSEQUENCE.
           READ CARD-FILE AT END GO TO FINISH.
           MOVE CUSTOMER-RECORD
               TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           GO TO MAINSEQUENCE.
       FINISH.
           CLOSE CARD-FILE PRINT-FILE.
           STOP RUN.
```

---

13. Figure 38 shows that when a record has been transmitted from the output area to the output file, the values:

    a. of the variables NAME, HOME-ADDRESS, and CUSTOMER-NUMBER will be printed in a continuous string.

    b. of individual variables would be easier to distinguish if they were separated by blanks.

* * *

Both

This figure illustrates the execution of the statements

MOVE NAME OF INPUT-RECORD TO NAME OF WORKING-RECORD.
MOVE HOME-ADDRESS OF INPUT-RECORD TO HOME-ADDRESS OF WORKING-RECORD.
MOVE CUSTOMER-NUMBER OF INPUT-RECORD TO CUSTOMER-NUMBER OF WORKING-RECORD.
MOVE WORKING-RECORD TO OUTPUT-RECORD.

used to insert blanks in a record for horizontal spacing on a printer line.

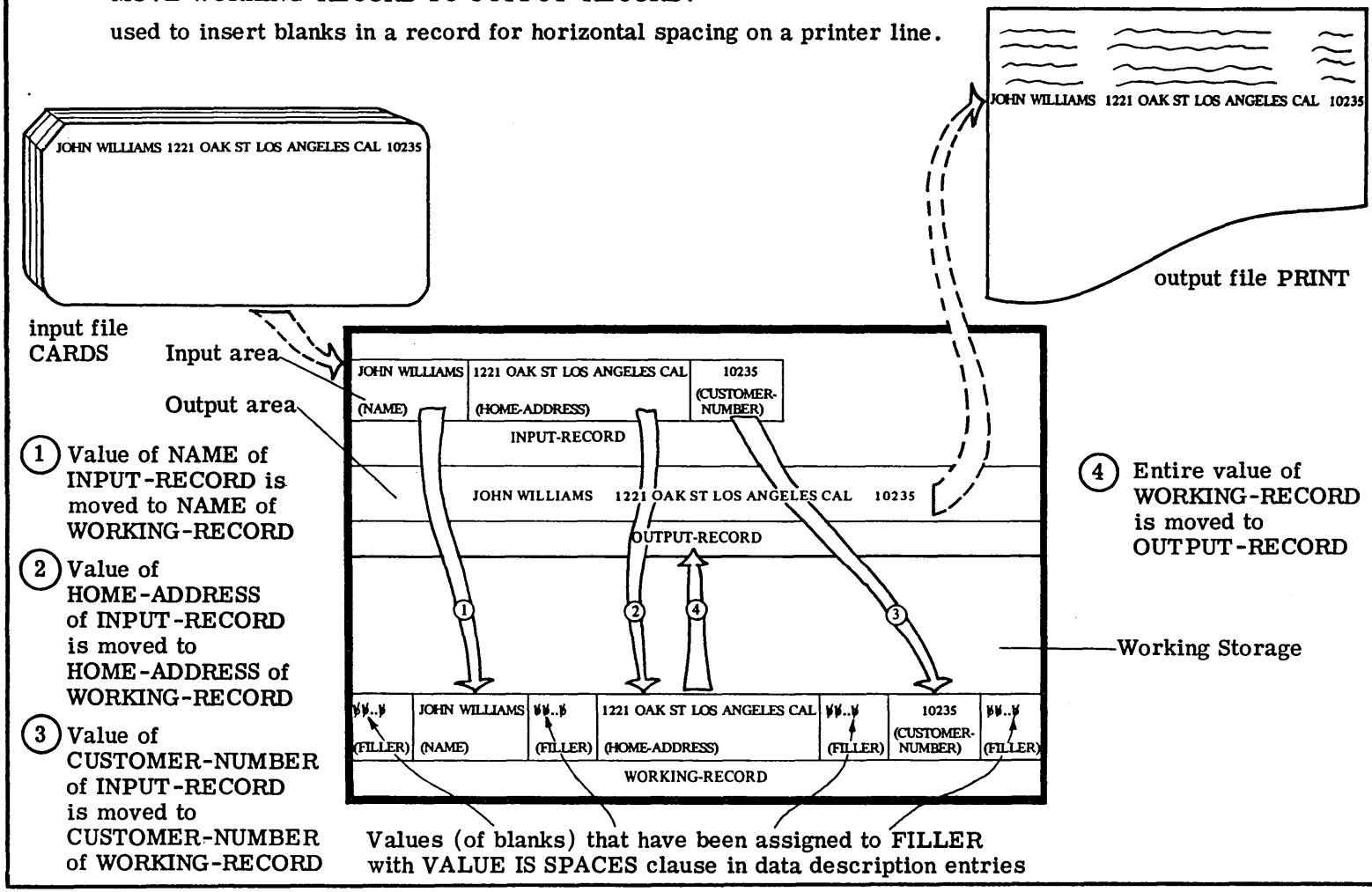


Figure 40

14. A programmer must provide for the first character in an output area to be reserved for the printer carriage control character. In addition, a programmer may wish to insert blanks between individual data items on a printer line to make them easier to distinguish. Figure 40 illustrates execution of statements that can be used to reserve the first character in the output area and to insert blanks in a record to provide horizontal spacing on a printer line. Read the explanation and look at the illustration in Figure 40. The arrows in Figure 40 are numbered to correspond to the MOVE statements. According to the explanation for Arrow 1, the first statement will move:

    a. the values of NAME, HOME-ADDRESS, and CUSTOMER-NUMBER in INPUT-RECORD to NAME, HOME-ADDRESS, and CUSTOMER-NUMBER in WORKING-RECORD.

    b. the value of NAME in INPUT-RECORD to NAME in WORKING-RECORD.

* * *

b

---

15. In Figure 40 values of elementary variables in INPUT-RECORD are moved to:

    a. elementary variables separated by FILLER items in a single record variable in working storage.

    b. separate output areas.

* * *

a

---

16. Figure 40 shows that the programmer has provided for horizontal spacing in a record in the output file. He has included FILLER items in the record variable in working storage to insert blanks:

    a. preceding the first data item.

    b. between data items.

    c. following the last data item.

* * *

All of these

---

17. Figure 40 shows that the programmer has assigned values of blanks to the FILLER items in WORKING-RECORD. This figure also shows that, as a result, spaces will precede, separate, and follow data items:

    a. in the output area.

    b. on a printer line in the output file PRINT.

* * *

Both

---

18. Figure 40 shows that a FILLER item used to insert blanks for horizontal spacing of data items on a printer line:

    a. will be padded with blanks as a result of a MOVE statement.

    b. have been assigned a value of blanks with a VALUE IS SPACES clause.

                    *   *   *

b (If a single statement were used to move the value of INPUT-RECORD to WORKING-RECORD, the value of INPUT-RECORD would be padded with blanks to the length of WORKING-RECORD.)

---

19. The MOVE statements in figure 40:

    a. move values from an input area to FILLER items in a working-storage variable.

    b. cause FILLER items in a working-storage variable to be padded with blanks.

    c. would leave the value of FILLER items in a working-storage variable undefined if a value of blanks had not been assigned to the items.

                    *   *   *

c

This figure shows the Data Division entries for inserting spaces between data items for an output file.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

01      DATA DIVISION.
02      FILE SECTION.
03      FD  CARDS
04          LABEL RECORDS ARE OMITTED.
05      01  INPUT-RECORD.
06          02 NAME PIC X(35).
07          02 HOME-ADDRESS PIC X(40).
08          02 CUSTOMER-NUMBER PIC X(5).
09      FD  PRINT
10          LABEL RECORDS ARE OMITTED.
11      01  OUTPUT-RECORD PIC X(121).
12      WORKING-STORAGE SECTION.
13      01  WORKING-RECORD.
14          02 FILLER PIC X(10) VALUE IS SPACES.
15          02 NAME PIC X(35).
16          02 FILLER PIC X(10) VALUE IS SPACES.
17          02 HOME-ADDRESS PIC X(40).
18          02 FILLER PIC X(10) VALUE IS SPACES.
19          02 CUSTOMER-NUMBER PIC X(5).
20          02 FILLER PIC X(11) VALUE IS SPACES.
```

Figure 41

20. Figure 41 shows the Data Division entries for the variables in Figure 40. The clause VALUE IS SPACES specifies that the FILLER item preceding NAME will be assigned an initial value of 10 blanks. The same clause in the data description entry on:

    a. line 18 specifies that the FILLER item preceding CUSTOMER-NUMBER will be assigned an initial value of 10 blanks.

    b. line 20 specifies that the FILLER item following CUSTOMER-NUMBER will be assigned an initial value of 11 blanks. Recall that the print output area must be specified or 121 positions to accomodate a character (one more than the number of print positions).

* * *

Both

---

21. In Figure 40 the FILLER item to the left of NAME in WORKING-RECORD is used to insert blanks:

    a. at the beginning of a printing line.

    b. between data items on a printing line.

    c. at the end of a printing line.

* * *

a (Only nine blanks would be inserted at the beginning of a printing line; the first position in the output area would be used for a printer carriage control character.)

---

22. The VALUE clause may be specified only in the Working-Storage Section. A programmer may use a VALUE clause in a data description entry to assign a value of blanks to a FILLER item in:

    a. an output area.

    b. a working-storage variable.

* * *

b

---

23. A programmer may specify a VALUE clause in a data description entry for a FILLER item in the:

    a. Working-Storage Section.

    b. File Section.

* * *

a

---

24. When the data items in an input area are to be printed with horizontal spacing, blanks can be inserted by:

    a. moving the data items to elementary variables in a working-storage record variable containing FILLER items with values of blanks assigned in a VALUE clause and then moving the value of the entire record variable to an output area.

    b. moving the data items to elementary variables in an output area containing FILLER items with values of blanks assigned in a VALUE clause.

* * *

a

---

25. Figure 40 shows that in moving values from WORKING-RECORD to OUTPUT-RECORD:

    a. OUTPUT-RECORD has been subdivided to receive elementary values from WORKING-RECORD.

    b. the value of WORKING-RECORD is moved as an entire record to OUTPUT-RECORD.

* * *

b

---

26. Figure 40 shows that the record in working storage:

    a. is written on a printer line directly from the working-storage variable.

    b. must be moved to an output area before it is written on a printer.

* * *

b

---

27. In Figure 41 the record description entry for the output area OUTPUT-RECORD:

    a. defines a level 01 variable of 121 characters.

    b. shows that when a record is not subdivided, the PICTURE clause is specified in the level 01 entry.

* * *

Both

---

28. Data items in an input record are to be printed on a line of a printer page on an 1132 printer with blanks preceding, separating, and following the data items. To provide for the blanks to be inserted into the record in the output file, a programmer would include a record description entry in the Data Division for:

    a. a working-storage variable containing FILLER items assigned values of blanks with a VALUE clause.

    b. an input area subdivided into elementary variables.

    c. an output area that need not be subdivided.

* * *

All of these

---

29. To provide for blanks to precede, separate and follow data items in a record in an output file, a programmer must include a statement in the Procedure Division to:

    a. move each data item individually to an elementary variable in a working-storage variable containing FILLER items.

    b. move the values of the entire working-storage record variable to an output area.

* * *

Both

---

30. The following problem incorporates all the features that you have learned to use to provide horizontal spacing in a printed report. Coding the solution is optional. If you do not code the solution, read it carefully to make sure you understand it.

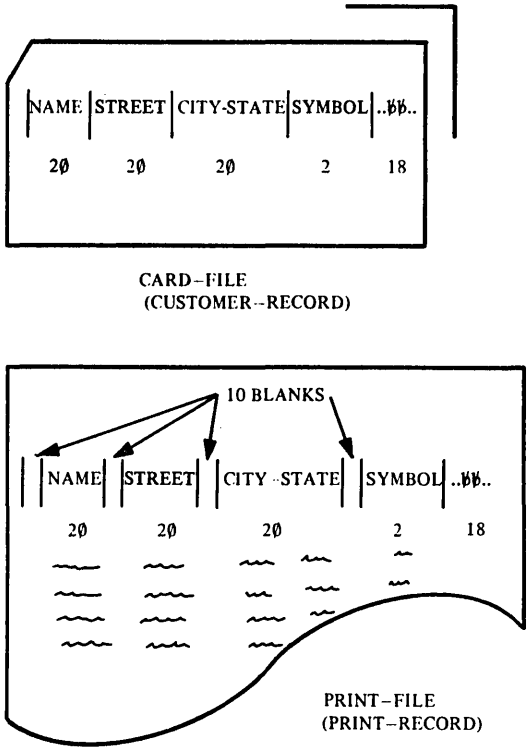


Figure 42

Coding the problem described below is a typical COBOL programming task and the problem is a type that you will probably encounter frequently. Each record in CARD-FILE is to be printed on one line of PRINT-FILE using an 1132 printer. Blanks are to be inserted as shown above. (Remember to provide an extra position at the beginning of your output record for the printer carriage control character.) Using WORKING-RECORD as the working-storage variable, code the Data Division and the Procedure Division for this problem. (Remember that the Working-Storage Section must follow the File Section.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CUSTOMER-RECORD.
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY-STATE PIC X(20).
           02  SYMBOL PIC X(2).
           02  FILLER PIC X(18).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       01  WORKING-RECORD.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  NAME PIC X(20).
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  STREET PIC X(20).
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  CITY-STATE PIC X(20).
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  SYMBOL PIC X(2).
           02  FILLER PIC X(18) VALUE IS SPACES.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT CARD-FILE
               OUTPUT PRINT-FILE.
       MAINSEQUENCE.
           READ CARD-FILE AT END GO TO FINISH.
           MOVE NAME OF CUSTOMER-RECORD
               TO NAME OF WORKING-RECORD.
           MOVE STREET OF CUSTOMER-RECORD
               TO STREET OF WORKING-RECORD.
           MOVE CITY-STATE OF CUSTOMER-RECORD
               TO CITY-STATE OF WORKING-RECORD.
           MOVE SYMBOL OF CUSTOMER-RECORD
               TO SYMBOL OF WORKING-RECORD.
           MOVE WORKING-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           GO TO MAINSEQUENCE.
       FINISH.
           CLOSE CARD-FILE PRINT-FILE.
           STOP RUN.
```

31. The FILLER item at the end of an output area could be omitted. Whenever a record is moved from one variable to a longer variable, the record is padded with blanks on the right to the length of the longer variable. A record in an input file is to be printed on a line of a 1132 printer. The record in the input file, however, is less than 120 characters. To provide for the record to be filled with blanks to 120 characters, a programmer:

    a. can move data items in the record to a working-storage variable containing a FILLER item with a value of blanks assigned in a VALUE clause.

    b. can move the record to an output area containing a FILLER item that will be padded with blanks as a result of the move.

    c. can move the record to a longer output area that will be padded with blanks as a result of the move.

* * *

All of these
(Whenever a record is moved from one variable to a shorter variable, the record in the sending variable will be truncated on the right to the length of the shorter receiving variable. The truncated data is lost. Remember: if carriage control is used, the print output area in core must be one position greater than the number of characters printed.)

---

first printing position

This figure shows how the format of data items is represented on a Printer Spacing Chart.

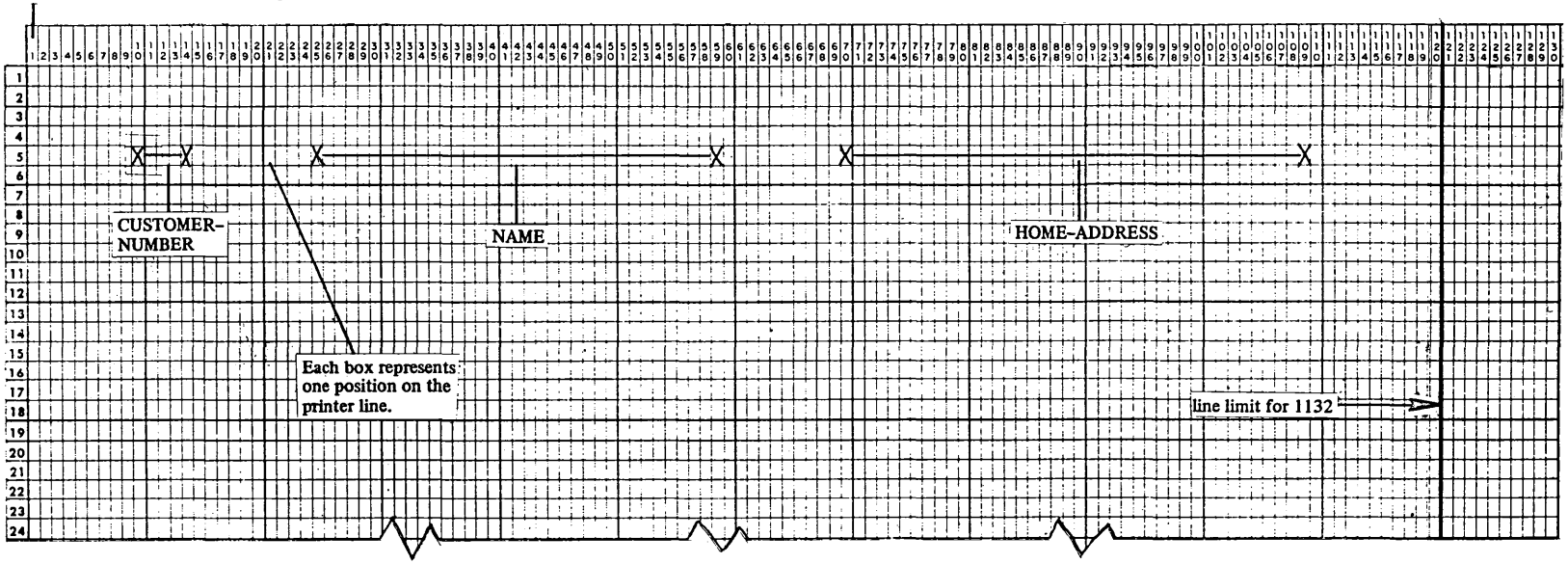


Figure 43

32. Figure 43 shows a Printer Spacing Chart that can be used to plan the format of printed records. According to Figure 43 the first printing position is:

    a. position 0.

    b. position 1.

* * *

b
(Position 0 represents the carriage control character that you must reserve in the output area.)

---

33. Figure 43 shows that for a 1132 printer used in an American National Standard COBOL program:

    a. the line limit follows position 120.

    b. a line consists of 120 columns.

* * *

Both

---

34. As shown in Figure 43, the position of a data item may be indicated on a Printer Spacing Chart by two x's connected by a horizontal line. According to Figure 43, the first ..... positions in the line will be blank. The value of CUSTOMER-NUMBER will be printed in the next ..... positions. Data items will be separated by ..... blanks. The value of HOME-ADDRESS will be followed by ..... blanks.

* * *

9,5,10,11

---

35. If you wished to print values of variables described in Figure 41 in the order shown in Figure 43, it would be necessary to rearrange the data description entries in:

    a. the File Section.

    b. the Working-Storage Section.

* * *

b

---

36. Rewrite the Working-Storage Section in Figure 41 so that values from the input area described in Figure 41 can be printed in the format shown in Figure 43.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WORKING-STORAGE SECTION.
       01  WORKING-RECORD.
           02 FILLER PIC X(10) VALUE IS SPACES.
           02 CUSTOMER-NUMBER PIC X(5).
           02 FILLER PIC X(10) VALUE IS SPACES.
           02 NAME PIC X(35).
           02 FILLER PIC X(10) VALUE IS SPACES.
           02 HOME-ADDRESS PIC X(40).
           02 FILLER PIC X(11) VALUE IS SPACES.
```

SUMMARY:

You have just learned how to prepare the horizontal spacing format of printed output. In the next lesson you will learn other aspects of printed data - vertical printing control, and title and heading printing.

END OF LESSON 8

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 9

LESSON 9 - VERTICAL SPACING (1)

INTRODUCTION

This lesson will continue to prepare you to program written reports. You will learn to write titles and headings and to provide for vertical printing control characters. You will also study picture representation for numeric and alphabetic data.

Specific COBOL language features that you will learn to use are:

Picture characters 9 and A
SPACES figurative constant
Provision for printer carriage control
    character in first position of output area
Title records
Heading records in working storage

If carriage control is specified in the device named in the ASSIGN clause, a blank position must be left in the first position of the print output area, so that a carriage control character may be filled in by the 1130 Monitor Program. This means that the second core position prints in the first printer position. Also, whenever carriage control is used, the PICTURE clause associated with the output area must specify one character more than the number of characters to be printed, to accomodate the carriage control character.

This lesson will require approximately one hour, with an additional half hour requirement if you do the optional problem.

1. The nonnumeric literals consisting of one or more blanks have been given the names SPACES. The reserved word SPACES is called a figurative constant. The numeric literals consisting of one or more zeros have been given the name ZEROS. The reserved word ZEROS is a:

   a. figurative constant.

   b. nonnumeric literal.

                    *   *   *

a

---

2. The figurative constant SPACES was specified in a VALUE clause following a PICTURE clause with an X specification, indicating that SPACES is specified for data made up of:

   a. numeric digits.

   b. alphanumeric characters (one of the characters in Figure 7).

                    *   *   *

b

---

3. You would expect the figurative constant ZEROS to be specified for:

   a. numeric data.

   b. data made up of any one of the characters in Figure 7.

                    *   *   *

a

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
          02 MAXIMUM-BALANCE PIC 9(4).
```

A PICTURE clause with a 9 specification is used to describe numeric data that may be made up of the digits 0 through 9. The PICTURE clause in the entry above specifies that MAXIMUM-BALANCE could have the value:

   a. 10.4

   b. 0104

   c. 100B

                    *   *   *

b

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02 PART-DESCRIPTION PICTURE A(30).
```

A PICTURE clause with an A specification is used to describe alphabetic data that may be made up of the letters of the alphabet and spaces. The PICTURE clause in the entry above specifies that PART-DESCRIPTION could have the value:

a. 5 OHM RESISTOR

b. FIVE-OHM RESISTOR

c. FIVE OHM RESISTOR

* * *

c (a is wrong because numbers cannot be accommodated in an A picture. b is wrong because special characters cannot be accommodated in an A picture.)

Picture and Edit Characters

Picture character	Data type	Specification	Additional explanation
X	alphanumeric	The associated position in the value will contain any character from the COBOL character set.	
A	alphabetic	The associated position in the value will contain an alphabetic character or a space.	
9	numeric or numeric edited	The associated position in the value will contain any digit.	
V	numeric	The decimal point in the value will be assumed to be at the location of the V. The V does not represent a character position.	
.	numeric edited	The associated position in the value will contain a point or a space.	A space will occur if the entire data item is suppressed.
$	numeric edited	a. (simple insertion) The associated position in the value will contain a dollar sign. b. (floating insertion) The associated position in the value will contain a dollar sign, a digit, or a space.	The leftmost $ in a floating string does not represent a digit position. If the string of $ is specified only to the left of a decimal point, the rightmost $ in the picture corresponding to a position that precedes the leading nonzero digit in the value will be printed. A string of $ that extends to the right of a decimal point will have the same effect as a string to the left of the point unless the value is zero; in this case blanks will appear. All positions corresponding to $ positions to the right of the printed $ will contain digits; all to the left will contain blanks.
,	numeric edited	The associated position in the value will contain a comma, space, or dollar sign.	A comma included in a floating string is considered part of the floating string. A space or dollar sign could appear in the position in the value corresponding to the comma.
S	numeric	A sign (+ or –) will be part of the value of the data item. The S does not represent a character position.	

Figure 44

6. Figure 44 is a chart of picture characters. Use Figure 44 to match types of data with the appropriate variable(s).

1) NAME	a. Alphabetic
2) HOME-ADDRESS	b. Numeric
3) QUANTITY, with values such as 200 and 999	c. Data made up of any characters in Figure 7
4) QUANTITY, with values such as 1 DOZ and 25 PKG	

* * *

1) a,c
2) c
3) b,c (Choice c is correct with respect to COBOL but such data is usually treated as numeric.)
4) c

---

7. Use Figure 44 to match the picture specification character with the variable(s) for which it could be specified.

1) NAME	a. X
2) HOME-ADDRESS	b. A
3) QUANTITY with values such as 200 and 999	c. 9
4) QUANTITY with values such as 1 DOZ and 25 PKG	

* * *

1) a,b
2) a
3) a,c
4) a

8.

INPUT-RECORD		
NAME (NOVELTYbDISTRIBUTORS)	ID-CODE (2901)	BALANCE (1299.43)

Write the record description entries for the input area INPUT-RECORD so that it could have the values shown. Remember that the decimal point precludes BALANCE being defined as numeric.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
01  INPUT-RECORD.
    02 NAME PIC A(20).
    02 ID-CODE PIC 9(4).
    02 BALANCE PIC X(7).
```

(The picture character X could be specified for ID-CODE and NAME. Values such as those in BALANCE normally do not contain decimal points in an input record and are described with the picture character 9. The decimal position is indicated by the picture character V, which you will learn to use in a subsequent lesson.)

This figure illustrates how the format of headings and data items for output is planned on a Printer Spacing Chart.

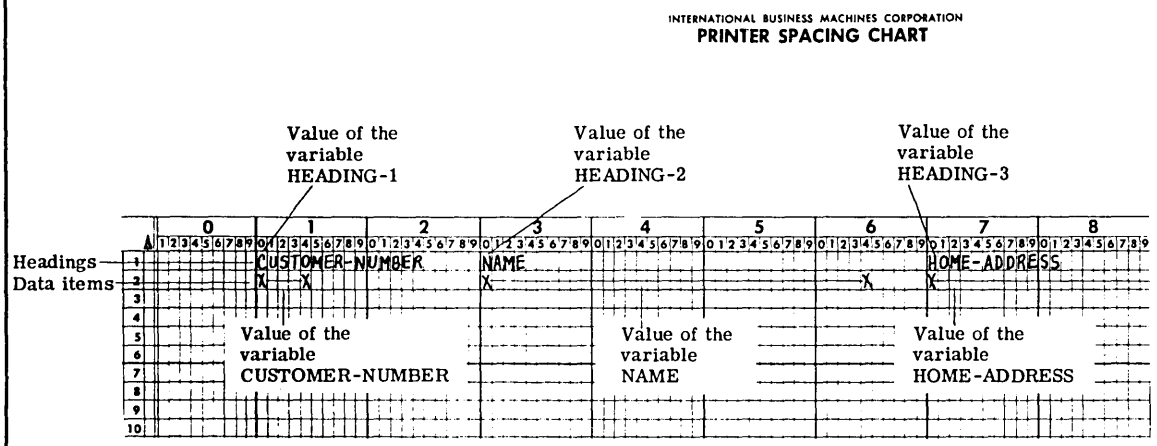
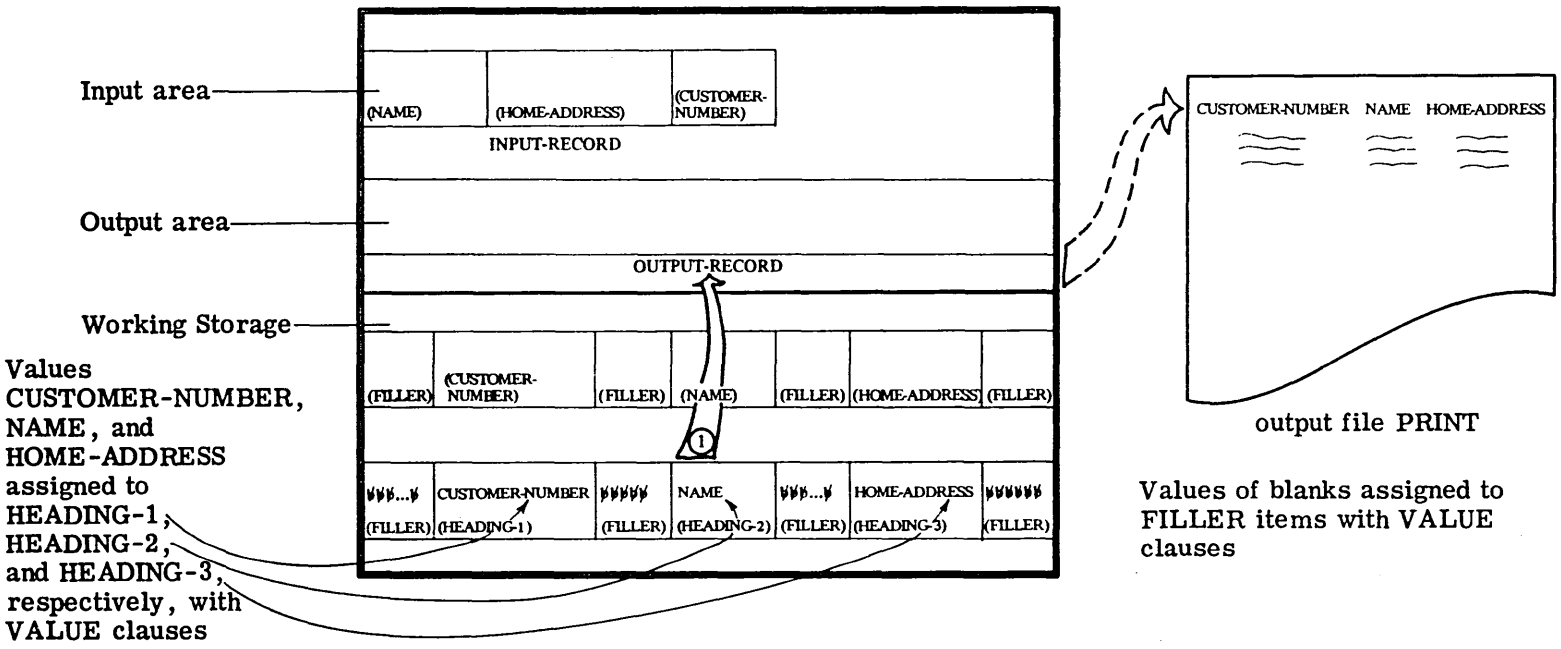


Figure 45

9. When several records are to be printed, a programmer may print headings at the top of each page. Figure 45 shows how the format for headings and data items is planned on a Printer Spacing Chart. The same chart is also used for the 1132 Printer. In Figure 46 the headings CUSTOMER-NUMBER, NAME, and HOME-ADDRESS are themselves values of the variables ..... , ... , and ..... , respectively.

This figure shows how headings are set up in working storage.




```
WORKING-STORAGE SECTION.
Ø1  HEADING-RECORD.
    Ø2 FILLER PIC X(1Ø) VALUE IS SPACES.
    Ø2 HEADING-1 PIC X(15)
       VALUE IS 'CUSTOMER NUMBER'.
    Ø2 FILLER PIC X(5) VALUE IS SPACES.
    Ø2 HEADING-2 PIC X(4)
       VALUE IS 'NAME'.
    Ø2 FILLER PIC X(36) VALUE IS SPACES.
    Ø2 HEADING-3 PIC X(12)
       VALUE IS 'HOME-ADDRESS'.
    Ø2 FILLER PIC X(39) VALUE IS SPACES.
```

Data Division entries to set up headings in HEADING-RECORD.

(1) The Value of HEADING-RECORD is moved to the output area OUTPUT-RECORD

Figure 46

* * *

HEADING-1
HEADING-2
HEADING-3
(These are user-supplied names and not reserved words.)

---

10. Figure 46 shows that the headings planned in Figure 45 have been set up as values of:

    a. an output area.

    b. a working-storage variable.

* * *

b

---

11. By setting up HEADING-RECORD in working storage a programmer can use a VALUE clause to assign:

    a. a value of blanks to FILLER items to provide for horizontal spacing of headings.

    b. the values CUSTOMER-NUMBER, NAME, and HOME-ADDRESS to HEADING-1, HEADING-2, and HEADING-3, respectively.

* * *

Both

---

12. In a record description entry for a heading record, a VALUE clause is used to specify:

    a. the headings that will be printed on a printer page.

    b. blanks to provide for horizontal spacing of headings.

* * *

Both

---

13. In Figure 46, the quotation marks enclosing CUSTOMER-NUMBER in the VALUE clause for HEADING-1 are:

    a. part of the value being assigned to HEADING-1.

    b. not counted in the PICTURE repetition factor which specifies the length of the variable.

* * *

b (The quotation marks indicate a constant that is a nonnumeric literal. They are not part of the value.)

This figure shows a format for headings on the Printer Spacing Chart.

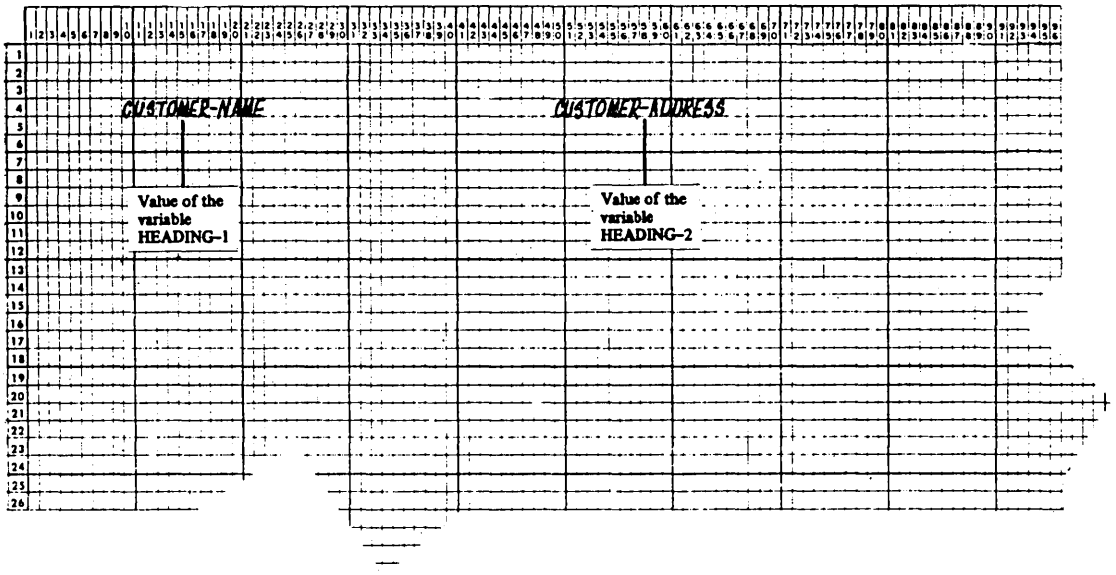


Figure 47

14. The Printer Spacing Chart in Figure 47 shows another set of headings and headings variables. Select the PICTURE clause that would be specified for each variable. (You may refer to Figure 44.)

1) HEADING-1	a. PICTURE A(13)
2) HEADING-2	b. PICTURE X(16)
	c. PICTURE X(13)
	d. PICTURE A(16)

* * *

1) c
2) b

---

15. Write the Working-Storage Section entries for HEADING-RECORD to set up headings to be printed as shown in Figure 47.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WORKING-STORAGE SECTION.
       01  HEADING-RECORD.
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  HEADING-1 PIC X(13)
               VALUE IS 'CUSTOMER-NAME'.
           02  FILLER PIC X(27) VALUE IS SPACES.
           02  HEADING-2 PIC X(16)
               VALUE IS 'CUSTOMER-ADDRESS'.
           02  FILLER PIC X(55) VALUE IS SPACES.
```

---

16. Figure 46 shows that the output file PRINT will contain a heading record and several data records. In Figure 46 the heading record and the data record are set up:

    a. in separate working-storage variables.

    b. alternately in the same working-storage variable.

* * *

a

---

17. Values of variables in working storage:

    a. can be transmitted directly to an output file with the simple form of the WRITE statement.

    b. must be moved to an output area before they can be transmitted to an output file with the simple form of the WRITE statement.

* * *

b

---

18. Figure 46 shows that values of both the heading variable and the data variable will be transmitted to the output file PRINT. In Figure 46:

    a. a separate output area has been set up for each of the record variables.

    b. a single output area is to be used for the record variables.

* * *

b

---

19. You will frequently be asked to provide a listing of records in a format outlined on a Printer Spacing Chart. A problem of this type is described in Figure 48. Read the problem statement and code the Data Division.

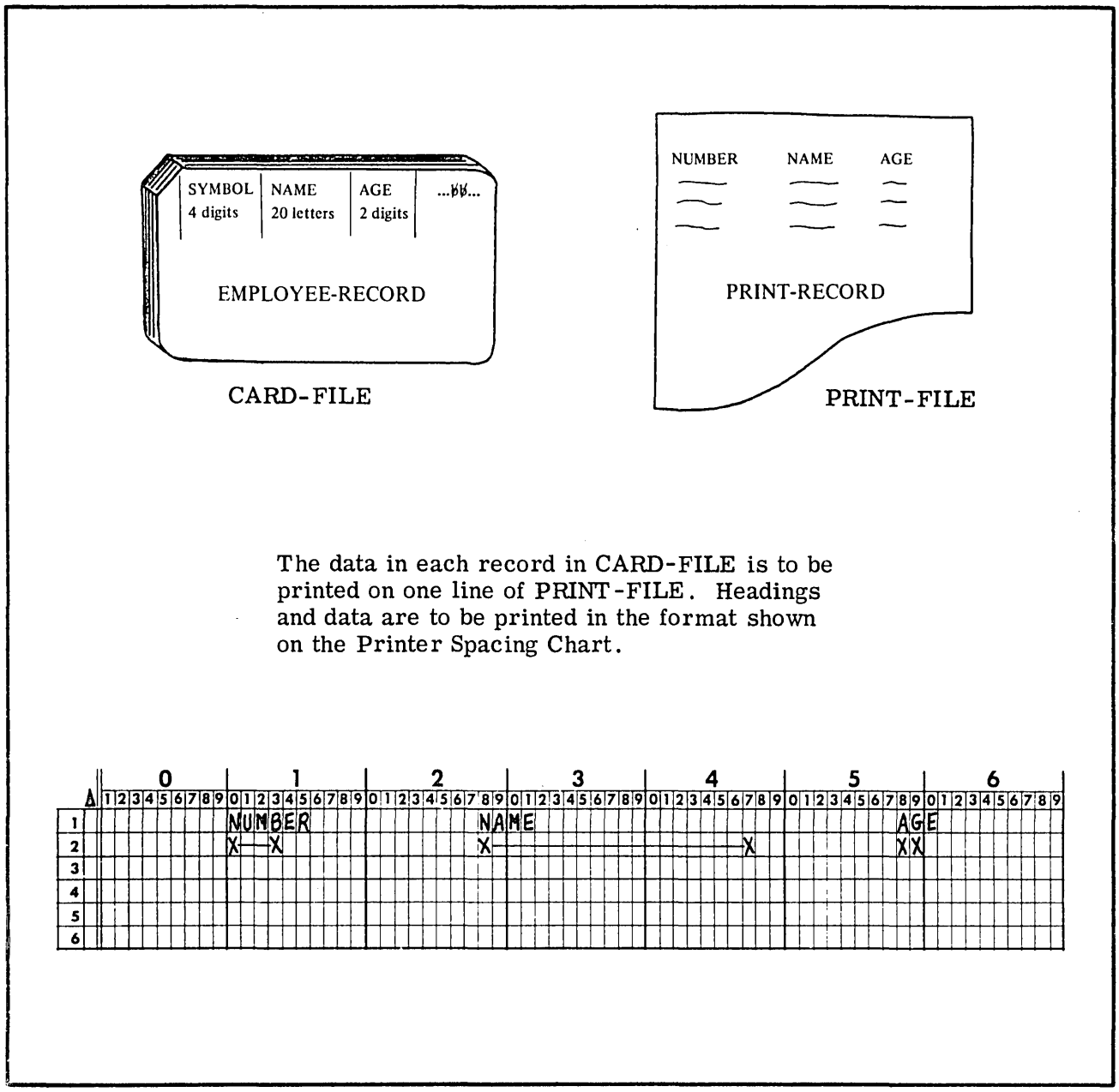


Figure 48

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  EMPLOYEE-RECORD.
           02  SYMBOL PIC 9(4).
           02  NAME PIC A(20).
           02  AGE PIC 9(2).
           02  FILLER PIC X(54).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(120).
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD.
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  HEADING-1 PIC A(6)
               VALUE IS 'NUMBER'.
           02  FILLER PIC X(12) VALUE IS SPACES.
           02  HEADING-2 PIC A(4) VALUE IS 'NAME'.
           02  FILLER PIC X(26) VALUE IS SPACES.
           02  HEADING-3 PIC A(3) VALUE IS 'AGE'.
           02  FILLER PIC X(60) VALUE IS SPACES.
       01  WORKING-RECORD.
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  SYMBOL PIC 9(4).
           02  FILLER PIC X(14) VALUE IS SPACES.
           02  NAME PIC A(20).
           02  FILLER PIC X(10) VALUE IS SPACES.
           02  AGE PIC 9(2).
           02  FILLER PIC X(61) VALUE IS SPACES.
```

---

20. The first variable containing values that must be moved to the output are first in order to print the line shown at the top of the printer page in Figure 46 is ..... .

                        *   *   *

HEADING-RECORD

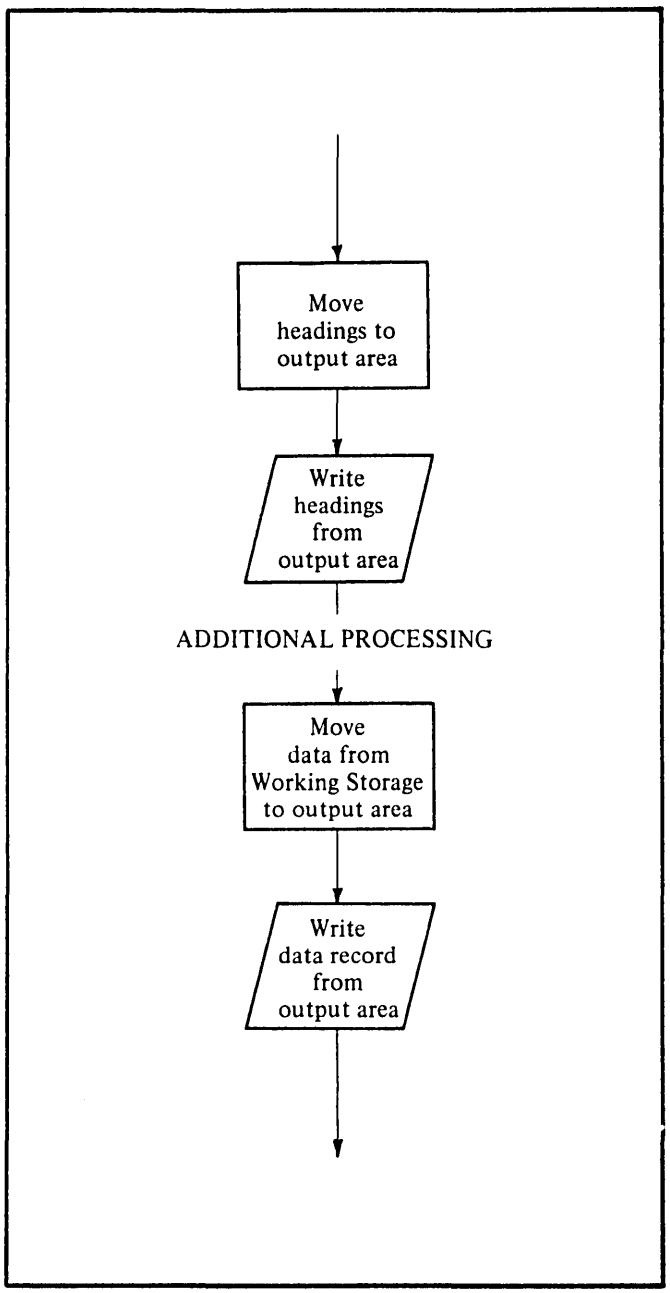


Figure 49

21. Figure 49 shows a segment of a flow chart for writing headings and data on a printer page. According to Figure 49 writing a heading record and a data record:

    a. requires two WRITE statements.

    b. is done with the same WRITE statement.

* * *

a

---

22. In order to write a heading record and a data record shown in Figure 46, you would use the statements:

    a. WRITE HEADING-RECORD.
       WRITE WORKING-RECORD.

    b. WRITE OUTPUT-RECORD.
       WRITE OUTPUT-RECORD.

                    *   *   *

b

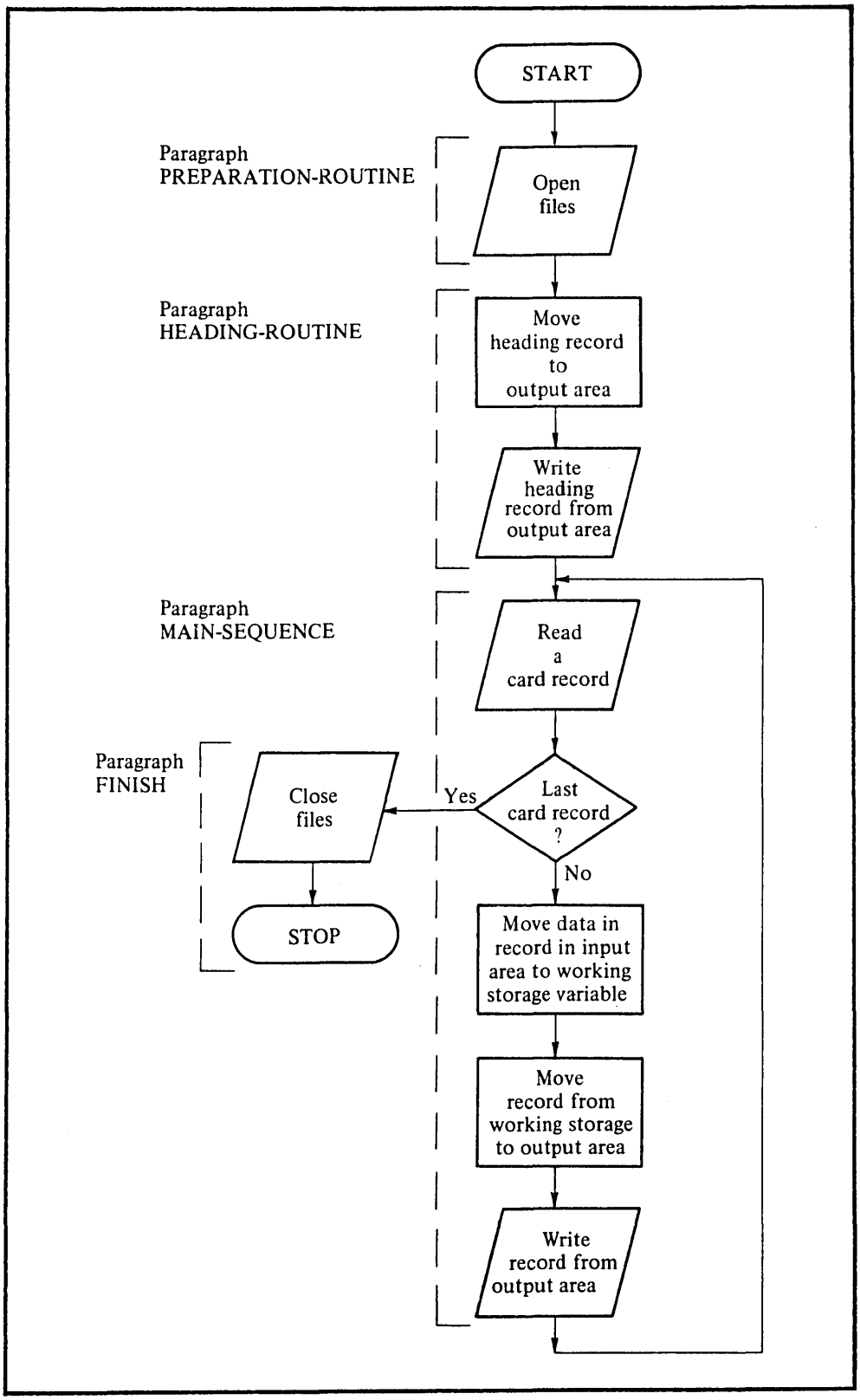

START
Paragraph
PREPARATION-ROUTINE
Open
files
Paragraph
HEADING-ROUTINE
Move
heading record
to
output area
Write
heading
record from
output area
Paragraph
MAIN-SEQUENCE
Read
a
card record
Paragraph
FINISH
Close
files
Yes
Last
card record
?
No
STOP
Move data in
record in input
area to working
storage variable
Move
record from
working storage
to output area
Write
record from
output area

Figure 50

23. Figure 50 is a flow chart for the Procedure Division for problems like the one described in Figure 48. The flow chart shows that headings are written before:

    a. a record is transmitted from the input file.

    b. the input file is opened.

* * *

a

---

24. A heading record is written before a data record is transmitted from the input file because:

    a. the data record will be transmitted to the heading variable, destroying the heading values.

    b. headings will be written only once on a page while several data records will be read and written on that page.

* * *

b

25.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  EMPLOYEE-RECORD.
           02  SYMBOL PIC 9(4).
           02  NAME PIC A(20).
           02  AGE PIC 9(2).
           02  FILLER PIC X(54).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(120).
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD.
           02  FILLER PIC X(10)
                   VALUE IS SPACES.
           02  HEADING-1 PIC A(6)
                   VALUE IS 'NUMBER'.
           02  FILLER PIC X(12)
                   VALUE IS SPACES.
           02  HEADING-2 PIC A(4)
                   VALUE IS 'NAME'.
           02  FILLER PIC X(26)
                   VALUE IS SPACES.
           02  HEADING-3 PIC A(3)
                   VALUE IS 'AGE'.
           02  FILLER PIC X(60)
                   VALUE IS SPACES.
       01  WORKING-RECORD.
           02  FILLER PIC X(10)
                   VALUE IS SPACES.
           02  SYMBOL PIC 9(4).
           02  FILLER PIC X(14)
                   VALUE IS SPACES.
           02  NAME PIC A(20).
           02  FILLER PIC X(10)
                   VALUE IS SPACES.
           02  AGE PIC 9(2).
           02  FILLER PIC X(61)
                   VALUE IS SPACES.
```

Follow the flow chart in Figure 50 and code the Procedure Division for the problem described in Figure 48. The Data Division for this problem, which you wrote in a preceding frame, is reproduced above.

* * *

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARD-FILE
                OUTPUT PRINT-FILE.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
       MAIN-SEQUENCE.
           READ CARD-FILE AT END GO TO FINISH.
           MOVE SYMBOL OF EMPLOYEE-RECORD
               TO SYMBOL OF WORKING-RECORD.
           MOVE NAME OF EMPLOYEE-RECORD
               TO NAME OF WORKING-RECORD.
           MOVE AGE OF EMPLOYEE-RECORD
               TO AGE OF WORKING-RECORD.
           MOVE WORKING-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE CARD-FILE PRINT-FILE.
           STOP RUN.
```

---

26. A programmer sets up variables in working storage in order to:

    a. use a VALUE clause to assign an initial value to a variable.

    b. rearrange data items from another record.

    c. provide for horizontal spacing by inserting blanks before, between, and after data items.

* * *

Any of these

---

27. Moving elementary data items from one variable to another is necessary if:

    a. FILLER items are to be inserted for horizontal spacing.

    b. data items in one record are to be arranged in a different order.

* * *

Either

The following two frames present independent topics.

28. HIGH-VALUE

HIGH-VALUES represent the highest value in a computer's collating sequence. The character for HIGH-VALUE is HEX 00FF.

LOW-VALUE

LOW-VALUES represent the lowest value in a computer's collating sequence. The character for LOW-VALUE is HEX 0000.

It is possible to move and process these characters.

MOVE HIGH-VALUES TO EMPLOYEE-NUMBER.

The above example will move HEX 00FF into the EMPLOYEE-NUMBER.

MOVE LOW-VALUES TO PRICE.

The above example will move HEX 0000 into the PRICE.

(HIGH-VALUE and LOW-VALUE are used, among other techniques, to define the beginnings and ends of data streams.)

---

29. The BLANK WHEN ZERO clause specifies that an item is to be set to blanks whenever its value is zero. The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric edited or numeric items.

MOVE AMOUNT TO PRINT BLANK WHEN ZERO.

If AMOUNT contains 1234 PRINT will contain 1234.
If AMOUNT contains 0000 PRINT will contain bbbb.

* * *

MOVE PRICE-1 TO PRINT-1 BLANK WHEN ZERO.

Match the following contents of variable PRICE-1 with the correct answers:

1) PRICE-1 contains 5673.	a. PRINT-1 will contain blanks
2) PRICE-1 contains 0000	b. PRINT-1 will contain 5673.

* * *

1) b
2) a

The problems in the next two frames incorporate what you have learned in printing a report with a two-line title. Coding the solutions is optional. If you do not code the solutions, read them carefully to make sure you understand them.

```
30.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

                                  CUSTOMER REPORT
                NAME                                ADDRESS
```

Figure 51

Printing a report title preceding the headings for columns of data is a common practice. Figure 52 shows how the flow chart in Figure 50 could be expanded to provide for a report title. The variable HEADING-RECORD-1 could be set up for the report title, and HEADING-RECORD-2 for the column headings. Code the Working-Storage Section of the Data Division to provide for the title and headings shown in the chart above. (Remember that data items consisting of letters and spaces may be described with either the picture character A or X.)

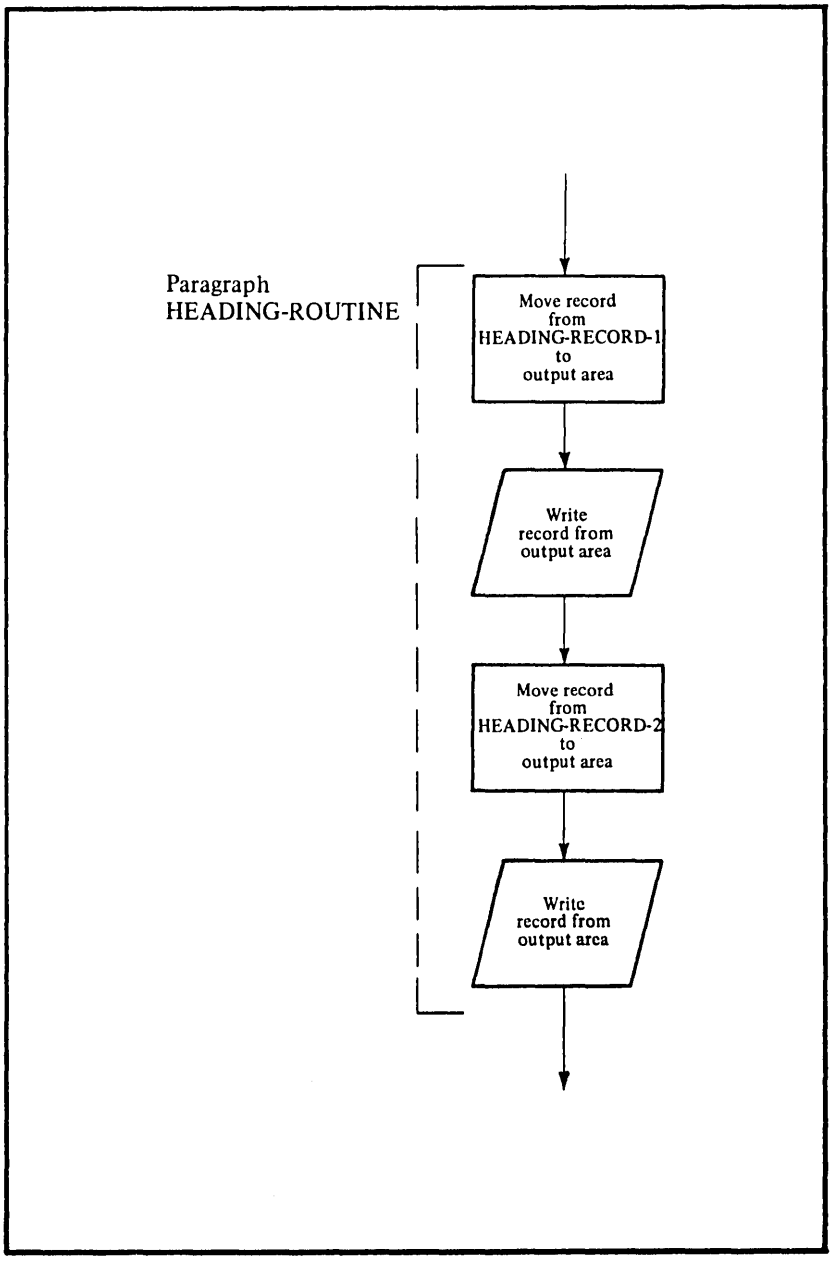

Paragraph
HEADING-ROUTINE
Move record from HEADING-RECORD-1 to output area
Write record from output area
Move record from HEADING-RECORD-2 to output area
Write record from output area

Figure 52

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WORKING-STORAGE SECTION.
       01  HEADING-RECORD-1.
           02  FILLER PIC X(39) VALUE IS SPACES.
           02  TITLE PIC X(15)
               VALUE IS 'CUSTOMER REPORT'.
           02  FILLER PIC X(67) VALUE IS SPACES.
       01  HEADING-RECORD-2.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-1 PIC X(4)
               VALUE IS 'NAME'.
           02  FILLER PIC X(36) VALUE IS SPACES.
           02  HEADING-2 PIC X(7)
               VALUE IS 'ADDRESS'.
           02  FILLER PIC X(54) VALUE IS SPACES.
```

---

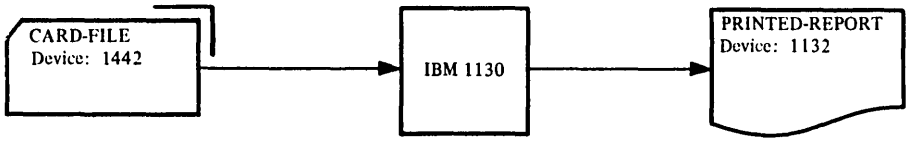
31. Write the paragraph HEADING-ROUTINE to print the title and headings shown in the preceding frame from the output area OUTPUT-RECORD.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       HEADING-ROUTINE.
           MOVE HEADING-RECORD-1
               TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD.
           MOVE HEADING-RECORD-2
               TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD.
```

---

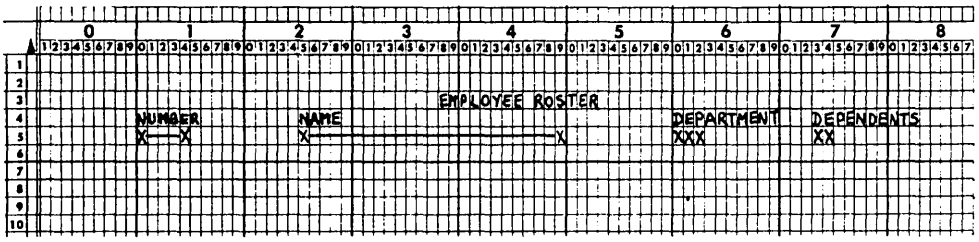
32. Figure 53 describes a problem that will give you an opportunity to practice using all the features you have learned in this lesson. The problem involves printing a report title, column headings, and data. Read the problem and code the Data Division and Procedure Division. (You may refer to Figures 50 and 52.)

Problem Statement



The system flow chart above shows the files and equipment to be used in this program. The forms of records in CARD-FILE and PRINTED-REPORT are illustrated below.

CARD-RECORD				
MARKER (5 digits)	DEPARTMENT (3 characters)	NAME (25 letters)	DEPENDENTS (2 digits)	FILLER (45 blanks)



The file PRINTED-REPORT is to consist of a title and headings followed by a listing of the records in CARD-FILE with the data items rearranged as shown in the Printer Spacing Chart. Use the variables CARD-RECORD, PRINT-RECORD, and WORK-RECORD.

Figure 53

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CARD-RECORD.
           02  MARKER PIC 9(5).                          (40)
           02  DEPARTMENT PIC X(3).
           02  NAME PIC A(25).                           (41)
           02  DEPENDENTS PIC 9(2).
           02  FILLER PIC X(45).
       FD  PRINTED-REPORT
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(120).                      (12)
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD-1.                             (45)
           02  FILLER PIC X(38) VALUE IS SPACES.
           02  TITLE PIC X(15)
               VALUE IS 'EMPLOYEE ROSTER'.
           02  FILLER PIC X(68) VALUE IS SPACES.         (14)
       01  HEADING-RECORD-2.                             (45)
           02  FILLER PIC X(10) VALUE IS SPACES.         (14)
           02  HEADING-1 PIC A(6)
               VALUE IS 'NUMBER'.
           02  FILLER PIC X(9) VALUE IS SPACES.          (14)
           02  HEADING-2 PIC A(4)
               VALUE IS 'NAME'.
           02  FILLER PIC X(31) VALUE IS SPACES.         (14)
           02  HEADING-3 PIC A(10)
               VALUE IS 'DEPARTMENT'.
           02  FILLER PIC X(3) VALUE IS SPACES.          (14)
           02  HEADING-4 PIC A(10)
               VALUE IS 'DEPENDENTS'.
           02  FILLER PIC X(38) VALUE IS SPACES.         (14)
       01  WORK-RECORD.
           02  FILLER PIC X(10) VALUE IS SPACES.         (14)
           02  MARKER PIC 9(5).
           02  FILLER PIC X(10) VALUE IS SPACES.         (14)
           02  NAME PIC A(25).
           02  FILLER PIC X(10) VALUE IS SPACES.         (14)
           02  DEPARTMENT PIC X(3).
           02  FILLER PIC X(10) VALUE IS SPACES.         (14)
           02  DEPENDENTS PIC 9(2).
           02  FILLER PIC X(46) VALUE IS SPACES.         (14)
```

```
0 0   1   1   2   2   3   3   4   4   5   5   6   6   7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARD-FILE
                OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.                                    (45)
           MOVE HEADING-RECORD-1
                TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           MOVE HEADING-RECORD-2
                TO PRINT-RECORD.
           WRITE PRINT-RECORD.
       MAIN-SEQUENCE.
           READ CARD-FILE AT END GO TO FINISH.
           MOVE MARKER OF CARD-RECORD
                TO MARKER OF WORK-RECORD.                  (65)
           MOVE DEPARTMENT OF CARD-RECORD
                TO DEPARTMENT OF WORK-RECORD.
           MOVE NAME OF CARD-RECORD
                TO NAME OF WORK-RECORD.
           MOVE DEPENDENTS OF CARD-RECORD
                TO DEPENDENTS OF WORK-RECORD.
           MOVE WORK-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE CARD-FILE PRINTED REPORT.
           STOP RUN.
```

SUMMARY:

You have now completed Lesson 9 in which you have learned to print report titles and headings as well as data records. You have learned to reserve the first position in an output area for the printer carriage control character and to work from a Printer Spacing Chart to produce a report in a specified format. You have learned to use the picture characters 9 and A to specify that a variable will have numeric or alphabetic values, respectively, and to use the figurative constant SPACES with the VALUE clause to assign an initial value to a variable.

END OF LESSON 9

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 10

LESSON 10 - VERTICAL SPACING (2)

INTRODUCTION

In the previous lesson you learned to set up record variables to provide for a title, column headings, and data records in a printed report. You also learned to provide horizontal spacing for each of the records in the report. The Procedure Division entries that you used, however, provide only for single-spaced records within a single page report. You will often be required to produce a report with one or more blank lines between various records, and the reports will usually be several pages in length.

In this lesson you will learn to specify vertical spacing in a printed report, including advancing to subsequent pages when a page has been filled.

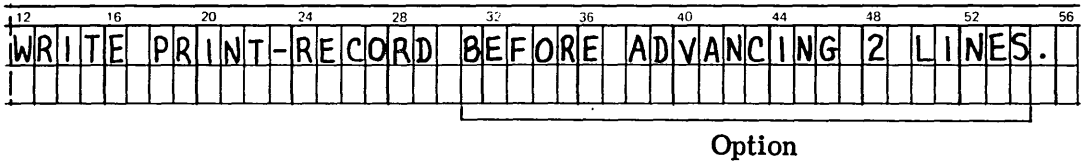
Specific COBOL language features you will learn to use are:

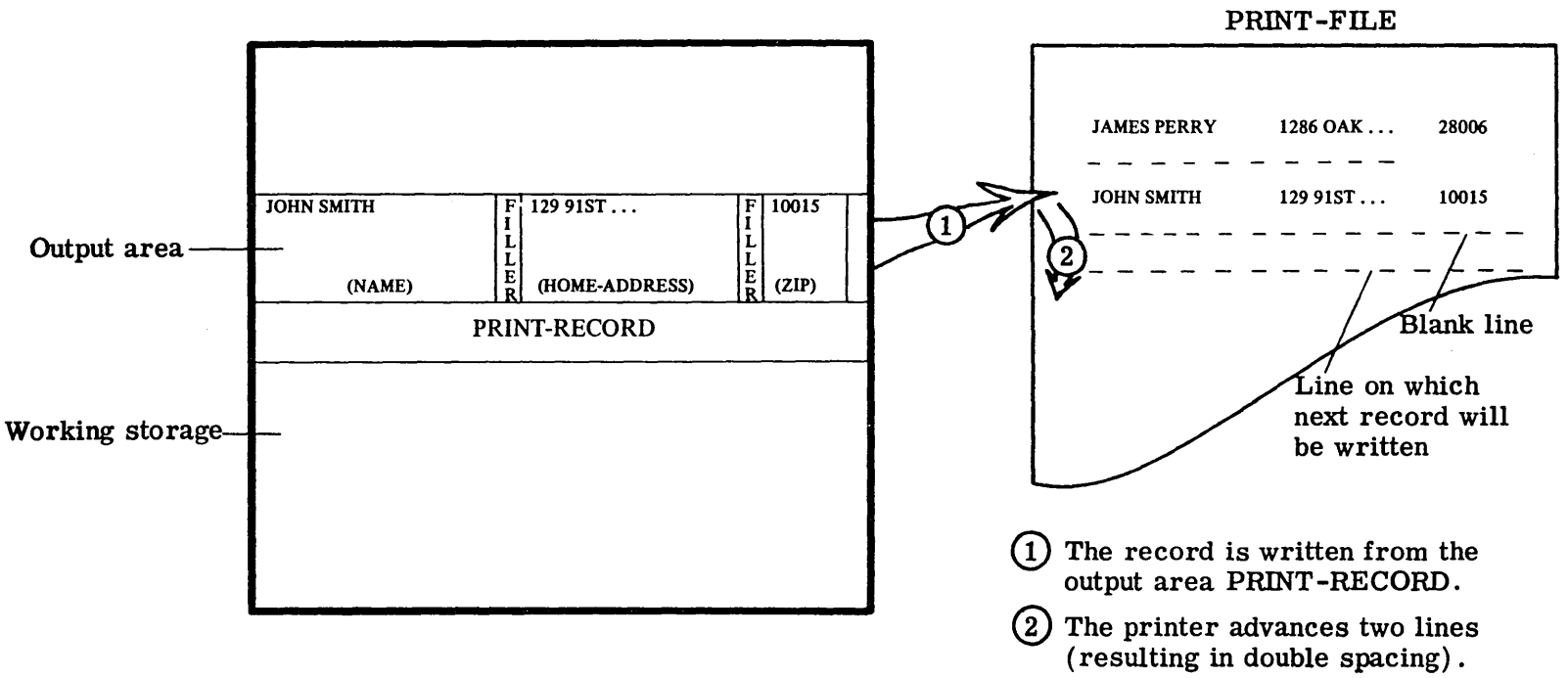
BEFORE ADVANCING option of the WRITE statement
AFTER ADVANCING option of the WRITE statement
AT END-OF-PAGE option of the WRITE statement
RESERVE clause

This lesson will require approximately three quarters of an hour.

In a previous lesson you provided for horizontal spacing by moving data items to a record in working storage containing FILLER items to which values of blanks had been assigned. You specified no vertical spacing, so single spacing, which is automatic for the 1132 printer, was provided. You can, however, specify whatever spacing you wish for a printed report. Unlike horizontal spacing, vertical spacing is specified in options of the WRITE statement.

This diagram illustrates execution of the statement





(1) The record is written from the output area PRINT-RECORD.

(2) The printer advances two lines (resulting in double spacing).

Figure 54

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WRITE record-name
    BEFORE ADVANCING integer LINES.
```

Option

The BEFORE ADVANCING option of the WRITE statement shown in the general format above can be used to specify vertical spacing. Figure 54 shows the result when integer is 2. Read the explanation and look at the illustration in the figure. Execution of the statement in Figure 54 causes:

a. a record to be written after the printer advances two lines.

b. the printer to advance two lines after the record is written.

* * *

b

---

2. Figure 54 shows that the result when integer in the BEFORE ADVANCING option is 2 is:

a. one blank line between two printed lines.

b. double-spaced records.

* * *

Both

---

3. Which of the following integers would be specified in the BEFORE ADVANCING option to produce three blank lines between records?

a. 4

b. 3

* * *

a

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE HEADING-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
               .
               .
               .
           MOVE DATA-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
               .
               .
               .


           WRITE record-name
               BEFORE ADVANCING integer LINES.
```

Instructions to write headings and data records are shown in the program segment above. Using the general form of the WRITE statement with the BEFORE ADVANCING option shown above as a guide, rewrite the WRITE statements in the program segment to:

1) triple space (insert two blank lines) after writing the headings.

2) double space (insert one blank line) after writing a data record.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 3 LINES.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 2 LINES.
```

---

5. Write a statement that will cause records from OUTPUT-RECORD to be printed with two blank lines between records.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE OUTPUT-RECORD
               BEFORE ADVANCING 3 LINES.
```

Options of the WRITE Statement

Option	Action Taken	Restrictions on user-supplied portion
BEFORE ADVANCING integer LINES	Printer advances the specified number of lines after record is written. (Records are separated by integer-1 blank lines.)	May be any positive integer less than 100
AFTER ADVANCING integer LINES	Printer advances the specified number of lines before record is written.	May be any positive integer less than 100.
AFTER ADVANCING mnemonic-name when mnemonic name has been defined in SPECIAL-NAMES paragraph in the Configuration Section of Environment Division as:		Must fit rules for data names
CSP	Spacing is suppressed.	
C01 thru C09 for 1403 C01 thru C06; C09 for 1132	Printer skips to channels 1 through 9, respectively	
C10 thru C12 for 1403 C12 for 1132	Printer skips to 10, 11, 12, respectively	
AT END-OF-PAGE imperative-sentence	When END-OF-PAGE condition exists (channel 12 punch on carriage control tape is sensed by an on-line printer) the imperative statement is executed after writing and spacing operations have been completed. An error message will result if RESERVE NO ALTERNATE AREA has not been specified for the associated file in the Environment Division.	

1. When an ADVANCING option is specified in a WRITE statement for a record file, a for of that option must be specified in every WRITE statement for records in the same file.

2. The first character in each logical record for a printer file must be reserved by the user for the carriage control character.

Figure 55

6.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WRITE record-name
    AFTER ADVANCING integer LINES.
```

Option

Another option for specifying vertical spacing is shown above. Figure 55 is a chart of options that may be specified in the WRITE statement for printer files. The first column shows general forms of the options. The second column describes the action taken when a WRITE statement specifying the option is executed. Figure 55 shows that a WRITE statement containing the above option causes the records to be written:

a. before the printer advances.

b. after the printer has advanced.

* * *

b

---

7. Write a statement to print data records from OUTPUT-RECORD with triple spacing. Use an ADVANCING option to cause spacing before the records are written.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WRITE OUTPUT-RECORD
    AFTER ADVANCING 3 LINES.
```

---

8. The third column in Figure 55 specifies restrictions on the user-supplied portion of each option. Figure 55 shows that integer in the BEFORE ADVANCING option may be any ..... .

* * *

positive integer less than 100

---

9. To write records from PRINT-RECORD with three blank lines between records you would use the statement:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 3 LINES.

                              *         *         *
```

Wrong:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 4 LINES.
```

---

10. Refer to Figure 55 if necessary, and match each effect with the statement(s) that would cause it.

    1) Advance three lines after a record is written.

    2) Double spacing.

    3) Triple spacing.

    a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 2 LINES.
```

    b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               BEFORE ADVANCING 3 LINES

                              *         *         *
```

1) b
2) a
3) b

11.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WRITE record-name
    ...
    AT END-OF-PAGE imperative-sentence.
```

Option

The action to be taken when the end of a page is reached is specified in the AT END-OF-PAGE option of the WRITE statement, shown in the general format above. The imperative sentence in the AT END-OF-PAGE option is similar in use to the AT END option of the READ statement. The imperative sentence:

a. gives an instruction to be executed at the end of a page.

b. may be one or more imperative statements.

* * *

Both
(END-OF-PAGE may be abbreviated EOP. The ... represents an ADVANCING option that must be present.)

---

12. If headings are to be printed at the top of each page, the imperative sentence in the AT END-OF-PAGE option might be a statement that will cause a branch:

    a. back to the instructions to write headings.

    b. to the instruction to read a card.

* * *

a

---

13. The statement that is used to cause a branch is:

    a. MOVE

    b. GO TO

* * *

b

---

14. In the GO TO statement you must specify:

    a. a specific statement that is to be executed.

    b. the paragraph name that precedes the statements to which the program will branch.

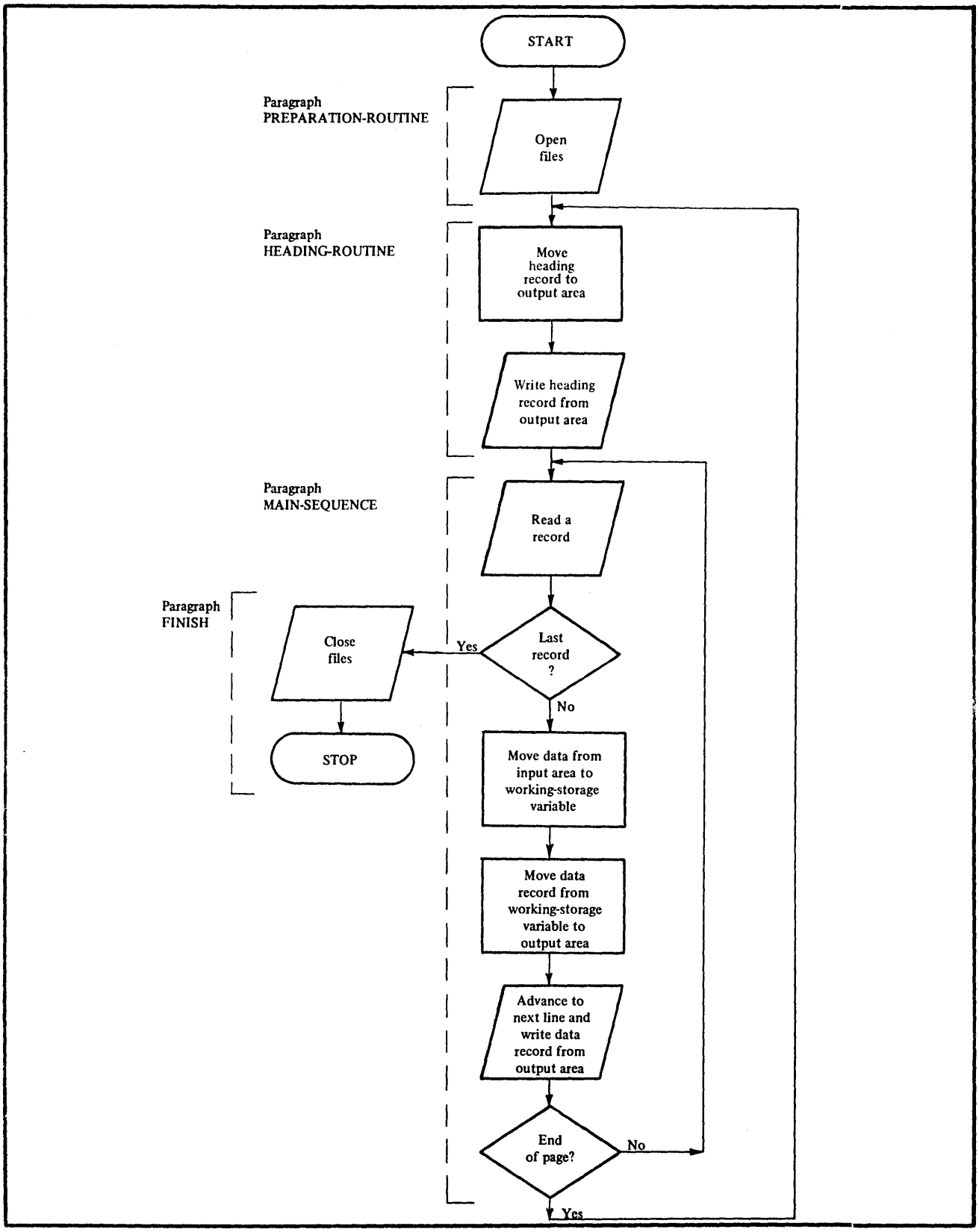
* * *

b

---

15. The statements to move headings and then write them into an output file are preceded by the paragraph name HEADING-ROUTINE. Write a statement to write a record from PRINT-RECORD, after skipping 7 lines on the report, including an instruction to branch to the heading routine when the end of a page is reached.

```
                                *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING 7 LINES
               AT END-OF-PAGE
                   GO TO HEADING-ROUTINE.
```


START
Paragraph
PREPARATION-ROUTINE
Open
files
Paragraph
HEADING-ROUTINE
Move
heading
record to
output area
Write heading
record from
output area
Paragraph
MAIN-SEQUENCE
Read a
record
Paragraph
FINISH
Close
files
Yes
Last
record
?
No
STOP
Move data from
input area to
working-storage
variable
Move data
record from
working-storage
variable to
output area
Advance to
next line and
write data
record from
output area
End
of page?
No
Yes

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARDFILE
                OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD
                TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD.
       MAIN-SEQUENCE.
           READ CARDFILE
                AT END GO TO FINISH.

           WRITE OUTPUT-RECORD
                AFTER ADVANCING 2 LINES
                AT END-OF-PAGE
                     GO TO HEADING-ROUTINE.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE CARDFILE PRINTED-REPORT.
           STOP RUN.
```

Figure 56

16. Figure 56 shows a program flow chart and corresponding coding for skipping to a new page and printing headings when the end of a page has been reached. The flow chart and the coding indicate that:

    a. a test for the end of the page and a branch to HEADING-ROUTINE can be specified in the statement to write a data record.

    b. a skip to a new page can be specified in the statement to write a heading record.

* * *

a

---

17. The AT END-OF-PAGE option in Figure 56 specifies a:

    a. skip to a new page

    b. branch to the paragraph HEADING-ROUTINE which contains a WRITE statement

* * *

b

---

18. Using the record variables shown in Figure 57, write the Procedure Division entries for the first three steps in the flow chart in Figure 56.

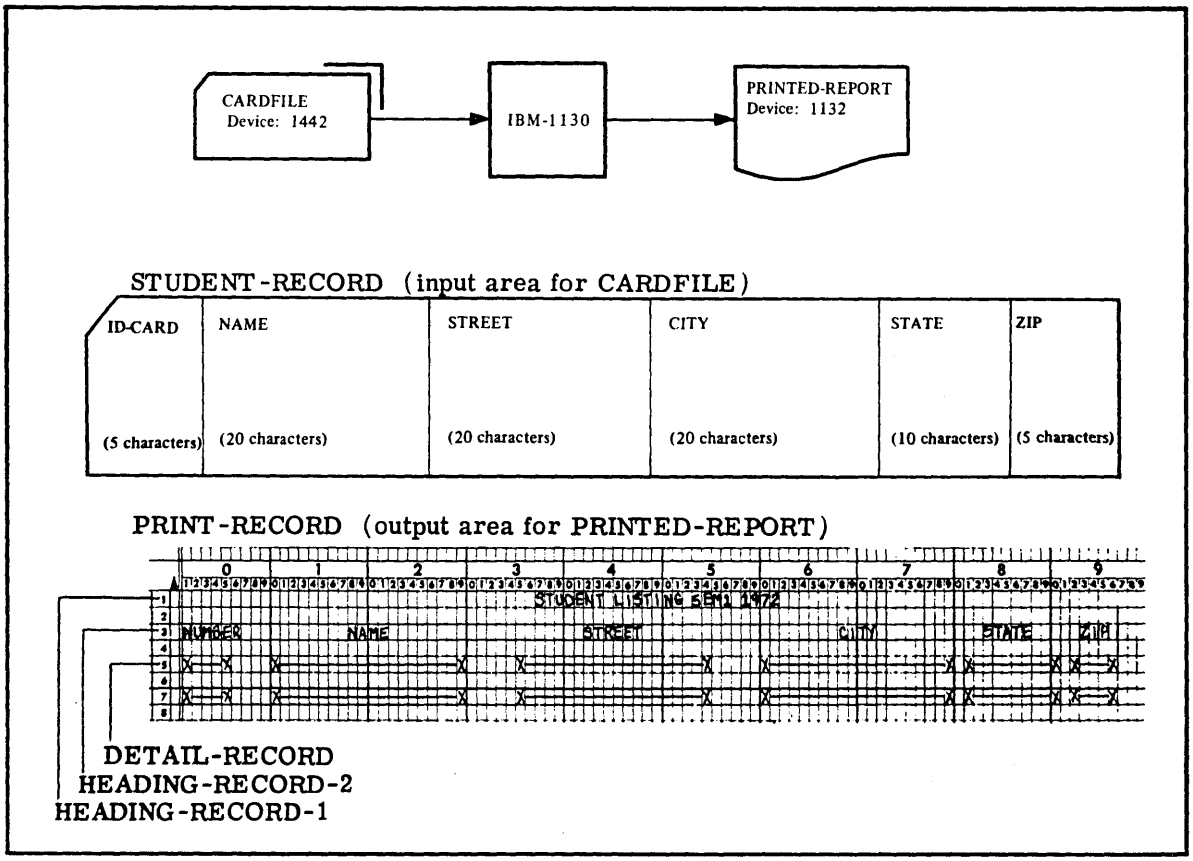


Figure 57

```
                                *         *         *
0    0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARDFILE
                OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
```

---

19. Write the Procedure Division entries for the last three steps in the paragraph MAIN-SEQUENCE in the flow chart in Figure 56, using the record variables illustrated in Figure 57. Include options to provide double spacing and branching at the end of a page.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE DETAIL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES
               AT END-OF-PAGE
                   GO TO HEADING-ROUTINE.
```

Alternate solution:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE DETAIL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               BEFORE ADVANCING 2 LINES
               AT END-OF-PAGE
                   GO TO HEADING-ROUTINE.
```

20.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           RESERVE NO ALTERNATE AREA
```

The use of the AT END-OF-PAGE option requires the form of the RESERVE clause shown above in the FILE-CONTROL paragraph of the program. This form of the RESERVE clause specifies that the file is not to be double buffered, which is the standard condition for a file. In which division of a program would the RESERVE clause be specified?

* * *

Environment Division

---

21. Which of the following would permit the use of the AT END-OF-PAGE option in a WRITE statement referring to PRINT-FILE?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FILE SECTION.
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED
           RESERVE NO ALTERNATE AREA.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FILE-CONTROL.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FILE-CONTROL.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREA.
```

* * *

c
(a is in the wrong division; b does not contain the required clause.)

---

22. Write a paragraph that will permit WRITE statements referring to the printer file PRINTED-REPORT to use the AT END-OF-PAGE option. Include the division and section headers.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREA.
```

SUMMARY:

You have now mastered one of the most intricate aspects of COBOL coding - the programming of horizontal and vertical controls for printed output. The next lesson will show you how you may assign special mnemonic names to be used in vertical forms of skipping or spacing, instead of using the less meaningful numeric carriage tape designations. You will also put into coding practice some of the new concepts you just learned.

END OF LESSON 10

LESSON 11

LESSON 11 - VERTICAL SPACING CONTROL FOR PRINTER OUTPUT

INTRODUCTION

In this lesson you will learn to substitute convenient mnemonic names for printing control characters used to control vertical spacing of printed output forms. You will also study usage of the OBJECT-COMPUTER paragraph in the Environment Division.

Specific COBOL language features you will study are:

SPECIAL-NAMES paragraph
OBJECT-COMPUTER paragraph

This lesson will require approximately three quarters of an hour.

1. You have now learned to code all the entries for producing a listing of the records in an input file in any specified format. The problem described below will give you an opportunity to practice coding the entire program for producing a listing.

   A listing of all the students enrolled in a community college for the first semester of 1972 is to be produced from the student records in a card file. The system flow chart and an illustration of the student record are shown in Figure 58. The format for the listing is shown in the Printer Spacing Chart in the figure along with the Data Division for the program LISTING to produce this listing. Code the other three divisions for LISTING. Use the elementary variable names from STUDENT-RECORD in DETAIL-RECORD.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE SECTION.
       FD  CARDFILE
           LABEL RECORDS ARE OMITTED.
       01  STUDENT-RECORD.
           02  ID-CARD PIC X(5).
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY PIC X(20).
           02  STATE PIC X(10).
           02  ZIP PIC X(5).
       FD  PRINTED-REPORT
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(120).
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD-1.
           02  FILLER PIC X(37) VALUE IS SPACES.
           02  TITLE PIC X(25) VALUE IS
               'STUDENT LISTING SEM1 1972'.
           02  FILLER PIC X(59) VALUE IS SPACES.
       01  HEADING-RECORD-2.
           02  FILLER PIC X VALUE IS SPACES.
           02  HEADING-1 PIC X(6)
                   VALUE IS 'NUMBER'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-2 PIC X(4)
                   VALUE IS 'NAME'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-3 PIC X(6)
                   VALUE IS 'STREET'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-4 PIC X(4)
                   VALUE IS 'CITY'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-5 PIC X(5)
                   VALUE IS 'STATE'.
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  HEADING-6 PIC X(3)
                   VALUE IS 'ZIP'.
           02  FILLER PIC X(25) VALUE IS SPACES.
```

```
01  DETAIL-RECORD.
    02  FILLER PIC X VALUE IS SPACES.
    02  ID-CARD PIC X(5).
    02  FILLER PIC X(4) VALUE IS SPACES.
    02  NAME PIC X(20).
    02  FILLER PIC X(5) VALUE IS SPACES.
    02  STREET PIC X(20).
    02  FILLER PIC X(5) VALUE IS SPACES.
    02  CITY PIC X(20).
    02  FILLER PIC X(5) VALUE IS SPACES.
    02  STATE PIC X(10).
    02  FILLER PIC X VALUE IS SPACES.
    02  ZIP PIC X(5).
    02  FILLER PIC X(21) VALUE IS SPACES.
```

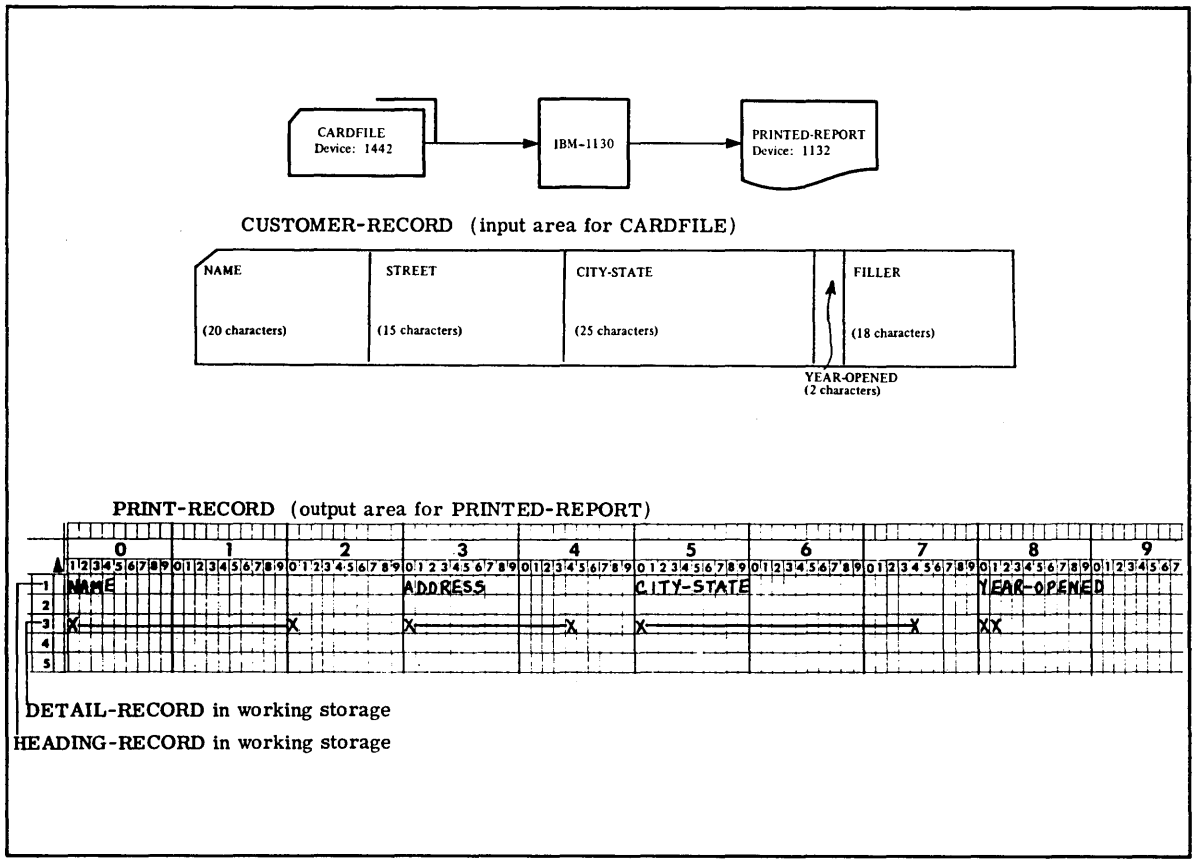


Figure 58

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LISTING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE CONTROL.
           SELECT CARDFILE
               ASSIGN TO RD-1442.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C.
               RESERVE NO ALTERNATE AREA.                (20)
       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARDFILE
               OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD-1
               TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 1 LINES.
           MOVE HEADING-RECORD-2
               TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES.
       MAIN-SEQUENCE.
           READ CARDFILE
               AT END GO TO FINISH.
           MOVE ID-CARD OF STUDENT-RECORD
               TO ID-CARD OF DETAIL-RECORD.
           MOVE NAME OF STUDENT-RECORD
               TO NAME OF DETAIL-RECORD.
           MOVE STREET OF STUDENT-RECORD
               TO STREET OF DETAIL-RECORD.
           MOVE CITY OF STUDENT-RECORD
               TO CITY OF DETAIL-RECORD.
           MOVE STATE OF STUDENT-RECORD
               TO STATE OF DETAIL-RECORD.
           MOVE ZIP OF STUDENT-RECORD
               TO ZIP OF DETAIL-RECORD.
           MOVE DETAIL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES.
               AT END-OF-PAGE                            (11)
                   GO TO HEADING-ROUTINE.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE CARDFILE PRINTED-REPORT.
           STOP RUN.
```

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WRITE record-name
    AFTER ADVANCING mnemonic-name.
```

Option

The option of the WRITE statement shown above may also be used to skip to the top of a page. Figure 55 shows that this option may be used to skip to a new page when mnemonic-name has been defined in the ..... paragraph of the Configuration Section of the Environment Division as ..... .

* * *

```
SPECIAL-NAMES
C01
```

---

3. The system name (C01, for example) cannot be used in a statement, so it must be given a mnemonic name by the programmer. This is done in the:

   a. File Section of the Data Division.

   b. SPECIAL-NAMES paragraph of the Environment Division.

* * *

b

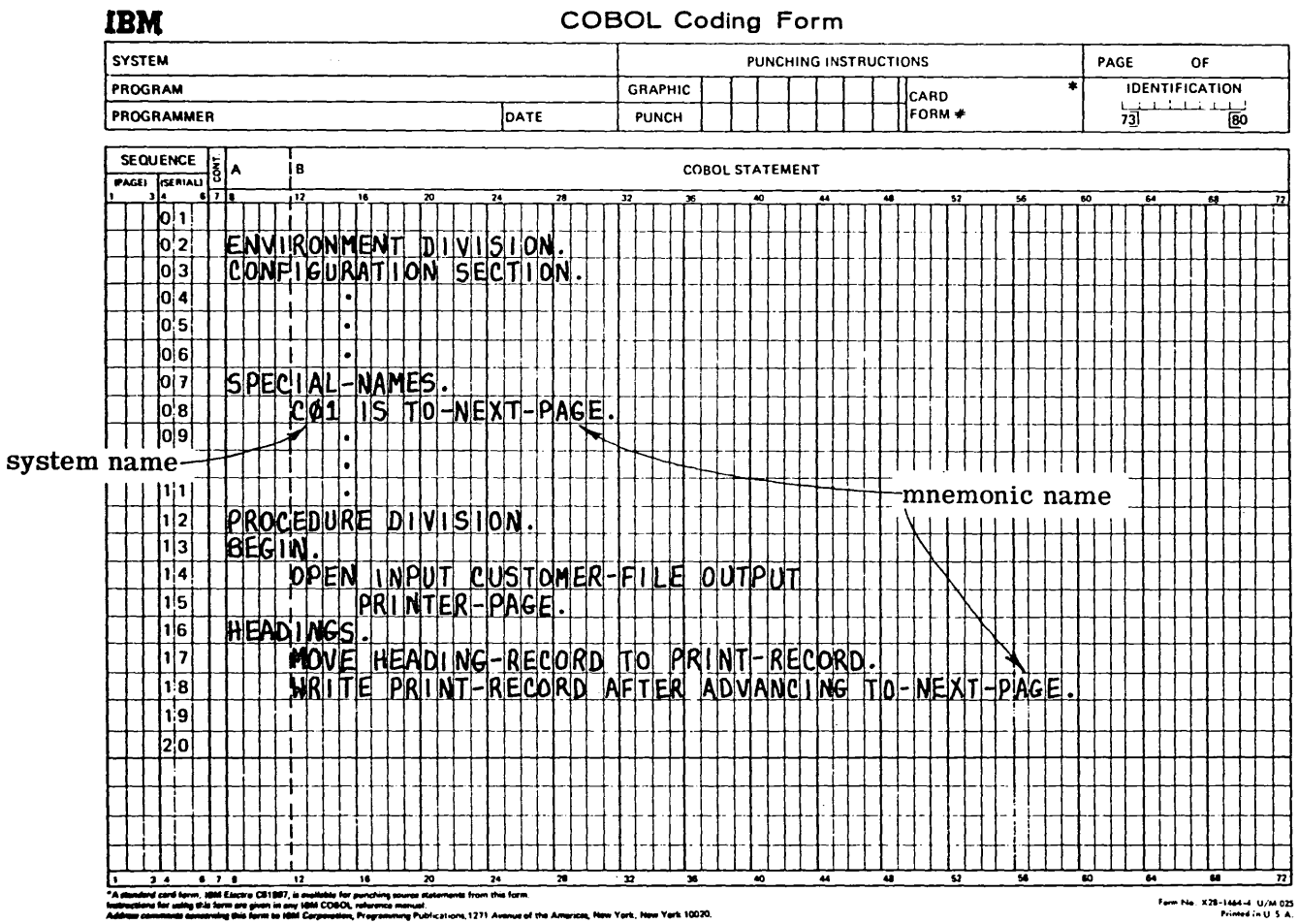

IBM
COBOL Coding Form
SYSTEM
PROGRAM
PROGRAMMER
DATE
PUNCHING INSTRUCTIONS
GRAPHIC
PUNCH
CARD FORM #
PAGE OF
IDENTIFICATION
73 80
SEQUENCE
(PAGE) (SERIAL)
CONT
A
B
COBOL STATEMENT
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
C01 IS TO-NEXT-PAGE.
PROCEDURE DIVISION.
BEGIN.
OPEN INPUT CUSTOMER-FILE OUTPUT
PRINTER-PAGE.
HEADINGS.
MOVE HEADING-RECORD TO PRINT-RECORD.
WRITE PRINT-RECORD AFTER ADVANCING TO-NEXT-PAGE.
system name
mnemonic name

Figure 59

4. The mnemonic name is a name chosen by the programmer that is defined in the SPECIAL-NAMES paragraph in the Configuration Section of the Environment Division. Figure 59 shows a mnemonic name defined in the SPECIAL-NAMES paragraph. In Figure 59:

   a. the mnemonic name is TO-NEXT-PAGE.

   b. the system name by which the mnemonic name is defined is C01.

* * *

Both

---

5. Match, using Figure 59.

1)	Mnemonic name	a.	C01
2)	System name	b.	AT END-OF-PAGE
		c.	TO-NEXT-PAGE

* * *

1) c
2) a

---

6. In Figure 59 the system name C01 specifies a skip to the first printing line. Its related mnemonic name TO-NEXT-PAGE, then, when used in a WRITE statement:

   a. specifies a skip to the first printing line.

   b. is used in place of C01 to cause a skip to a new page.

* * *

Both
(Remember that the system name C01 cannot be specified in a statement.)

7.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          WRITE record-name
              AFTER ADVANCING mnemonic-name.
```

Using the AFTER ADVANCING option, write a statement to write a record from OUTPUT-RECORD after a skip to the first printing line. Use the mnemonic name from Figure 59.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          WRITE OUTPUT-RECORD
              AFTER ADVANCING TO-NEXT-PAGE.
```

---

8. The mnemonic name specified for C01 in the AFTER ADVANCING option of the WRITE statement:

   a. could be any name of the programmer's choice.

   b. must always be TO-NEXT-PAGE.

* * *

a

---

9. Using Figure 20 as a guide, write the Environment Division entries to define the mnemonic name TO-FIRST-LINE as the system name C01.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       SPECIAL-NAMES.
           C01 IS TO-FIRST-LINE.
```

(If used, the SPECIAL-NAMES paragraph always follows the OBJECT-COMPUTER paragraph.)

---

10. The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed. This paragraph is entered into the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

```
OBJECT-COMPUTER. computer-name

    [MEMORY SIZE integer   {WORDS     }  ].
                           {CHARACTERS}
                           {MODULES   }
```

The following example demonstrates the complete Configuration Section.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130 MEMORY SIZE 8192 WORDS.
       SPECIAL-NAMES.
           C01 IS TO-FIRST-LINE.
```

---

11. Write a statement to write a record from OUTPUT-RECORD after a skip to the first printing line. Use the AFTER ADVANCING option and the name defined in the previous frame.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE OUTPUT-RECORD
               AFTER ADVANCING TO-FIRST-LINE.
```

---

12. A skip to the first printing line may be specified with the statement:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING TO-NEXT-PAGE.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING C01.
```

* * *

a

---

13. So far you have learned that a skip to the first printing line may be specified in the statement:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING TO-NEXT-PAGE.
```

---

14. A mnemonic name must be specified in the SPECIAL-NAMES paragraph if you are using the:

a. AFTER ADVANCING option with integer

b. AFTER ADVANCING option with mnemonic name

* * *

b

---

15. If the ADVANCING option is used in a WRITE statement for a file, every other WRITE statement for that file must contain some form of the same option. To produce the single-spaced output file PRINTED-REPORT, you could use the statement:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING 0.
```

to write headings, and the statement

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES.
```

to write data records.

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING TO-NEXT-PAGE.
```

to write headings, and the statement

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD
               AFTER ADVANCING 1 LINES.
```

to write data records.

* * *

b
AFTER ADVANCING 1 LINES is necessary even though normal single spacing is desired. All WRITE statements referring to PRINT-RECORD must contain AFTER ADVANCING clauses once the option is used.

16.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINTED-REPORT
               AFTER ADVANCING TO-TOP-OF-PAGE
```

Single spacing is used in a printed report for all records except the heading record, which is on a new page. Paragraph PRINT-HEADINGS contains the statement above. The statement you would use to cause a branch to PRINT-HEADINGS at the appropriate time would be:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-REPORT
               AT END-OF-PAGE
               GO TO PRINT-HEADINGS.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINTED-REPORT
               AFTER ADVANCING 1 LINE
               AT END-OF-PAGE
               GO TO PRINT-HEADINGS.
```

* * *

b

---

17. Rewrite the Configuration Section of the Environment Division and the Procedure Division of the program in Figure 60 to produce the listing in Figure 58 using the AFTER ADVANCING option. Use NEW-PAGE as the mnemonic name for C01.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LISTING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE CONTROL.
           SELECT CARDFILE
               ASSIGN TO RD-1442.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREA.

       DATA DIVISION.
       FILE SECTION.
       FD  CARDFILE
           LABEL RECORDS ARE OMITTED.
       01  STUDENT-RECORD.
           02  ID-CARD PIC X(5).
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY PIC X(20).
           02  STATE PIC X(10).
           02  ZIP PIC X(5).
       FD  PRINTED-REPORT.
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD-1.
           02  FILLER PIC X(37) VALUE IS SPACES.
           02  TITLE PIC X(25) VALUE IS
               'STUDENT LISTING SEM1 1972'.
           02  FILLER PIC X(59) VALUE IS SPACES.
       01  HEADING-RECORD-2.
           02  FILLER PIC X VALUE IS SPACES.
           02  HEADING-1 PIC X(6)
                   VALUE IS 'NUMBER'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-2 PIC X(4)
                   VALUE IS 'NAME'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-3 PIC X(6)
                   VALUE IS 'STREET'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-4 PIC X(4)
                   VALUE IS 'CITY'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-5 PIC X(5)
                   VALUE IS 'STATE'.
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  HEADING-6 PIC X(3)
                   VALUE IS 'ZIP'.
           02  FILLER PIC X(25) VALUE IS SPACES.
```

```
01  DETAIL-RECORD.
    02  FILLER PIC X VALUE IS SPACES.
    02  ID-CARD PIC X(5).
    02  FILLER PIC X(4) VALUE IS SPACES.
    02  NAME PIC X(20).
    02  FILLER PIC X(5) VALUE IS SPACES.
    02  CITY PIC X(20).
    02  FILLER PIC X VALUE IS SPACES.
    02  STATE PIC X(10).
    02  FILLER PIC X VALUE IS SPACES.
    02  ZIP PIC X(5).
    02  FILLER PIC X(49) VALUE IS SPACES.
PROCEDURE DIVISION.
PREPARATION-ROUTINE.
    OPEN INPUT CARDFILE
    OUTPUT PRINTED-REPORT.
HEADING-ROUTINE.
MOVE HEADING-RECORD-1 TO PRINT-RECORD.
    MOVE PRINT-RECORD.
    MOVE HEADING-RECORD-2 TO PRINT-RECORD.
    WRITE PRINT-RECORD AFTER ADVANCING 3 LINES.
MAIN-SEQUENCE.
    READ CARDFILE AT END GO TO FINISH.
    MOVE ID-CARD OF STUDENT-RECORD TO ID-CARD OF DETAIL-RECORD.
    MOVE STREET OF STUDENT-RECORD TO STREET OF DETAIL-RECORD.
    MOVE CITY OF STUDENT-RECORD TO CITY OF DETAIL-RECORD.
    MOVE STATE OF STUDENT-RECORD TO STATE OF DETAIL-RECORD.
    MOVE ZIP OF STUDENT-RECORD TO ZIP OF DETAIL-RECORD.
MOVE DETAIL-RECORD TO PRINT-RECORD.
WRITE PRINT=RECORD AFTER ADVANCING 3 LINES.
AT END-OF-PAGE GO TO HEADING-ROUTINE.
GO TO MAIN-SEQUENCE.
FINISH.
    CLOSE CARD-FILE PRINTED-REPORT.
    STOP RUN.
```

Figure 60

```
                        *       *       *
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       SPECIAL-NAMES.
           C01 IS NEW-PAGE.


       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT CARDFILE
               OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD-1
               TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING NEW-PAGE.
           MOVE HEADING-RECORD-2
               TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES.
       MAIN-SEQUENCE.
           READ CARDFILE
               AT END GO TO FINISH.
           MOVE ID-CARD OF STUDENT-RECORD
               TO ID-CARD OF DETAIL-RECORD.
           MOVE NAME OF STUDENT-RECORD
               TO NAME OF DETAIL-RECORD.
           MOVE STREET OF STUDENT-RECORD
               TO STREET OF DETAIL-RECORD.
           MOVE CITY OF STUDENT-RECORD
               TO CITY OF DETAIL-RECORD.
           MOVE STATE OF STUDENT-RECORD
               TO STATE OF DETAIL-RECORD.
           MOVE ZIP OF STUDENT-RECORD
               TO ZIP OF DETAIL-RECORD.
           MOVE DETAIL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 2 LINES
               AT END-OF-PAGE
                   GO TO HEADING-ROUTINE.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE CARDFILE PRINTED-REPORT.
           STOP RUN.
```

SUMMARY:

You have just completed Lesson 11 in which you learned to use the SPECIAL-NAMES paragraph option of the WRITE statement to provide vertical spacing for printed reports.

END OF LESSON 11

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 12

LESSON 12 - LIBRARY ENTRIES

INTRODUCTION

In this lesson you will learn how you may avoid recording programming routines used more than one time. The COPY statement will allow you to include entries in your source program from a library of source program entries. You will also apply new concepts from the last few lessons in practice problem coding.

Specific COBOL language features you will learn to use are:

COPY statement

This lesson will require approximately three quarters of an hour.

Now that you have learned to use the options of the WRITE statement to control vertical spacing, you can produce a listing of a card file in any format that may be requested. Another feature of the COBOL language can reduce the coding required to produce the listing, however. The card records illustrated in Figure 58 might be processed in several ways by several programs. The record description entries in the Data Division would be coded in each of the programs. Entries such as these that are used in many programs may be stored in a library of source coding. The coding segments are called books and are given distinct names by which they can be referred to in a COBOL program. Although you will not learn to create such a library in this course, you will learn to include source coding that is already part of a library in your program.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

COPY library-name

The COPY statement is used to include entries in a source program from a library of source coding. Locate the section titled COPY Statement under SOURCE PROGRAM LIBRARY FACILITY in the Language Specifications. The General Format chart shows the entries in which the COPY statement may be specified. Option 1 in the chart indicates that a valid use of the COPY statement could be:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

CONFIGURATION SECTION. COPY COMPUTERS.

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

OBJECT-COMPUTER. COPY INSTALLATION.

* * *

b
(a is incorrect because it is not possible to copy an entire section.)

A library name, like a program name, may be any combination of digits, letters, and hyphens with a maximum length of 30. The initial and final characters must not be hyphens. In addition to fitting these rules, library names must not be COBOL reserved words and the first five characters must not be duplicated in any other library name.

---

2. Refer to the portion of the General Format chart for the specified option for each entry below to determine whether the entry is a valid use of the COPY statement.

a. Option 2

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       INPUT-OUTPUT SECTION.
           COPY IN-OUT-SECT.
```

b. Option 3

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SELECT COPY FILES.
```

c. Option 4

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  CARD-FILE
               COPY FILE-DESCRIPTION-ENTRY.
```

d. Option 5

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  COPY CARD-RECORD.
```

e. Option 6

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       77  SAVE-NUMBER COPY LIBRARY-1.
```

f. Option 7

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       BEGIN. COPY OPEN-PARAGRAPH.
```

* * *

c,e,f
(a is incorrect because it is not possible to copy an entire section. In b the file name must be specified preceding the COPY statement. In d the data name must be specified preceding the COPY statement. The library name is used only in the COPY statement. The data name or file name is used in all other entries.)

---

3. The library texts shown in Figure 61 are to be incorporated into the program in Figure 60. Write the statements that would include the library texts.

1. Library name: TITLE-LINE
   Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  FILLER PIC X(37) VALUE IS SPACES.
           02  TITLE PIC X(25) VALUE IS
               'STUDENT LISTING SEM1 1972'.
           02  FILLER PIC X(59) VALUE IS SPACES.
```

2. Library name: COLUMN-HEADINGS
   Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  FILLER PIC X VALUE IS SPACES.
           02  HEADING-1 PIC X(6) VALUE IS 'NUMBER'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-2 PIC X(4) VALUE IS 'NAME'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-3 PIC X(6) VALUE IS 'STREET'.
           02  FILLER PIC X(20) VALUE IS SPACES.
           02  HEADING-4 PIC X(4) VALUE IS 'CITY'.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  HEADING-5 PIC X(5) VALUE IS 'STATE'.
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  HEADING-6 PIC X(3) VALUE IS 'ZIP'.
           02  FILLER PIC X(25) VALUE IS SPACES.
```

3. Library name: INFORMATION-LINE
   Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  FILLER PIC X VALUE IS SPACES.
           02  ID-CARD PIC X(5).
           02  FILLER PIC X(3) VALUE IS SPACES.
           02  NAME PIC X(20).
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  STREET PIC X(20).
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  CITY PIC X(20).
           02  FILLER PIC X VALUE IS SPACES.
           02  STATE PIC X(10).
           02  FILLER PIC X(2) VALUE IS SPACES.
           02  ZIP PIC X(5).
           02  FILLER PIC X(24) VALUE IS SPACES.
```

4. Library name: STANDARD-PARAGRAPH
   Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE HEADING-RECORD-1
               TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           MOVE HEADING-RECORD-2
               TO PRINT-RECORD.
           WRITE PRINT-RECORD AFTER ADVANCING 3 LINES.
```

Figure 61

* * *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  HEADING-RECORD-1 COPY TITLE-LINE.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  HEADING-RECORD-2
               COPY COLUMN-HEADINGS.
```

3)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  DETAIL-RECORD COPY INFORMATION-LINE.
```

4)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       HEADING-ROUTINE. COPY STANDARD-PARAGRAPH.
```

(These four COPY statements eliminate the necessity of writing 33 lines of coding.)

Use the IBM 1130 Disk Monitor System, Version 2 manual for the operating system in your computer installation. If you do not work in a computer installation, you may use the Language Specifications.

Consult the Language Specifications Manual which has excellent indexes and contains all the rules for coding the American National Standard COBOL. As a COBOL programmer you will use this as your primary reference source. Quickly review the manual now to familiarize yourself with its structure and content. Then continue by coding a solution for the problem described in the following frame.

---

4. In preparation for an advertising campaign, a major manufacturing firm has requested a listing of customer master records in the form shown in the Printer Spacing Chart in Figure 62. The master records are on cards in a form shown by the diagram of the input area. Since these master records and detail records are processed by several different programs, their record description entries are common to all of the programs. The record description entry for the master records exists as a library text called MASTER. The record description entry for the detail records exists as library text called DETAIL-LINE. The heading entries exist as library text called HEADINGS. The library texts are shown in Figure 62. The title, however, is unique and must be defined in this program. Code the program called LISTING to produce the listing of customer master records.

Library name: MASTER
Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  CHARGE-CARD PIC X(5).
           02  NAME PIC X(20).
           02  STREET PIC X(20).
           02  CITY PIC X(20).
           02  STATE PIC X(10).
           02  ZIP PIC X(5).
```

Library name: DETAIL-LINE
Text:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  FILLER PIC X.
           02  C-NUMBER PIC X(5).
           02  FILLER PIC XXXX VALUE IS SPACES.
           02  NAME PIC X(20).
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  STREET PIC X(20).
           02  FILLER PIC X(5) VALUE IS SPACES.
           02  CITY PIC X(20).
           02  FILLER PIC X VALUE IS SPACE.
           02  STATE PIC X(10).
           02  FILLER PIC XX VALUE IS SPACES.
           02  ZIP PIC X(5).
           02  FILLER PIC X(23) VALUE IS SPACES.
```

Library name: HEADINGS
Text

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  FILLER PIC X.
           02  HEADING-1 PIC X(17) VALUE IS
               'NUMBER           '.
           02  HEADING-2 PIC X(24) VALUE IS
               'NAME                     '.
           02  HEADING-3 PIC X(26) VALUE IS
               'STREET                       '.
           02  HEADING-4 PIC X(15) VALUE IS
               'CITY           '.
           02  HEADING-5 PIC X(11) VALUE IS
               'STATE      '.
           02  HEADING-6 PIC X(3) VALUE IS 'ZIP'.
           02  FILLER PIC X(24) VALUE IS SPACES.
```

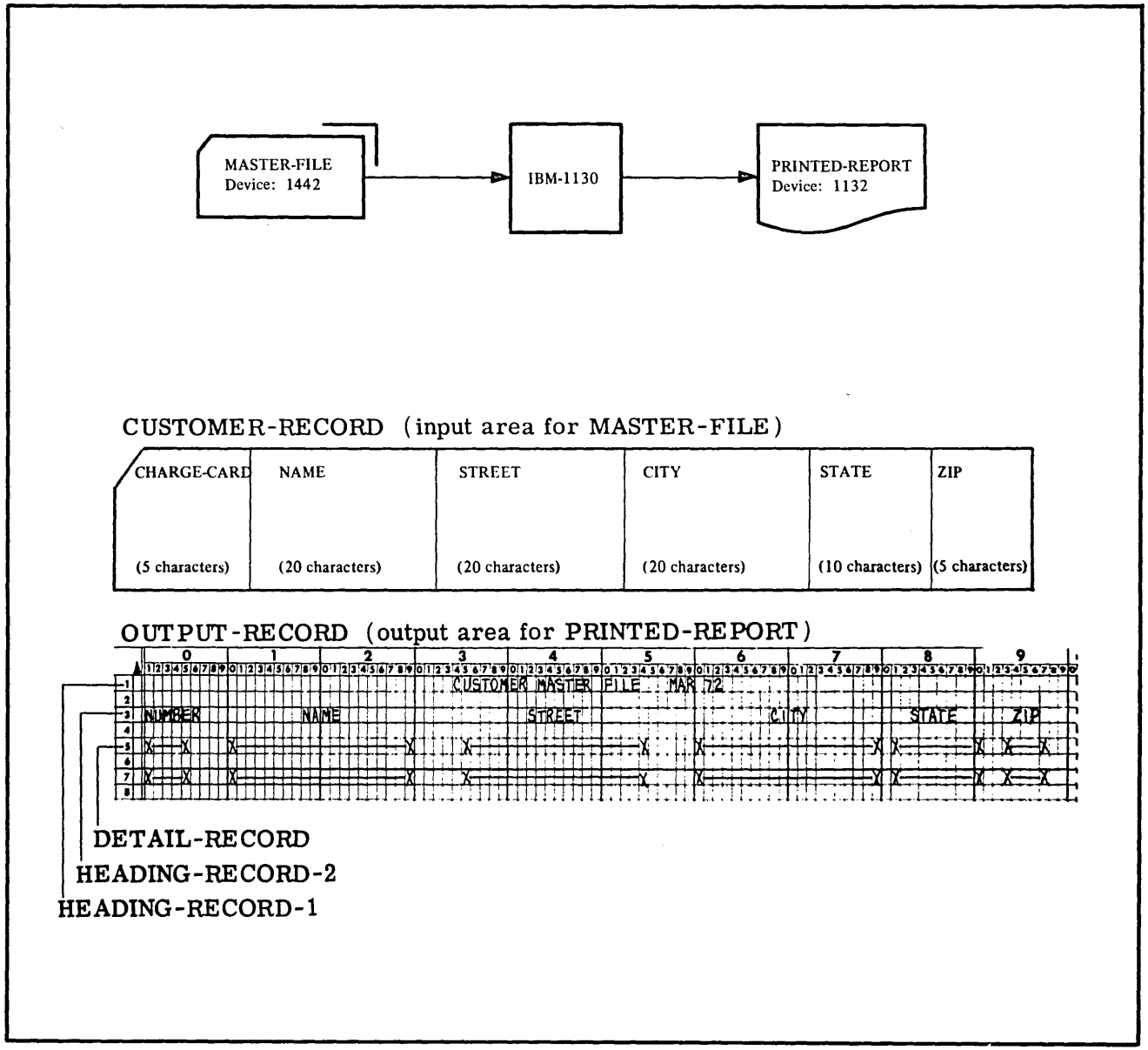

MASTER-FILE
Device: 1442
IBM-1130
PRINTED-REPORT
Device: 1132
CUSTOMER-RECORD (input area for MASTER-FILE)
CHARGE-CARD
NAME
STREET
CITY
STATE
ZIP
(5 characters)
(20 characters)
(20 characters)
(20 characters)
(10 characters)
(5 characters)
OUTPUT-RECORD (output area for PRINTED-REPORT)
CUSTOMER MASTER FILE MAR 72
NUMBER
NAME
STREET
CITY
STATE
ZIP
DETAIL-RECORD
HEADING-RECORD-2
HEADING-RECORD-1

Figure 62

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LISTING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE CONTROL.
           SELECT MASTER-FILE
               ASSIGN TO RD-1442.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C.
               RESERVE NO ALTERNATE AREAS.          (20)
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER-FILE
           LABEL RECORDS ARE OMITTED.
       01  CUSTOMER-RECORD COPY MASTER.              (40)
       FD  PRINTED-REPORT
           LABEL RECORDS ARE OMITTED.
       01  OUTPUT-RECORD.
           02  OUT-AREA PIC X(121).
       WORKING-STORAGE SECTION.
       01  HEADING-RECORD-1.
           02  FILLER PIC X(34) VALUE IS SPACES.
           02  HEADING-1 PIC X(29)
               VALUE IS 'CUSTOMER MASTER FILE   MAR 72'.
           02  FILLER PIC X(58) VALUE IS SPACES.
       01  HEADING-RECORD-2 COPY HEADINGS.           (40)
       01  DETAIL-RECORD COPY DETAIL-LINE.           (40)
       PROCEDURE DIVISION.
       PREPARATION-ROUTINE.
           OPEN INPUT MASTER-FILE
               OUTPUT PRINTED-REPORT.
       HEADING-ROUTINE.
           MOVE HEADING-RECORD-1
               TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD.
           MOVE HEADING-RECORD-2
               TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD
               AFTER ADVANCING 2.
       MAIN-SEQUENCE.
           READ MASTER-FILE
               AT END GO TO FINISH.
           MOVE CHARGE-CARD OF CUSTOMER-RECORD
               TO C-NUMBER OF DETAIL-RECORD.
           MOVE NAME OF CUSTOMER-RECORD
               TO NAME OF DETAIL-RECORD.
           MOVE STREET OF CUSTOMER-RECORD
               TO STREET OF DETAIL-RECORD.
           MOVE CITY OF CUSTOMER-RECORD
               TO CITY OF DETAIL-RECORD.
           MOVE STATE OF CUSTOMER-RECORD
               TO STATE OF DETAIL-RECORD.
           MOVE ZIP OF CUSTOMER-RECORD
               TO ZIP OF DETAIL-RECORD.
           MOVE DETAIL-RECORD TO OUTPUT-RECORD.
           WRITE OUTPUT-RECORD
               AFTER ADVANCING 2
               AT END-OF-PAGE                       (11)
                   GO TO HEADING-ROUTINE.
           GO TO MAIN-SEQUENCE.
```

```
FINISH.
    CLOSE MASTER-FILE
        PRINTED-REPORT.
    STOP RUN.
```

SUMMARY:

You have now completed Lesson 12 in which you have learned to include text from a library of source coding in your source program.

END OF LESSON 12

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 13

LESSON 13 - SEQUENTIAL DISK FILE OUTPUT

INTRODUCTION

In the problems you have coded in previous lessons you have used only card files and printer files. In this lesson you will use disk files which have the advantages of higher input and output speeds and more compact storage of data. You will learn how records in a disk file can be blocked to provide even more compact storage. In addition to learning to create a master file on disk you will learn to specify a single statement to read and move a record as well as to specify arithmetic operations. To begin, you will review some of the basic physical concepts concerning disk storage, and principles pertaining to file organization.

Specific COBOL language features you will learn to use in this lesson are:

ASSIGN clause for disk files
STANDARD option of the LABEL RECORDS clause
BLOCK CONTAINS clause with CHARACTER option
INTO option of the READ statement
FILE LIMIT clause

This lesson will require approximately one hour.

1.

DISK STORAGE

Disk storage provides the 1130 system with low-cost random or sequential access data storage. Disk storage for the 1130 is divided into two separate entities, each dependent upon the other. The disk storage drive provides the access mechanism, the magnetic read/write heads, and the mechanical and electronic components necessary to record and retrieve data. The disk cartridge provides a storage medium.

The disk ..... is the storage medium that is mounted in the disk ....

* * *

cartridge
storage drive

---

2. The storage capacity provided to the 1130 system by the disk storage is 512,000 words per storage drive. A total on-line capacity up to 2,560,000 words is provided. Off-line capacity is virtually unlimited because the interchangeable disk cartridge is easily removed and replaced with another.

   The storage capacity limit of the 1130 disk storage is approximately 2 and a half million words.

   a. True

   b. False

* * *

b. Only on-line capacity is limited. Disk cartridges are removable.

---

3. The disk cartridge (Figure 63) is a single disk completely enclosed in protective housing. The recording medium is an oxide-coated disk that provides two surfaces for the magnetic recording of data. When mounted on a storage drive, the disk rotates at 1,500 revolutions per minute.

   Match the most closely related items in these two columns:

1.	Disk cartridge	a.	Magnetically recorded
2.	Data	b.	Contains 2 surfaces
3.	Disk rotation	c.	Can be read off-line
		d.	1,500 revolutions per minute.

* * *

1. b
2. a
3. d

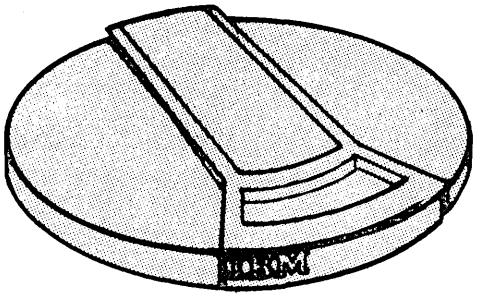


Figure 63

---

4. The disk storage access mechanism has two horizontal arms. Each arm has a magnetic read/write head, and each head is positioned to read or write on the corresponding disk surface as the access arms straddle the disk in the manner of a large tuning fork. The entire assembly moves horizontally forward and backward so that the heads have access to the entire recording area.

   The magnetic head can ..... or ..... on the disk. Access to all data is possible because the horizontal arms ..... .

* * *

read
write
move

---

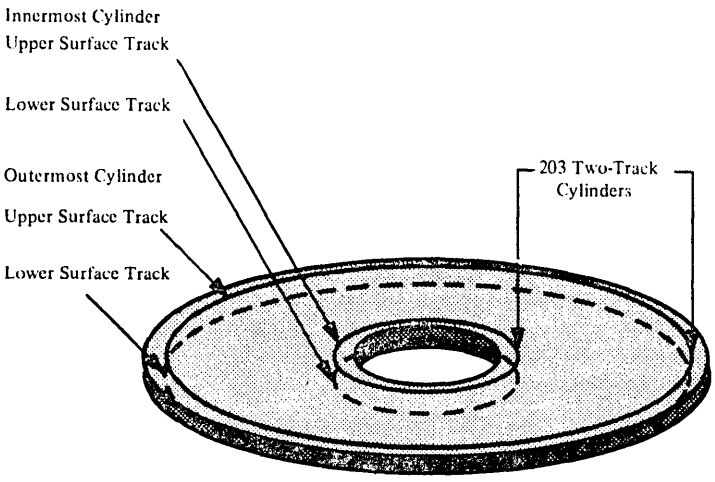
5. The disk access mechanism is moved back and forth by programmed commands and can be placed in any one of 203 positions, from a point near the periphery of the disk to a point near the center of the disk. At each position, the heads can read or write in a circular pattern on both surfaces of the disk, as it revolves. The circular patterns of data are called tracks. The track on the upper surface of the disk and the corresponding track on the lower surface, both of which can be read or written while the access mechanism is in the same posistion, are called a cylinder. Figure 64 shows the innermost and outermost cylinders of two tracks each. To complete the picture, the 201 intermediate cylinders, or pairs of tracks, should be visualized; they were omitted for the sake of clarity of the diagram.

   Match the items that correspond best:

1. Programmed command	a. Control access mechanism motion
2. Tracks	b. Corresponding tracks, upper and lower surface
3. Cylinder	c. Circular pattern of data
	d. Round-shaped disk housing

* * *

1. a
2. c
3. b



NOTE: The thickness of the disk has been greatly exaggerated to show the relative positions of the upper and lower surface tracks.

Figure 64

6. For convenience in transferring data between the CPU core storage and disk storage, each track is divided into four equal segments called sectors. Sectors are numbered by the cylinder, from 0 through 7, as shown in Figure 65. Sectors 0 - 3 divide the upper surface track, and sectors 4 - 7, the lower. A sector contains 321 data words and is the largest segment of data that can be read or written with a single instruction. The first word of the sector cannot be used by the programmer if the assembler program or other components of the monitor system are to be used.

   How many words can be read or written with a single instruction?

* * *

321, of which 320 are available to the programmer.

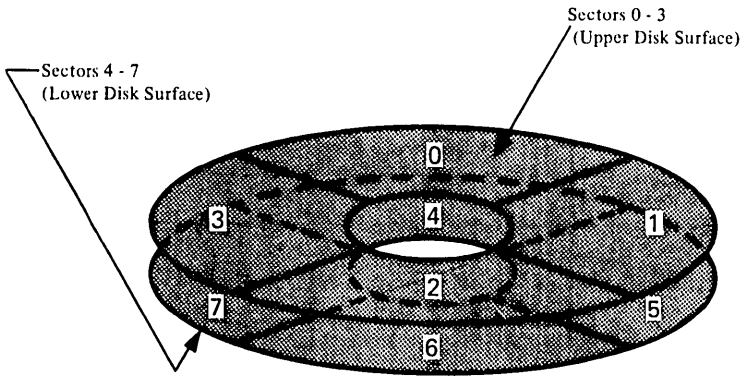


Figure 65

---

7. A disk storage word comprises 16 data bits and four check and space bits. Figure 66 shows the organizational components of disk storage. Note that capacities are based on the 320 word sector; also the number of cylinders is 200 rather than 203. Three cylinders (24 sectors) are provided as alternates to be used if a surface is defective.

   Match the columns:

1. Number of effective data bits per disk storage word	a. 203 b. 16 c. 20
2. Number of effective words per sector	d. 320 e. 321 f. 200
3. Number of effective cylinders	

* * *

1. b
2. d
3. f

No. of / Per	Word	Sector	Track	Cylinder	Disk
Bits	16	5,120	20,480	40,960	8,192,000
Data Words		320	1,280	2,560	512,000
Sectors			4	8	1,600
Tracks				2	400
Cylinders					200

Figure 66

8. Timing considerations of disk storage operations involve three elements: access time, reading and writing data, and the time during which the cpu is tied up. Once a seek, read, or write instruction is initiated, disk storage operation is virtually independent of the CPU (Central Processing Unit).

   What are the three elements involved in timing disk write operation?

* * *

access time
writing data
time during which the CPU is tied up

FILE ORGANIZATION

There are two types of disk-file organization in 1130 COBOL: sequential and random.

In a sequential disk-file, the records are written and read one after the other. Record four, for example, can not be read or written unless record three is read or written first, and so on.

In a random disk-file records are written or read without respect to their location. For example, record seven can be read first, then record one, and so on.

The disk storage devices will be discussed extensively later. In earlier lessons the disk devices will be used in a sequential mode.

---

9. Records in a file must be logically organized so that they can be retrieved efficiently for processing. Some factors to be considered in selecting a method of organization are file volatility, activity and size. Name three factors influencing the choice of a method of file organization.

* * *

File volatility, activity and size.

---

10. Volatility. A static file is one that has a low percentage of additions and deletions, while a volatile file is one that has a high rate of additions and deletions.

    Activity. If a low percentage of the records are to be processed on a run, the file should probably be organized in such a way that any record can be quickly located without having to look at all the records in the file. The distribution of the activity is also a consideration. The records processed most frequently should certainly be the ones that can be located most quickly. An active file must be organized very carefully, since the time involved in locating the records may amount to an appreciable period of time. At the other extreme, an inactive file may be referred to so infrequently that the time required to locate the records is immaterial.

    Size. A file so large that it cannot be all online (available to the system) at one time must be organized and processed in certain ways. A file may be so small that the method of organization makes little difference, since the time required to process it is very short no matter how it is organized. Usually, files are planned on the basis of their anticipated growth over a period of time.

    A ..... file has a high rate of changes. Active records must be accessed ..... . Files are planned on the basis of future ..... .

* * *

volatile
quickly
growth

---

11. The distinction between the organization of a master file and the order of the input detail records processed against the file is important. In sequential processing, the input transactions are grouped together, sorted into the same sequence as the master file, and the resulting batch is then processed against the master file. When cards are used to store the master files, sequential processing is the most efficient means of processing. Direct access storage devices are very efficient sequential processors, especially when the percentage of activity against the master file is high.

    In .... input transactions are grouped together (batched) and sorted into the same sequence as the ....

* * *

sequential processing
master file

---

12. Random processing is the processing of detail transactions against a master file in whatever order they occur. With direct access devices, random processing can be very efficient, since a file can be organized in such a way that any record can be quickly located.

    It is possible, on a run, to process the input transactions against more than one file. This saves setup and sorting time. It is not necessary to wait until a batch of transactions has been accumulated to make processing worthwhile. The transactions can be processed inline - that is, as soon as they are available.

    Match these columns:

1. Random processing	a. Arranging data in a given sequence.
2. Direct access	b. Retrieval of a given record without sequential search
3. Sorting	c. Transactions processed without regard to sequence
4. Batching	d. Accumulation of transactions into a group

* * *

1. c
2. b
3. a
4. d

IBM 1130 COBOL directly supports a sequential and a random method of disk data organization. The indexed sequential method may be used, but the programmer must develop techniques for file creation and accessing. Further explanation will be given later in this course.

---

13. Sequential organization. In a sequential file, records are organized solely on the basis of their successive physical location in the file. The records are usually read or updated in the same order in which they appear. For example, the hundredth record is usually read only after the first 99 have been read.

    Individual records cannot be located quickly. Records usually cannot be deleted or added unless the entire file is rewritten. This organization is generally used when most records are processed each time the file is used.

    Sequential file organization would be recommended to accomodate a reservation system, where a relatively few requests are processed against a large file.

    a. True

    b. False

* * *

b. Sequential file organization is not practical unless an appreciable number of records are active each time the file is used.

---

14. Indexed Sequential Organization. An indexed sequential file is similar to a sequential file in that rapid sequential processing is possible. Indexed sequential organization, however, by reference to indexes associated with the file, makes it also possible to quickly locate individual records for random processing. Moreover, a separate area of the file is set aside for additions; this obviates a rewrite of the entire file, a process that would usually be necessary when adding records to a sequential file. Although the added records are not physically in key sequence, the indexes are referred to in order to retrieve the added records in key sequence, thus making rapid sequential processing possible.

    Name three advantages of the indexed sequential file organization method.

* * *

Rapid sequential processing is possible.
Random processing is also possible.
The entire file does not have to be rewritten to make additions.

---

15. Random Organization. A file organized in a random manner is characterized by some predictable relationship between a key of a record and the location of that record on a DASD. This relationship is established by the user. This organization method is generally used for files whose characteristics do not permit the use of sequential or indexed sequential organizations, or for files where the time required to locate individual records must be kept to an absolute minimum. This method has considerable flexibility. The accompanying disadvantage is that the programmer is largely responsible for the logic and programming required to locate the records, since he establishes the relative addresses of records on the DASD.

    When is the random organization sometimes used in preference to sequential or indexed sequential?

* * *

It is used when the time required to locate individual records must be kept to a minimum.

---

16. The programmer, in computing the space needed to contain his program and data, must first determine the number of words per logical record. A logical record, as distinguished from a physical record, is a given amount of data associated with one particular transaction in a computer application e.g; the information pertaining to an item in inventory. A physical record is always 320 words, since this is the length of a disk sector.

    A ..... record is related data being processed in a computer application. A ..... record is determined by sector size.

* * *

logical
physical

---

17. Disk files can be blocked, but because the size of a disk sector is fixed at 320 words, the compiler and the execution-time input-output routines assume that the number of records contained in a disk block is exactly 320 divided by the length of the logical record, with any remainder discarded.

Once the blocking factor is known, the number of sectors, cylinders and disk cartridges needed may each be determined in succession as is shown here:

5000 RECORDS @ 32 WORDS/RECORD

320/32 = 10 = BLOCKING FACTOR

```
                    500 SECTORS
                          500 SECTORS
                    ------------------
10 RECORDS/SECTOR  ) 5000 RECORDS

                          62.5 CYLINDERS
                    ------------------
8 SECTORS/CYLINDER ) 500 SECTORS
```

SINCE DATA FILES ALWAYS REQUIRE AN INTEGER NUMBER OF SECTORS, THIS FILE WILL REQUIRE 62.5 CYLINDERS.

SINCE A DISK CARTRIDGE CONTAINS 200 CYLINDERS, THIS FILE REQUIRES

62.5/200 OR 31.3% OF A CARTRIDGE

1. What would be the blocking factor of records 20 words each in length?

2. How many sectors would be needed to hold 3,000 records? How many cylinders?

* * *

1. Blocking factor: 16
   320/20 = 16

2. Number of sectors needed: 188
   3000/16 = 187, with a remainder of 8,or,
   in effect, 188.

3. Number of cylinders needed: 23.5
   188/8 = 23.5

---

18. Spanned records are permitted. A spanned record is a logical record whose length exceeds the fixed length of a physical record. Such records are permitted only on disk - where the physical record size is 320 words - and may be processed sequentially or randomly. Any record whose size is greater than 320 is considered a spanned record. Such records always occupy an integral number of physical disk sectors; each such record always begins in a new sector, and no special length indicator is imbedded in the record.

    Why should spanned records be avoided if possible?

* * *

Since each record must start in a new sector, those sectors containing the ends of records will usually not be fully utilized.

---

19. The 1130 COBOL compiler will allocate a record area in main storage the size of the largest (or only) record, as specified by the record description entries. On the external medium, all records are carried as fixed length logical records; each record is the size defined for the largest record in the file. For a disk file, the maximum size which may be defined for a logical record is 4095. If any record defined for a disk file is specified as greater than 320 words in size, the file is considered to consist of spanned records.

    1130 COBOL records are ..... in length. Each record is the size defined for the ..... record in the file.

* * *

fixed
largest

---

20. When a disk file is to be used in a COBOL program, the file must be named in a SELECT clause and assigned to an input/output device in an ASSIGN clause. For a disk file, the form of system-name is something different. Three facts must be specified in the name. (See Figure 20.)

    1) The file number of the file (to be equated with an actual file by means of an *FILES supervisor control record at XEQ time).

    2) The number of record slots (to be) allocated for the file on disk.

    3) Whether the file is to use a shared disk buffer during execution, or its own unique disk buffer.

    The form of system-name for a disk file is:

        DF-filenumber-numberofrecords[-X]

    where

        filenumber is the number of the file to be equated at XEQ time; the number must be in the range 1 through 32767 and be written without preceding zeros

        numberofrecords is the number of record slots (to be) allocated for the file; it must be a number in the range 1 through 32767, written without preceding zeros

        -X specifies that the file is to have its own unique disk buffer; if -X is not specified, the shared disk buffer will be utilized

---

21. Write a SELECT and ASSIGN clause for the disk file. Use 600 as the number of records, specifying that the file is to have its own buffer area. Use Figure 21 as a guide.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SELECT DISKFILE ASSIGN TO DF-1-600-X.
```

---

22. The elements of a system name are as follows:

    The DF stands for disk file and it is standard in all disk system names.

    The next element is the symbolic file number which is a number from 1 to 32767. The programmer may choose any of these numbers. The number must be unique for each disk file. The third element is the length. That is the number of records the programmer wishes to have on the file. If the file consists of 500 records write 500. If the file consists of 900 records write 900, etc.

    The last element is a -X. This indicates that the file will have its own buffer. When the -X is not specified, only one buffer is created for all disk files. When a READ or WRITE statement is issued, a whole disk sector is read or written. A sector may contain more than one record. When a READ is issued from disk number 1 and later another READ is issued from disk number 2 and the -X has not been specified in the system name, reading will be done into one buffer from both disks. The second READ statement will destroy what was read by the first. When another record is required later from the first disk, the sector has to be brought back into core, whereas if the file has its own buffer all records of each sector will be processed before another sector reading or writing is initiated. Each buffer is 320 words. In a large program these words may not be spared. Therefore, the programmer may have to generate only one buffer.

```
          symbolic file
DF -      number          - Length [-X]
```

    The symbolic file number must be the same in the system name and *FILES control card.

    EXAMPLE:

    DF-10-700-X

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

*FILES(10,PAYRL)

Control Card

---

23. A disk file must have an FD entry in the Data Division. The requirements for an FD entry for a disk file are the same as for a card file. An FD entry for a disk file must:

    a. contain the LABEL RECORDS clause.

    b. be preceded by a level number.

                    *   *   *

a

Because card and printer files cannot contain records to label the file, the OMITTED option of the LABEL RECORDS clause was required for the files used in programs in previous lessons. Since it is possible to write on disk on which data has already been recorded, thus destroying the original data, standard records are often included to label the file and protect the data. If standard records are included, they are checked automatically at the time the file is opened. Usually, it is not important for the programmer to know the details of label format.

---

24. In order to prevent data on a disk from being accidentally destroyed, standard records used to label the file may be included in the disk file. Inclusion of standard records is specified in the LABEL RECORDS clause. You might expect that the LABEL RECORDS clause for a disk file with standard records would be:

    a. LABEL RECORDS ARE OMITTED.

    b. LABEL RECORDS ARE STANDARD.

                    *   *   *

b (Standard labels are IBM-supplied labels)

---

25. Write an FD entry for the file MASTERDISK, which has standard records used to label the file that are to be checked by the computer.

                    *   *   *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
FD  MASTERDISK
    LABEL RECORDS ARE STANDARD.
```

The data records in a disk file can be blocked, thus conserving space on the disk. When an input file contains blocked records, the first execution of a READ statement for that file causes physical transmission of a block from the file and the first record in the block is available for processing. Each subsequent execution of a READ statement causes each subsequent record to become available until all the records in the block have been processed. Then the next READ statement executed will cause physical transmission of another block and the processing described previously will be repeated. Similarly, if the records of an output disk are to be blocked, the records are transmitted to the output file as a block.

26.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
FD  MASTERDISK
    BLOCK CONTAINS 5 RECORDS
    LABEL RECORDS ARE STANDARD.
```

Because records are transmitted as a block, the computer must know how many records are in a block. The BLOCK CONTAINS clause in the FD entry is used to specify the number of records in a block. The FD entry above specifies that records are to be transmitted in a block of ..... records.

* * *

five
(The record description entry that follows an FD entry specifies a size of a record. The block size, in turn, can be determined by multiplying the size of a record by the number of records in a block.)

---

27. Write the FD entry for the disk file CUSTOMER-FILE. The records are blocked into groups of four. Standard records are included to label the file.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
FD  CUSTOMER-FILE
    BLOCK CONTAINS 4 RECORDS
    LABEL RECORDS ARE STANDARD.
```

(The order of the BLOCK and LABEL clauses is not important, but only the last clause is followed by a period.)

---

28. Unlike records in a card file, records in a disk file do not have to be a set length such as 80 characters. The records can be any length you desire. If each record in a disk file contains two data items, and each data item is 10 characters long:

    a. each record will be 20 characters long.

    b. you should account for another 60 characters in the record description entry.

* * *

a

29.

PART-RECORD				
PART-NUMBER (6 char)	STOCK-INDEX (5 char)	UNITS-ON-HAND (4 digits)	UNIT-PRICE (4 digits)	TOTAL VALUE (6 digits)

Write the FD and record description entries for the disk file PART-FILE. The records, which are illustrated above, are to be blocked into groups of eight. Standard labels are to be included.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  PART-FILE
           BLOCK CONTAINS 8 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  PART-RECORD.
           02  PART-NUMBER PIC X(6).
           02  STOCK-INDEX PIC X(5).
           02  UNITS-ON-HAND PIC 9(4).
           02  UNIT-PRICE PIC 9(4).
           02  TOTAL-VALUE PIC 9(6).
```

In the previous lessons you have used only card files and printer files. Now that you have learned to describe disk files in your COBOL program, you can use them as you have used card and printer files.

---

30. Write two COBOL statements to cause a record in PART-FILE to be read into the associated input area described in the preceding frame and the entire record to be made available in the working-storage variable PART-RECORD-2.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ PART-FILE.
           MOVE PART-RECORD TO PART-RECORD-2.
```

31.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           READ PART-FILE INTO PART-RECORD-2.
```

The INTO option can be used in the READ statement to replace the two statements you wrote in frame 11, thus reducing the number of statements in the source program. The statement above has the same effect as the READ and MOVE statements.

a. a READ statement with the INTO option has the same effect as a READ statement followed by a simple MOVE statement.

b. after the READ statement above is executed, the record from PART-FILE will be available in both the input area and in working storage.

* * *

Both

32.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        DATA DIVISION.
        FILE SECTION.
        FD  INPUT-FILE
            LABEL RECORDS ARE OMITTED.
        01  ITEM-RECORD.
            02  ITEM-NUMBER PIC X(10).
            02  ITEM-NAME PIC X(20).
            02  ITEM-DESCRIPTION PIC X(50).
        FD  OUTPUT-FILE
            BLOCK CONTAINS 5 RECORDS
            LABEL RECORDS ARE STANDARD.
        01  ITEM-RECORD-O PIC X(80).
        FD  COMPLETE-FILE
            BLOCK CONTAINS 4 RECORDS
            LABEL RECORDS ARE STANDARD.
        01  ITEM-1.
            02  ITEM.
                03  ITEMNUMBER PIC X(10).
                03  ITEMNAME PIC X(20).
                03  ITEMDESCRIPTION PIC X(50).
            02  ITEM-INDEX PIC 9(5).
        WORKING-STORAGE SECTION.
        77  ITEM-2 PIC X(80).
        01  ITEM-3.
            02  DESCRIPTION PIC X(80).
            02  ITEM-VALUE PIC 9(5).
```

The READ statement with the INTO option has the form:

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

            READ file-name INTO variable-name.
```

The variable name specified in the INTO option can be a name in the data description entry in the Working-Storage Section or in an output record description entry in the File Section (if the associated output file has been opened prior to execution of the READ statement.) Refer to the preceding Data Division and determine which of the following statements is valid:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO ITEM-RECORD-O.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO ITEM-1.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO ITEM.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO ITEM-2.
```

e.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO ITEM-3.
```

f.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO DESCRIPTION.
```

* * *

All of these (When statement b and e are executed, the leftmost 80 positions of the receiving field will be filled with characters and the remaining positions will be padded with blanks, according to the rules for a simple MOVE statement specifying group variables.)

33.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  INDEXFILE
           LABEL RECORDS ARE OMITTED.
       01  INDEX-RECORD.
           02  ID-NUMBER PIC X(6).
           02  OTHER-DATA PIC 9(15).
       WORKING-STORAGE SECTION.
       01  WORK-AREA.
           02  I-D PIC X(6).
           02  INDEX-FACTOR PIC 9(5).
           02  MULTIPLES PIC 9(5).
           02  VALUE-INDEX PIC X(5).
```

Refer to the portion of the Data Division shown above and code one statement that would produce the same result as the two statements below.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INDEXFILE AT END GO TO REPEAT.
           MOVE INDEX-RECORD TO WORK-AREA.

                              *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INDEXFILE INTO WORK-AREA
               AT END GO TO REPEAT.
```

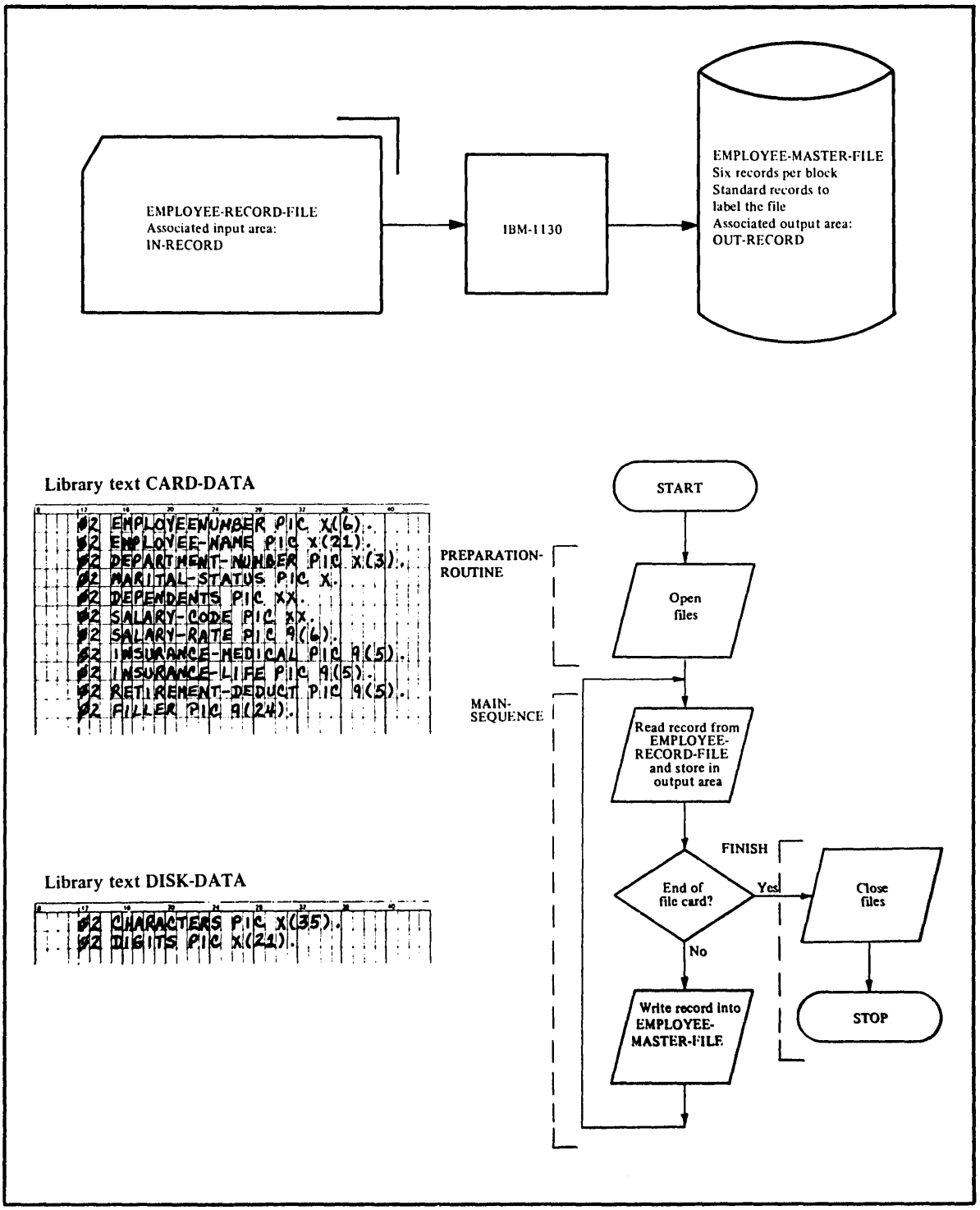

EMPLOYEE-RECORD-FILE
Associated input area:
IN-RECORD
IBM-1130
EMPLOYEE-MASTER-FILE
Six records per block
Standard records to
label the file
Associated output area:
OUT-RECORD
Library text CARD-DATA
02 EMPLOYEENUMBER PIC X(6).
02 EMPLOYEE-NAME PIC X(21).
02 DEPARTMENT-NUMBER PIC X(3).
02 MARITAL-STATUS PIC X.
02 DEPENDENTS PIC XX.
02 SALARY-CODE PIC XX.
02 SALARY-RATE PIC 9(6).
02 INSURANCE-MEDICAL PIC 9(5).
02 INSURANCE-LIFE PIC 9(5).
02 RETIREMENT-DEDUCT PIC 9(5).
02 FILLER PIC 9(24).
Library text DISK-DATA
02 CHARACTERS PIC X(35).
02 DIGITS PIC X(21).
START
PREPARATION-ROUTINE
Open files
MAIN-SEQUENCE
Read record from EMPLOYEE-RECORD-FILE and store in output area
FINISH
End of file card?
Yes
Close files
No
Write record into EMPLOYEE-MASTER-FILE
STOP

Figure 67

34. The program EMPLOYEE-MASTER-PROGRAM is to be used to transfer a master file from cards to disk. The system flow chart and a program flow chart are shown in Figure 67 along with the two library texts CARD-DATA and DISK-DATA which are to be incorporated into the source program for the record description entries for IN-RECORD and OUT-RECORD, respectively. Use Figure 67 as a guide for coding the missing portions of the program shown below.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMPLOYEE-MASTER-PROGRAM.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE CONTROL.

 1)

       DATA DIVISION.
       FILE SECTION.
       FD  EMPLOYEE-RECORD-FILE
           LABEL RECORDS ARE OMITTED.
 2)
       FD  EMPLOYEE-MASTER-FILE

 3)

       PROCEDURE DIVISION.


 4)
```

* * *

1)

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SELECT EMPLOYEE-RECORD-FILE
               ASSIGN TO RD-1442.
           SELECT EMPLOYEE-MASTER-FILE
               ASSIGN TO DF-1-800-X.                (1)
```

(Recall: the figure "800" in the ASSIGN clause is a figure denoting the number of records in EMPLOYEE-MASTER-FILE.)

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  IN-RECORD COPY CARD-DATA.
```

3)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           LABEL RECORDS ARE STANDARD                (4)
           BLOCK CONTAINS 6 RECORDS.                 (6)
       01  OUT-RECORD COPY DISK-DATA.
```

4)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PREPARATION ROUTINE.
           OPEN INPUT EMPLOYEE-RECORD-FILE
                OUTPUT EMPLOYEE-MASTER-FILE.
       MAIN-SEQUENCE.
           READ EMPLOYEE-RECORD-FILE                 (11)
               INTO OUT-RECORD
               AT END GO TO FINISH.
           WRITE OUT-RECORD.
           GO TO MAIN-SEQUENCE.
       FINISH.
           CLOSE EMPLOYEE-RECORD-FILE
               EMPLOYEE-MASTER-FILE.
           STOP RUN.
```

(Moving a record from the input file to the shorter output area resulted in truncation of the rightmost 24 characters in the input record. This caused no problem since the truncated characters were not to be included in the output record.)

You have now learned to code the entries to write records into a file on disk as well as to read records from a card file and to write records into a printer file in any specified format.

---

35. The FILE-LIMIT clause serves only as documentation. This clause need not be specified in the FILE-CONTROL Section and if specified it will be treated as comments. If the clause is used it specifies the beginning and the end of a logical file on a mass-storage device.

```
FILE-LIMIT IS
                        integer-1 thru integer-2
FILE-LIMITS ARE
```

The PROCESSING MODE clause also serves only as documentation and indicates the order in which records are processed. The following example demonstrates the FILE-LIMIT and PROCESSING MODE clauses.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FILE CONTROL.
           SELECT DISK-FILE
               ASSIGN TO DF-1-600-X
               ACCESS IS SEQUENTIAL
               FILE-LIMITS ARE 100 THRU 600
               PROCESSING MODE IS SEQUENTIAL.
```

---

36. BLOCK CONTAINS clause is used to specify the size of a physical record. If the CHARACTER option is used, the number of CHARACTERS is the same as the number of words.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  NEW-MASTER.
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 100 CHARACTERS.
```

The VALUE OF clause uniquely specifies the description of an item in the label records associated with a file and serves only as documentation.

The DATA RECORDS clause serves only as documentation and identifies the records in the file by name.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       FD  NEW-MASTER.
           BLOCK CONTAINS 15 RECORDS
           LABEL RECORDS ARE STANDARD
           VALUE OF DISK-FILE IS PAYROLL
           DATA RECORD IS PAY-MASTER.
```

SUMMARY:

You have now completed lesson 13 in which you have learned to code the necessary entries to create a disk file with standard records to label the file and blocked data records to provide compact storage. You have also learned to specify a single statement to read and move a record.

END OF LESSON 13

LESSON 14

LESSON 14 - SEQUENTIAL DISK FILE; ARITHMETIC OPERATIONS

INTRODUCTION

In this lesson you will learn to use the MULTIPLY and ADD statements with GIVING, ROUNDED, and SIZE ERROR options in both cases. You will also learn how to use the PICTURE character V to represent the implied decimal point in data manipulation.

This lesson will require approximately one hour.

1. As a programmer you might be required to write a program using the file created in the preceding frame to compute payroll data. Assuming that the base pay rate is an hourly rate, earnings would be computed by:

   a. multiplying the number of hours worked by the base pay rate.

   b. adding the hourly rate to the hours worked.

* * *

a

2.

MULTIPLY { variable-name-1 / numeric-literal-1 } BY variable-name-2

The MULTIPLY statement of the form shown above is used to specify multiplication. Use the form of the MULTIPLY statement shown above to select the valid MULTIPLY statements from those shown below:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY AMOUNT BY 5.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY 5 BY AMOUNT.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY 5 BY 5.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY FACTOR BY AMOUNT.
```

e.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY 'FACTOR' BY AMOUNT.
```

* * *

b,d

3.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MULTIPLY FACTOR BY AMOUNT.
```

When the statement above is executed, the product of the value stored in AMOUNT and the value stored in FACTOR is computed and then stored in AMOUNT.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MULTIPLY AMOUNT BY FACTOR.
```

When the statement above is executed:

a. the product of the value stored in AMOUNT and the value stored in FACTOR is computed.

b. the product is stored in AMOUNT.

* * *

a
(The product is stored in FACTOR.)

---

4. Write a statement to compute the product of SALARY and 1.04 and to store the product in SALARY.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MULTIPLY 1.04 BY SALARY.
```

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY X BY Y GIVING Z.

Often you may want to compute the product of two values and store the product in a third variable. In this case, the GIVING option can be included in the MULTIPLY statement. When the statement above is executed, the product of the values in X and Y is computed and stored in Z. (The values in X and Y remain unchanged.)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY RATE BY HOURS GIVING PAY.

When the statement above is executed, the product of the values in ..... and ..... is computed and stored in ..... .

* * *

RATE
HOURS
PAY

---

6. Write a statement to compute the product of the values in COST and TAX-RATE and to store the product in TAX.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY COST BY TAX-RATE
GIVING TAX.

---

7. Write a statement to compute the product of 2 and the value in TOTAL and to store it in TOTAL.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY 2 BY TOTAL.

---

8. All variables used in arithmetic computations must be elementary numeric variables. If a variable is to be specified in a MULTIPLY statement it must be defined with a picture of:

   a. X's

   b. 9's

   c. A's

* * *

b

9.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MULTIPLY QUANTITY BY AMOUNT
               GIVING TOTAL.
```

Write the data description entries for level 77 variables QUANTITY and AMOUNT necessary for execution of the statement above. Both variables will contain values up to five digits.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       77  QUANTITY PIC 9(5).
       77  AMOUNT PIC 9(5).
```

---

10. When a numeric value is recorded without a decimal point, it may be necessary to specify the number of decimal places by inserting a V into the picture for the variable that is to have that value. The picture 9999V99 for a variable specifies that values of the variable will have two decimal places. The value of the variable is said to have an assumed decimal point. Code the picture for a variable whose values are eight digit numbers with two decimal places and an assumed decimal point. The actual decimal point in a value is aligned with the implied decimal point in a receiving core area governed by a picture clause containing an implied point. As a result of moving such a value, the value will be truncated left or right, or padded with zeros to exactly fill the receiving core area.

* * *

999999V99

---

11. Any value or picture that contains no other assumed (or actual) decimal point is assumed to have a decimal point following the rightmost digit or digit position. Numeric data is automatically aligned in a variable beginning at the decimal point; padding with zeros or truncation of excess digits can occur at either or both ends of the value. The table below illustrates padding and truncation that occurs when certain values are assigned to variables with certain pictures. (The symbol / is used here and in future frames to represent an implied decimal position.)

VALUE ASSIGNED	PICTURE	RESULT
2/1	9V9	21 (no padding or truncation)
2/1	99V99	0210 (padding on right and left)
2/1	999	002 (padding on left; truncation on right)
2/1	V999	100 (padding on right; truncation on left)
21	99V99	2100 (padding on right)

If the value 9/876 were read into a variable with the picture 999V999, the resulting value would be:

a. 987600 (padded on the right with zeros with an assumed decimal point preceding the 6).

b. bb9876 (padded on the left with blanks with the assumed decimal point preceding the 8).

* * *

Neither (009876 padded on the left with zeros with the assumed decimal point preceding the 8.)

12.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           .
           .
           .
       WORKING-STORAGE SECTION.
       77  PRICE PIC 999V99.
       77  DISCOUNT PIC 9V99.
       77  PROFIT PIC 99V9.
       77  TOTAL PIC 9999V99.
           .
           .
           .
           MULTIPLY DISCOUNT BY PRICE.
           .
           .
           .
```

The values 20/98 and 5/63 have been read into the variables PRICE and DISCOUNT, which are defined in the Working-Storage Section shown above. When the MULTIPLY statement is executed, the product of the values of DISCOUNT and PRICE is computed as 118/1174. The picture for PRICE, in which the product will be stored, specifies that the value of PRICE is to have two decimal places to the right of an assumed decimal point. Consequently, the last two decimal places in the product are truncated and the value stored in PRICE is 118/11.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY PROFIT BY TOTAL.
```

If PROFIT and TOTAL have the values 11/1 and 11/11 respectively, the value stored in TOTAL after execution of the MULTIPLY statement above is:

a. 123/321

b. 1233/21

* * *

Neither (The product is 123/321, the rightmost decimal digit is truncated or lost, and the value stored in TOTAL is 0123/32.)

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY DISCOUNT BY PRICE ROUNDED.

A product can be rounded rather than merely truncated by specifying the ROUNDED option in the MULTIPLY statement as shown above. When this statement is executed, the product will be rounded to the position corresponding to the rightmost position specified in the picture for PRICE. If the digit to the right of this position is five or greater, the digit in this position is increased by one. If the picture specified for PRICE is 999V99 and the values of DISCOUNT and PRICE are 080 and 11111 respectively, the value of PRICE after execution of the statement above will be ..... .

* * *

08889

14.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  PERCENT PIC 99V9.
77  FIRST-TOTAL PIC 9999V99.
```

Write a statement to compute the product of the values of PERCENT and FIRST-TOTAL, round the product to the nearest cent (hundredth), and store it in FIRST-TOTAL.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

MULTIPLY PERCENT BY FIRST-TOTAL ROUNDED.

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       WORKING-STORAGE SECTION.
       77  WIDTH PIC 99V99.
       77  HEIGHT PIC 99V99.
       77  SQUARE-FEET PIC 999V999.
```

The GIVING option and the ROUNDED option may be specified in the same MULTIPLY statement; the reserved word ROUNDED follows the name of the variable in which the result is stored. Write a statement to compute the product of the values of WIDTH and HEIGHT and store the value in SQUARE-FEET with the value rounded.

```
                          *         *         *
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           MULTIPLY WIDTH BY HEIGHT
               GIVING SQUARE-FEET ROUNDED.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DIMENSION-CALCULATION.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INPUT-FILE
               ASSIGN TO RD-1442.
           SELECT AREA-DISK
               ASSIGN TO DF-1-999-X.
       DATA DIVISION.
       FILE SECTION.
       FD  INPUT-FILE
           LABEL RECORDS ARE OMITTED.
       01  DIMENSION-RECORD.
           02  ITEM-NUMBER PIC X(6).
           02  DIMENSION1 PIC 99V9.
           02  DIMENSION2 PIC 99V9.
           02  FILLER PIC X(68).
       FD  AREA-DISK

       01  AREA-RECORD.
           02  ITEMNUMBER PIC X(6).
           02  AREA-2 PIC 999V9.
       WORKING-STORAGE SECTION.
       77  AREA-1 PIC 999V9.
       01  WORK-RECORD.
           02  ITEM-ID PIC X(6).
           02  DIMENSION-1 PIC 99V9.
           02  DIMENSION-2 PIC 99V9.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT-FILE
               OUTPUT AREA-DISK.
       CALCULATE.
```

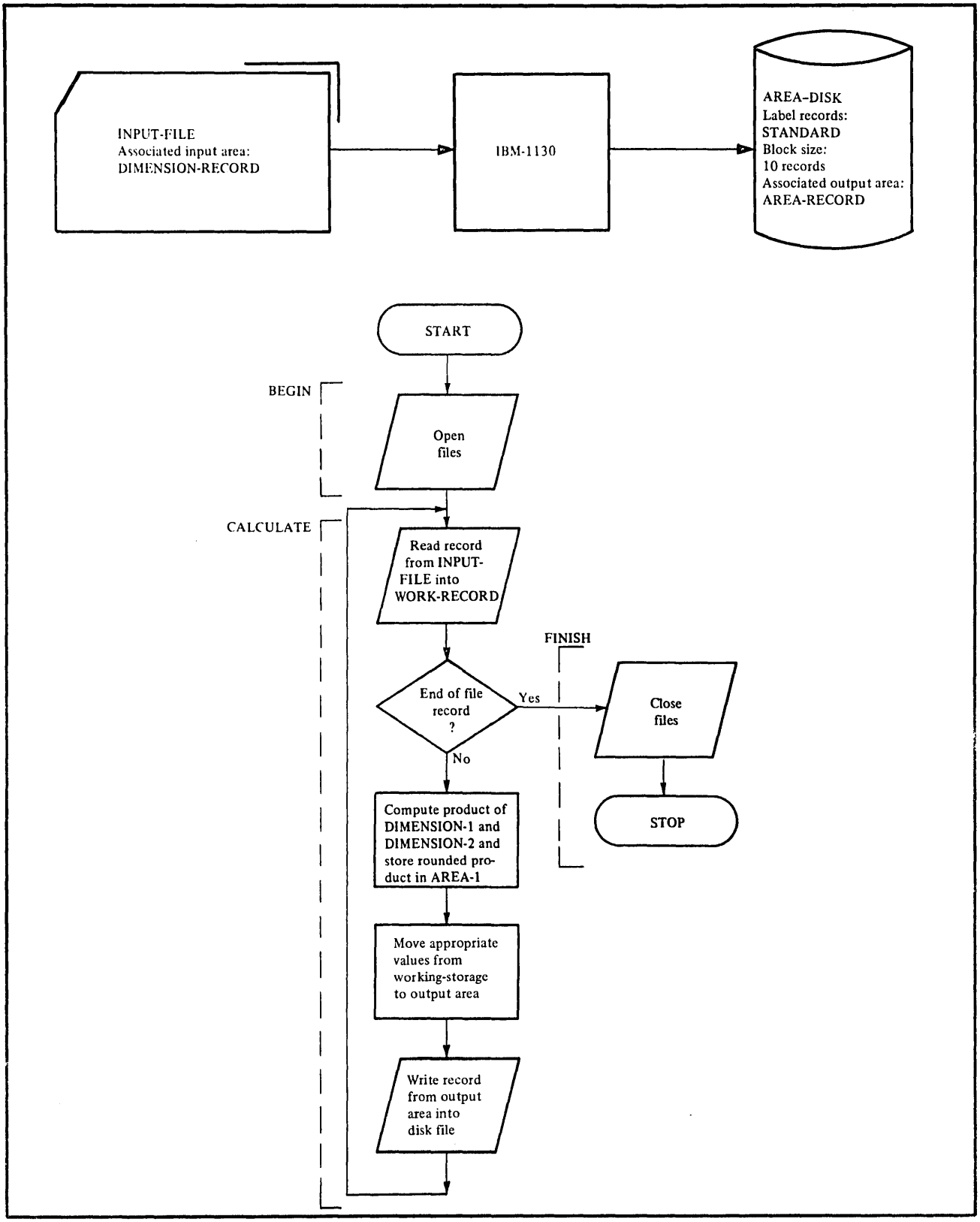

INPUT-FILE
Associated input area:
DIMENSION-RECORD
IBM-1130
AREA-DISK
Label records:
STANDARD
Block size:
10 records
Associated output area:
AREA-RECORD
START
BEGIN
Open files
CALCULATE
Read record from INPUT-FILE into WORK-RECORD
FINISH
End of file record ?
Yes
Close files
No
STOP
Compute product of DIMENSION-1 and DIMENSION-2 and store rounded product in AREA-1
Move appropriate values from working-storage to output area
Write record from output area into disk file

Figure 68

16. World-Wide Floor Coverings, Ltd. has requested a program to compute areas for floor coverings to be announced in their annual catalog. From dimension records in a card file area records are to be created and stored in a file on disk. A system flow chart and a program flow chart are shown in Figure 68 along with portions of the source program. Complete the program DIMENSION-CALCULATION by coding the:

    1) missing portion of the FD entry for AREA-DISK.

    2) remainder of the Procedure Division.

                    *   *   *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           BLOCK CONTAINS 10 RECORDS
           LABEL RECORDS ARE STANDARD.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO WORK-RECORD
               AT END GO TO FINISH.
           MULTIPLY DIMENSION-1 BY DIMENSION-2
               GIVING AREA-1 ROUNDED.
           MOVE AREA-1 TO AREA-2.
           MOVE ITEM-ID TO ITEMNUMBER.
           WRITE AREA-RECORD.
           GO TO CALCULATE.
       FINISH.
           CLOSE INPUT-FILE AREA-DISK.
           STOP RUN.
```

17.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           .
           .
           .
       WORKING-STORAGE SECTION.
       77  HOURS-WORKED PIC 99V9.
       77  HOURLY-RATE PIC 99V99.
       77  GROSS-PAY PIC 999V99.
           .
           .
           .
           MULTIPLY HOURS-WORKED BY HOURLY-RATE
               GIVING GROSS-PAY ROUNDED.
           .
           .
           .
```

If the result of an arithmetic operation will not fit into the designated variable after any specified rounding has been performed, a size error condition exists. If the values of HOURS-WORKED and HOURLY-RATE were 95/5 and 11/25 respectively, the result of the multiplication operation would be 1074/375. After the specified rounding the value would be 1074/38. Since this is still larger than the variable GROSS-PAY, a size error condition would exist. For which of the following products of HOURS-WORKED and HOURLY-RATE would a size error condition exist?

a. 32/926

b. 401/025

c. 1002/05

* * *

c
(The size error condition exists whenever truncation on the left is necessary for storing the result of a calculation.)

18.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY A BY B.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY A BY B
               ON SIZE ERROR
               GO TO ERROR-ROUTINE.
```

If a size error condition occurred during execution of the first statement above, the value of B would be unpredictable. To preclude an unpredictable value, the SIZE ERROR option may be specified as shown in the second statement above. If a size error condition occurred during execution of this statement, the value of B would not be altered. Instead, the statement specified in the SIZE ERROR option would be executed. The statement in the SIZE ERROR option specifies that the:

a. statements following the paragraph name ERROR-ROUTINE will be executed only when a size error condition exists.

b. statement following
ON SIZE ERROR
will be executed each time the MULTIPLY statement is executed.

* * *

a

---

19. To specify action to be taken when the size of the result of an arithmetic operation exceeds the size of the variable in which it is to be stored, the ..... option is included in the statement specifying the operation.

* * *

SIZE ERROR (The action specified should either correct the error or indicate to the operator that an error has occurred and terminate execution.)

---

20. Write a statement to specify that the values of UNIT-PRICE and QUANTITY are to be multiplied and the rounded product stored in TOTAL-PRICE which has the PICTURE clause 999V99. If the product is equal to or greater than 100000, the statements following the paragraph name SPECIAL-RATE are to be executed.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY UNIT-PRICE BY QUANTITY
               GIVING TOTAL-PRICE ROUNDED
               ON SIZE ERROR GO TO SPECIAL-RATE.
```

21.

```
ADD  {variable-name-1}  {variable-name-2}
     {  literal-1    }  {  literal-2    }...TO variable-name-m.
```

You can now specify multiplication in a COBOL program by using the MULTIPLY statement. The statement used to specify addition is the ADD statement of the form shown above. All the values of the variables (or literals) appearing before the reserved word TO are added to the value of variable-m.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD SUBTOTAL TO TOTAL.
```

When the statement above is executed, the:

a. value of SUBTOTAL will be added to the value of TOTAL.

b. sum of SUBTOTAL and TOTAL will be stored in TOTAL.

* * *

Both

22.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD TOTAL1 TOTAL2 100 TO TOTAL.
```

When the statement above is executed, the values of TOTAL1 and TOTAL2 and the value 100 will be added to the value of TOTAL; the sum of the four will be stored in TOTAL. Write a statement to compute the sum of the values of DEPT1, DEPT2, DEPT3, and YEARLY and to store this sum in YEARLY.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD DEPT1 DEPT2 DEPT3 TO YEARLY.
```

---

23. The GIVING option can be included in the ADD statement. The effect is the same as for the MULTIPLY statement. When the GIVING option is used in an ADD statement, the reserved word TO is omitted. Which of the following statements correctly specifies that the sum of FICA, STATETAX, FEDTAX, and INSURANCE is to be computed and stored in DEDUCTIONS?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD FICA STATETAX FEDTAX INSURANCE
               TO DEDUCTIONS.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD FICA STATETAX FEDTAX TO INSURANCE
               GIVING DEDUCTIONS.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD FICA STATETAX FEDTAX INSURANCE
               GIVING DEDUCTIONS.
```

* * *

c

---

24. The ROUNDED and SIZE ERROR options may also be included in an ADD statement. You might assume that the:

a. ROUNDED option would cause the sum to be rounded to the number of decimal places specified for the variable in which it is to be stored.

b. SIZE ERROR option would specify the action to be taken if the sum exceeds the size specified for the variable in which it is to be stored.

* * *

Both

25.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       WORKING-STORAGE SECTION.
       77  TOTAL PIC 999V99.
       77  TOTAL-1 PIC 999V99.
       77  TOTAL-2 PIC 999V99.
```

Write a statement specifying that the values of TOTAL-1 and TOTAL-2 are to be added and their sum is to be rounded to the nearest hundredth and stored in TOTAL. If the sum is equal to or greater than 1000, the value of STORE-NUMBER is to be displayed on the console typewriter.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           ADD TOTAL-1 TOTAL-2 GIVING TOTAL
               ROUNDED ON SIZE ERROR
               DISPLAY STORE-NUMBER
               UPON CONSOLE.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. MASTER-CUSTOMER-DISK.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CUSTOMER-FILE
               ASSIGN TO RD-1442.
           SELECT CUSTOMER-DISK
               ASSIGN TO DF-2-100.
       DATA DIVISION.
       FILE SECTION.
       FD  CUSTOMER-FILE
           LABEL RECORDS ARE OMITTED.
       01  CUSTOMER-RECORD.
           02  CUSTOMER-NUMBER PIC X(6).
           02  NAME PIC X(20).
           02  HOME-ADDRESS PIC X(30).
           02  YEAR-OPENED PIC XX.
           02  MAXIMUM-BILL PIC X(6).
           02  FILLER PIC X(16).

       FD  CUSTOMER-DISK
       01  MASTER-RECORD.
           02  NAME PIC X(20).
           02  CUSTOMER-NUMBER PIC X(6).
           02  HOME-ADDRESS PIC (X30).
           02  YEAR-OPENED PIC XX.
           02  MAXIMUM-CREDIT PIC 9(6).
           02  MAXIMUM-BILL PIC 9(6).
           02  PRESENT-BILL PIC 9(6).
           02  PAYCODE PIC X.
       WORKING-STORAGE SECTION.
       77  COUNTER PIC 9(6) VALUE ZEROS.
```

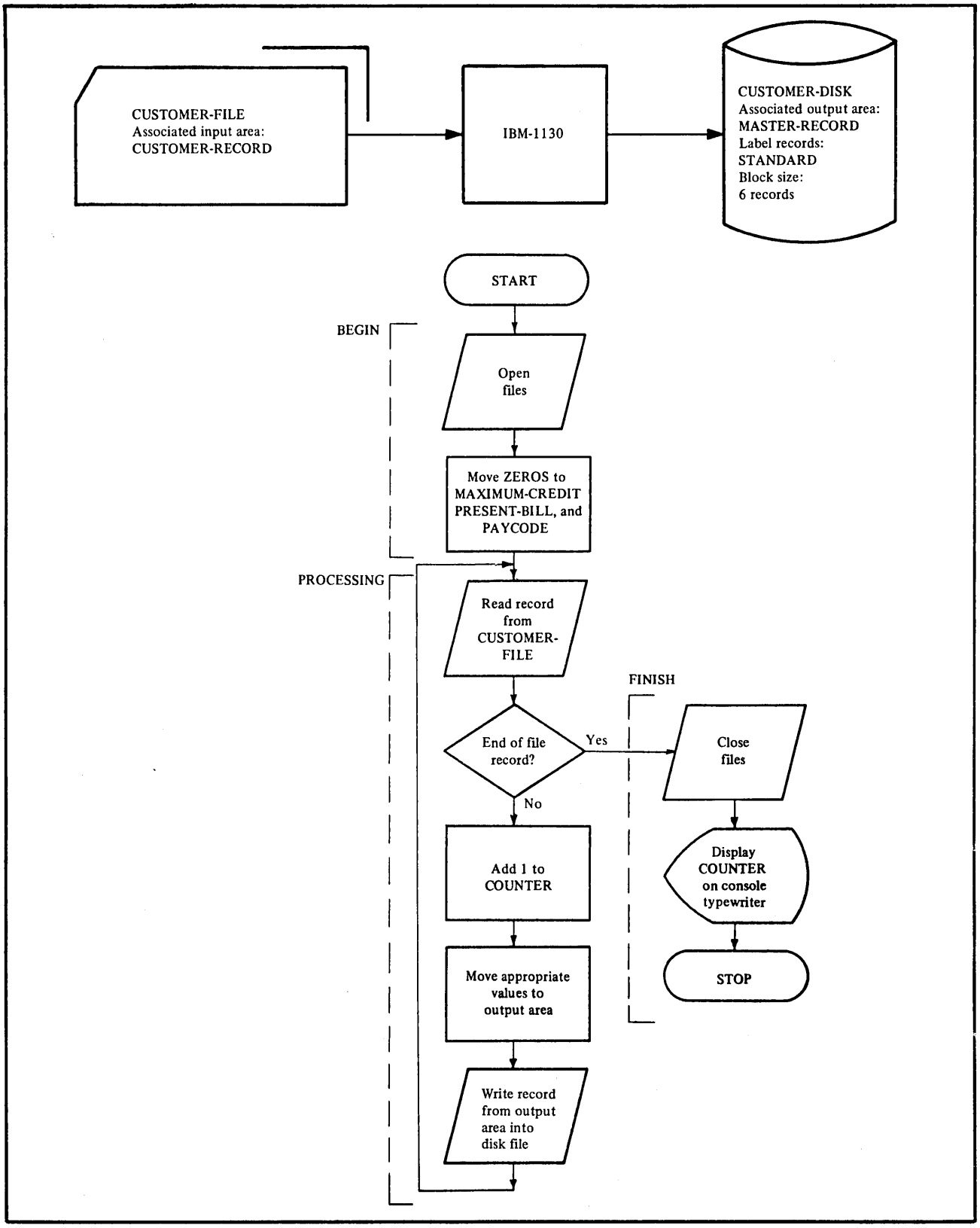

CUSTOMER-FILE
Associated input area:
CUSTOMER-RECORD
IBM-1130
CUSTOMER-DISK
Associated output area:
MASTER-RECORD
Label records:
STANDARD
Block size:
6 records
START
BEGIN
Open files
Move ZEROS to MAXIMUM-CREDIT PRESENT-BILL, and PAYCODE
PROCESSING
Read record from CUSTOMER-FILE
FINISH
End of file record?
Yes
Close files
No
Add 1 to COUNTER
Display COUNTER on console typewriter
Move appropriate values to output area
STOP
Write record from output area into disk file

Figure 69

26. As a result of recent expansion of the line of products at Universal Enterprises, the number of customers has greatly increased. Since customer master records are stored in punched cards, the increase in the number of customers has created a problem for the firm's data processing center, in terms of both storage and processing time. To alleviate the situation the master records are to be stored on disk. The program MASTER-CUSTOMER-DISK is to transfer the records from cards to disk. A system flow chart and a program flow chart are included in Figure 69. A portion of the program that has already been coded is also included. You are to complete this program to create a master file on disk by coding the:

    1) missing portion of the FD entry for CUSTOMER-DISK.

    2) Procedure Division.

    (The variable COUNTER is used to count the number of customer records.)

* * *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          BLOCK CONTAINS 6 RECORDS
          LABEL RECORDS ARE STANDARD.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT CUSTOMER-FILE
               OUTPUT CUSTOMER-DISK.
           MOVE ZEROS TO MAXIMUM-CREDIT.
           MOVE ZEROS TO PRESENT-BILL.
           MOVE ZERO TO PAYCODE.
       PROCESSING.
           READ CUSTOMER-FILE AT END
               GO TO FINISH.
           ADD 1 TO COUNTER.                        (35)
           MOVE NAME OF CUSTOMER-RECORD
               TO NAME OF MASTER-RECORD.
           MOVE CUSTOMER-NUMBER OF CUSTOMER-RECORD
               TO CUSTOMER-NUMBER OF MASTER-RECORD.
           MOVE HOME-ADDRESS OF CUSTOMER-RECORD
               TO HOME-ADDRESS OF MASTER-RECORD.
           MOVE YEAR-OPENED OF CUSTOMER-RECORD
               TO YEAR-OPENED OF MASTER-RECORD.
           MOVE MAXIMUM-BILL OF CUSTOMER-RECORD
               TO MAXIMUM-BILL OF MASTER-RECORD.
           WRITE MASTER-RECORD.
           GO TO PROCESSING.
       FINISH.
           CLOSE CUSTOMER-FILE
               CUSTOMER-DISK.
           DISPLAY COUNTER UPON CONSOLE.
           STOP RUN.
```

(Moving zeros, spaces, or any other constant to certain variables at the beginning of a program is referred to as initializing the variables. If your solution appears different and you are not sure of its correctness, consult your advisor.)

SUMMARY:

You have just learned to code MULTIPLY and ADD statements including the:

GIVING option to specify the variable in which the computed result will be stored.

ROUNDED option to specify that the computed result will be rounded to the nearest digit in the position corresponding to the rightmost position in the picture for the variable in which the result is to be stored.

SIZE ERROR option to specify the action to be taken when the computed result exceeds the size of the variable in which the result is to be stored.

END OF LESSON 14

LESSON 15

LESSON 15 - EDITING NUMERIC DATA

INTRODUCTION

Specification of arrangement of data in a printed report is an important aspect of COBOL programming. In previous lessons you have learned to specify vertical and horizontal spacing in a printed report. In this lesson you will learn several ways to specify how individual data items are to appear. You will learn to specify that a decimal point, dollar sign, or comma is to be printed in the appropriate place in a data item. This is commonly called editing variables for reports.

Specific language features you will learn in this lesson are:

Picture character .(decimal point)
Picture character $(dollar sign)
Picture character ,(comma)

This lesson will require approximately three quarters of an hour.

1. You know that a V appearing in a picture specifies that the value of the associated variable will have an assumed decimal point. A point appearing in the picture specifies that the value of the associated variable will have an actual decimal point. You would expect that a point appearing in a picture would cause a:

   a. decimal point to be printed with the data item.

   b. separate character position to be reserved in the value.

* * *

Both

---

2. Any variable whose picture contains 9's and a V or just 9's may be used in calculations. To print numeric values, however, you would want an actual decimal point to be placed in the appropriate position in the value. A variable PRICE has the picture 99V99. In order to print the value of PRICE, you would move it to an edited variable with a picture 99.99. Another variable DOZENS has the picture 999V9. You would move DOZENS to an edited variable with the picture 999.9 if you wished to:

   a. have calculations performed on the value.

   b. include the value of DOZENS in a printed report.

* * *

b

3.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
MOVE numeric-variable
    TO edited-variable
```

numeric variable 9V9

edited variable 99.99

A value moved from a numeric variable (9's or 9's and a V) to an edited variable containing a . is aligned with the assumed decimal point being placed in the actual decimal point position. Any extra 9 positions on either end of the edited variable are filled, or padded, with zeros. Assume the numeric variable above is RATE with a value of 52 and the edited variable is PAY-SCALE. The value of PAY-SCALE after the MOVE statement would be:

   a. 5.2

   b. 05.2

   c. 05.20

   d. 5.20

* * *

c

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  PAYMENT PIC 999V9999.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  AMOUNT PIC 999.99.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE PAYMENT TO AMOUNT.
```

The actual point (.) is used to edit numeric data for printed reports while the assumed point (V) is used in data for calculations. Sometimes it is not necessary to print as many digit positions as are used in calculations. Assume that the value of PAYMENT is 2734201. When the MOVE statement is executed, the V is aligned with the . and the three digit positions preceding the point are moved intact, but only the first two digits following the point are moved since AMOUNT has only room for two digits. The last two digits are truncated, or lost, and the value of AMOUNT is:

a. 273.4201

b. 273.42

* * *

b

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  RATE PIC 999.99.
           02  AVERAGE PIC 999.9.
```

A point is assumed to the right of the rightmost digit of any numeric literal or data item that contains no actual point or no assumed point specified by a V in a picture. Every numeric literal or numeric data item, thefore, contains either an assumed or an actual point. The point, whether assumed or actual, is always aligned with any actual point specified in the picture for a receiving variable. If the value 24 were moved to RATE and the value of RATE were then printed, the value would be printed. If the value 73 was moved to AVERAGE and the value of AVERAGE was then printed, the value ..... would be printed.

* * *

073.0

6.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  RATE PIC 99.99.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  PER-CENT PIC XXXX.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE RATE TO PER-CENT.
```

No decimal point alignment takes place when an edited variable is moved to an alphanumeric (X) variable. The MOVE is accomplished from left to right, one character at a time. After the value 5025 is moved to RATE and the MOVE statement above is executed, the value of PER-CENT is:

a. 50.25

b. 5025

c. 0.25 (The leftmost character is truncated because RATE has five positions and PER-CENT has only four)

* * *

None of these
(The rightmost character is truncated giving 50.2 as shown below.

```
sending variable
    RATE                50.25

                        1 2 3 4
receiving variable
    PER-CENT            XXXX
```

The rightmost digit of the sending variable is truncated since there is no room in the receiving variable. The decimal point is moved the same as any other character and must be counted in the length of the receiving field).

7.

JUSTIFIED clause

The JUSTIFIED clause can be specified in the definition of an alphanumeric elementary variable to cause values of the variable to be aligned on the right end of the variable rather than on the left.

```
{JUSTIFIED} RIGHT
{JUST      }
```

Normally, the rule for positioning data within a receiving alphanumeric or alphabetic data item is:

- The data is aligned in the receiving field beginning at the leftmost character position within the receiving field. Unused character positions to the right are filled with spaces. If truncation occurs, it will be at the right.

  The JUSTIFIED clause affects the positioning of data in the receiving field as follows:

1. When the receiving data item is described with the JUSTIFIED clause and the data item sent is larger than the receiving data item, the leftmost characters are truncated.

2. When the receiving data item is described with the JUSTIFIED clause and is larger than the data item sent, the data is aligned at the rightmost character position in the data item. Unused character positions to the left are filled with spaces.

The JUSTIFIED clause may only be specified for elementary items.

This clause must not be specified for level-88 data items.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  LAST-NAME PICTURE X(30) JUSTIFIED RIGHT.
```

---

8. The SYNCHRONIZED clause specifies the alignment of an elementary item on one of the proper boundaries in core storage.

   The SYNCHRONIZED clause is used to ensure efficiency when performing arithmetic operations on an item.

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

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  DISK-FILE.
           02  DISK-ID PIC X(5).
           02  DISK-NAME PIC X(29).
           02  ON-HAND PIC S9(6) USAGE COMP SYNC LEFT.
           02  ON-ORDER PIC S9(6) USAGE COMP SYNC LEFT.
```

The structure of 1130 COBOL is such that boundary alignment is inconsequential; thus the clause is of no functional value and is, in effect, treated as comments.

9.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       77  FACTOR1 PIC 9V99.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       77  FACTOR2 PIC 9.99.
```

Which of the following is a correct interpretation of the data description entries above?

a. If 123 were moved to FACTOR1, the value of FACTOR1 would be 1.23.

b. If 123 were moved to FACTOR2, the value of FACTOR2 would be 123.

c. The value of FACTOR1 could be moved to a variable with the picture XXX without truncation occurring.

d. The value of FACTOR2 could be moved to a variable with the picture XXXX without the data item's being padded.

* * *

c,d

10.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       77  MULTIPLICAND PIC 9V9.

           02  MULTIPLIER PIC 99.99.

               MOVE MULTIPLICAND TO MULTIPLIER.
```

When an item with an assumed decimal point is moved to a variable with a picture that specifies an actual decimal point, the item is placed in the variable with the assumed and actual points aligned. Padding with zeros or truncation occurs to the left and right as necessary. The value of MULTIPLICAND is 75. After execution of the MOVE statement the value of MULTIPLIER will be ..... .

* * *

07.50

11.

OUTPUT-RECORD		
ITEM-NUMBER (6-digit integer)	COST (6-digit number with two decimal places and an actual point)	PERCENT-SECURED (5-digit number with three decimal places and an assumed point)

Write the record description entry for the record variable OUTPUT-RECORD illustrated above.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
01  OUTPUT-RECORD.
    02  ITEM-NUMBER PIC 9(6).
    02  COST PIC 9999.99.
    02  PERCENT-SECURED PIC 99V999.
```

12.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    02  AMOUNT PIC $999.99.
```

An actual decimal point is usually specified for monetary values that are to be printed. A dollar sign may also be specified. As a result, a dollar sign will be inserted into the printed value in the position corresponding to its position in the picture. If 77.25 is moved to AMOUNT, the value printed will be $077.25. If 550 is moved to AMOUNT, the value printed will be ..... .

* * *

$005.50

---

13. Write the picture for the variable TOTAL to specify that values of TOTAL are to be printed as six-digit numbers with two decimal places and an actual decimal point and that the values are to be preceded by a dollar sign.

* * *

$9999.99
(This could also be written $9(4).99.)

14.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  AMOUNT PIC $999.99.
```

Like the actual decimal point, the $ becomes a part of the value of the variable and therefore must be counted in the length of the value. To which of the following variables could the value of AMOUNT be moved without truncation occurring?

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PERMANENT-RECORD.
           02  VARIABLE-1 PIC X(5).
           02  VARIABLE-2 PIC X(6).
           02  VARIABLE-3 PIC X(7).

                          *         *         *
```

VARIABLE-3

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  AMOUNT PIC $999.99.
```

When AMOUNT is specified as shown above and the value of AMOUNT has fewer than three digits to the left of the point, you know that the value will be padded with zeros to the left and the dollar sign will be inserted to the left of the zeros. It is possible to specify that the dollar sign will be inserted immediately to the left of the first nonzero digit to the left of the point.

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  AMOUNT PIC $$$$.99.
```

Value moved to AMOUNT	Printed value of AMOUNT
7/21	$7.21
012/3	$12.30
100/	$100.00
2319/	$319.00
/825	$.82
00/05	$.05

The PICTURE clause above specifies that a dollar sign will be inserted immediately to the left of the first nonzero digit preceding the decimal point. The table above shows the effect of moving values to AMOUNT. Refer to the table to decide which of the following statements apply to the use of a string of dollar signs.

a. The position of the printed dollar sign "floats" through the positions represented by the dollar sign in the picture.

b. Leading zeros to the left of the point are suppressed.

c. The leftmost dollar sign does not represent a digit position.

d. Printing the value of AMOUNT would require reserving six character positions on the printer.

* * *

a,b,c
(Seven character positions would be required for printing the value AMOUNT. The maximum required space must be reserved.)

16.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  UNIT-PRICE PIC $$$.99.
```

Give the printed value when each of the following values is moved to UNIT-PRICE and then printed.

1) /25

2) 29/95

3) 1/19

4) /05

5) 119/50

6) 00/00

* * *

1) $.25
2) $29.95
3) $1.19
4) $.05
5) $19.50
6) $.00

17.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  COST PIC $$9.99.

           02  PRICE PIC $$99.99.
```

The floating dollar sign need not be placed in every position to the left of the point. It must, however, begin in the leftmost position and occur in consecutive digit positions. Moving 05 to COST which is defined above will result in $0.05, since the 9 in the picture indicates that the position will contain a digit. Give the printed value after each of the following values has been moved to PRICE and then printed.

1) /25

2) /0

3) 119/

4) 1/98

* * *

1) $00.25
2) $00.00
3) $119.00
4) $01.98

18.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  BALANCE PIC $$$$.99.

           02  BALANCE PIC $$$$.$$.

           02  AMOUNT-DUE PIC $$$.$$.
```

The dollar sign may be specified in all positions of the picture of a variable. If dollar signs are present to the right of the point, the value of the data item is not printed if it is equal to zero. Any other value is printed just as if no dollar sign were present to the right of the point. Give the printed value of AMOUNT-DUE if the value moved to is:

1) zero

2) 79/28

3) /03

* * *

1) nothing
2) $79.28
3) $.03

---

19. Match the data item with the picture(s) that would cause it to be printed in the form shown.

1)	$$$.99	a.	$01.23
2)	$99.99	b.	00.5
3)	$$9.99	c.	$10.25
4)	99.9	d.	$.25
5)	$$$.$$	e.	$0.70
		f.	123.25
		g.	nothing

* * *

1) c,d
2) a,c
3) c,e
4) b
5) c,d,g

20.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  FUND PIC 999,999.
```

Large numbers are easier to read if commas are used in the appropriate positions. It is possible to specify that a comma is to be inserted in a value by placing a comma in the appropriate position in the picture. The picture of FUND above specifies that a comma is to appear in the printed value in the position corresponding to the position of the comma in the picture. If the value of FUND is 876003, the printed value will appear as ..... .

* * *

876,003

21.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PRINT-RECORD.
           02  ITEMNUMBER PIC 9(5).
           02  QUANTITY PIC 99,999.
           02  GROSS PIC $$9,999.99.
           02  COMMISSION PIC 99.99
```

Show how the printed values of the variables in the record description entry above would appear if the following values had been moved to the variables.

Variable	Value moved to variable	Printed value of variable
ITEMNUMBER	77342	
QUANTITY	8995	
GROSS	55624/93	
COMMISSION	10/00	

* * *

Printed value of variable

```
    77342
   08,995
$55,624.93
    10.00
```

22.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  TOTAL-PRICE PIC $$$,$$$.99.
```

A comma may be included in a string of dollar signs. The comma will be printed only if at least one digit is printed preceding the comma position. If the value 1772/20 were moved to TOTAL-PRICE and printed, it would appear as ..... . If the value 204/ were moved to TOTAL-PRICE and printed, it would be printed as ..... .

* * *

$1,772.20
$204.00

23.

STATISTICS-RECORD			
SOCIAL-SECURITY-NUMBER (9 char)	PERCENTAGE Value Range: 00.00-85.00	GROSS Value Range: $.00-$900,000.00	CONTRIBUTION Value Range: $0.00-$50.00

Write the record description entry for STATISTICS-RECORD. Account for the editing of all variables as illustrated in the diagram of the record above.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  STATISTICS-RECORD.
           02  SOCIAL-SECURITY-NUMBER PIC X(9).
           02  PERCENTAGE PIC 99.99.
           02  GROSS PIC $$$$,$$$.99.
           02  CONTRIBUTION PIC $$9.99.
```

(The picture for CONTRIBUTION has a permanent digit position preceding the decimal point to allow for printing as shown in the diagram.)

---

24. The contents of a variable whose picture consists of either all 9's or 9's and a V is considered numeric data. The contents of a variable whose picture consists of 9's and a decimal point, a comma, and/or one or more dollar signs is numeric-edited data. Match the kind of data that the variable will contain with each data description entry.

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  QUOTA PIC 999.99.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  CUSTOMER-NUMBER PIC 9(6).
```

3)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  TOTAL PIC 999V99.
```

4)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  AMOUNT PIC $$99.99.
```

5)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  QUANTITY PIC 999,999.
```

a. numeric data

b. numeric-edited data

* * *

1) b
2) a
3) a
4) b
5) b

---

25. Match each picture character with the description of the picture in which it can appear.

1) Picture for variable containing numeric data	a. 9
2) Picture for variable containing numeric-edited data	b. $
	c. comma
	d. decimal point
	e. V

* * *

1) a,e
2) a,b,c,d

26.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  TOTAL 999.99.

           02  PERCENT 99V99.

           02  QUANTITY 9(5).

           02  AMOUNT PIC $$999.
```

Variables containing numeric-edited data cannot be specified in calculations, although they may be specified in the GIVING option. According to the data description entries above, which of the following statements is correct?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD QUANTITY TO TOTAL.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY PERCENT BY QUANTITY
               GIVING TOTAL.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY PERCENT BY AMOUNT.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           ADD 10 QUANTITY GIVING AMOUNT.
```

* * *

b,d

SUMMARY:

You have just learned how to incorporate the decimal point, dollar sign, and comma into printed variable output. By means of the editing process just presented, these symbols are printed in the appropriate places in data items, according to your specifications.

END OF LESSON 15

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 16

LESSON 16 - CONDITIONAL BRANCHING

INTRODUCTION

Tests and branching decisions based on the results of these tests are a necessary part of any data-processing procedures. The AT END option of the READ statement and the AT-END-OF-PAGE option of the WRITE statement are two ways of specifying branching on the basis of a test, or conditional branching, in a COBOL program. Another important feature of COBOL used to specify conditional branching is the IF statement, which you will learn to code in this lesson.

Specific language features you will learn in this lesson are:

IF statement
EXAMINE statement

This lesson will require approximately three quarters of an hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF CARD-NUMBER
               LESS THAN MASTER-NUMBER
               ADD 1 TO COUNTER.
```

The statement above is an example of a COBOL IF statement. An IF statement causes a condition to be evaluated, or tested, and an action to be taken based on whether the result of the test is true or false. You might expect that the condition in the IF statement above is:

a. CARD-NUMBER LESS THAN MASTER-NUMBER

b. ADD 1 TO COUNTER

* * *

a

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF TOTAL EQUAL TO 100 GO TO MATCH.
```

The condition in the statement above is ..... .

* * *

TOTAL EQUAL TO 100

---

3. The examples in the previous frames show that a condition can be:

a. a statement of a relationship between two variables or between a variable and a numeric literal.

b. an instruction to the computer.

* * *

a

---

4. The phrases LESS THAN, EQUAL TO, and GREATER THAN are used in conditions in a COBOL statement to express a relationship between two variables or between a variable and a numeric literal. Write conditions to express the following relationships:

   a. the value of TEST-1 is greater than 10.

   b. the value of EMPLOYEE-NUMBER is equal to the value of WORK-RECORD-NUMBER.

* * *

a. TEST-1 GREATER THAN 10
b. EMPLOYEE-NUMBER EQUAL TO
   WORK-RECORD-NUMBER

(The optional word IS may precede any of the three phrases given above.)

---

5. The appearance of a condition in an IF statement causes the condition to be evaluated. The result will be either true or false. Give the result (true or false) of the evaluation of the conditions in the following IF statements when CARD-NUMBER and MASTER-NUMBER have the values 250 and 500, respectively.

   1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF CARD-NUMBER
               LESS THAN MASTER-NUMBER
               ADD 1 TO COUNTER.
```

   2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF MASTER-NUMBER EQUAL TO 1000
               GO TO SECOND-ROUTINE.
```

* * *

1) true
2) false

---

6. Nested IF statements are not permitted in 1130 COBOL.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SERVICE CODE EQUAL TO 27
               ADD 5.00 TO AMOUNT,
               IF CALL TIME LESS THAN MINIMUM
                   MOVE 'BASE RATE' TO LEGEND.
```

Figure 70

Figure 70 above illustrates the IF statement. As a result of the above coding:

a. if SERVICE CODE is equal to 27,5.00 will be added to AMOUNT

b. a compile time error will occur

* * *

b (Nested IF statements are invalid).

7.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF condition statement-1.
```

A general form of the IF statement is shown above. If the result of the evaluation of the condition is true, statement-1 is executed. Then the sentence immediately following the IF statement is executed (unless statement-1 is a GO TO statement.) If the result is false, the sentence immediately following the IF statement is executed.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF SALES GREATER THAN 100
    MOVE 'ABOVE' TO MEMO.
ADD SALES TO YEAR-TO-DATE SALES.
```

When the IF statement above is executed, if SALES is equal to:

a. 50, 'ABOVE' will be moved to MEMO and the value of SALES will be added to the value of YEAR-TO-DATE SALES

b. 150, 'ABOVE' will be moved to MEMO and the ADD statement will not be executed.

* * *

Neither (If SALES is greater than 100, 'ABOVE' will be moved to MEMO and then the ADD statement will be executed. If SALES is 100 or less, the MOVE statement will not be executed; the ADD statement will be the next statement executed.)

---

8. Write a statement to specify that if SCORE is greater than 89, 4 will be added to GRADE-POINT.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SCORE GREATER THAN 89
               ADD 4 TO GRADE-POINT.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SALES GREATER THAN 100
               MOVE 'ABOVE' TO MEMO
               ADD SALES TO YEAR-TO-DATE SALES.
```

More than one statement may be included in an IF statement. When the condition in the statement above is true, the MOVE and ADD statements will be executed. When the condition is false, neither the MOVE nor the ADD statement will be executed, and control will be transferred to the next sentence; that is, to the statement following the next period.

The IF statement of the form you have been using can be used to specify an additional step, an action to be taken, when a certain condition exists. Another situation can also be provided for by an IF statement.

9.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF TOTAL1 EQUAL TO TOTAL2
              ADD 1 TO NEW
              ELSE ADD 1 TO COUNTER.
          READ NEW-FILE.
```

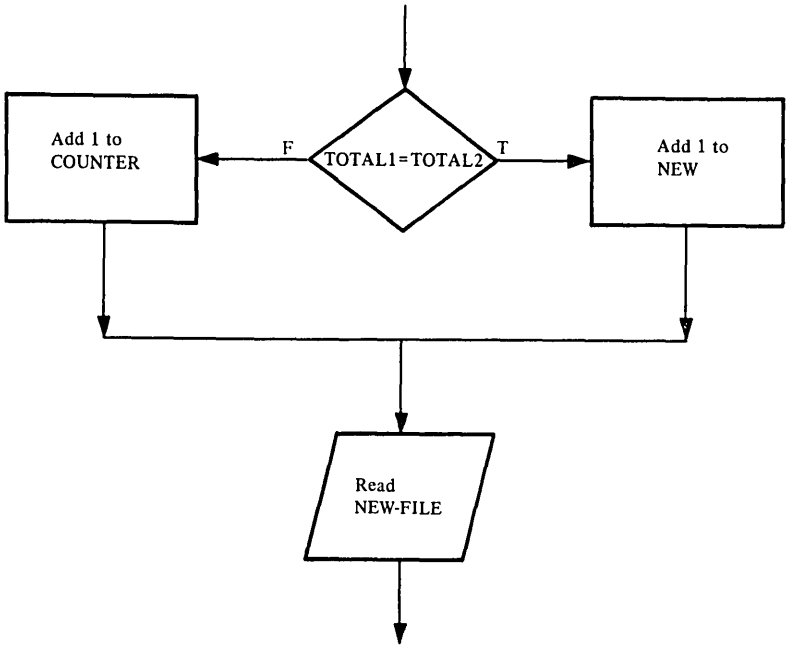


Figure 71

You may wish to specify two different actions to be taken based on the two results of a test as illustrated above. The flow chart shows the action specified by the IF statement in which the ELSE option has been specified. The flow chart and the statement show that:

a. if the values of TOTAL1 and TOTAL2 are equal, 1 will be added to the value of NEW and the sentence following the IF statement will be executed.

b. if the values of TOTAL1 and TOTAL2 are not equal, 1 will be added to the value of COUNTER and the sentence following the IF statement will be executed.

c. the statement specified in the ELSE option will be executed only when the condition is false.

* * *

All of these

---

10. Identify the IF statement(s) below that specifies the following: If the value of SCORE is less than 50, 'FAIL' is to be moved to GRADE. If the value of SCORE is equal to or greater than 50, 'PASS' is to be moved to GRADE. In either case, the next step is to transfer control to the paragraph MAIN-ROUTINE.

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SCORE GREATER THAN 50
               MOVE 'FAIL' TO GRADE
               ELSE MOVE 'PASS' TO GRADE.
           GO TO MAIN-ROUTINE.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SCORE LESS THAN 50
               MOVE 'FAIL' TO GRADE
               ELSE  MOVE 'PASS' TO GRADE.
           GO TO MAIN-ROUTINE.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SCORE LESS THAN 50
               MOVE 'FAIL' TO GRADE.
           MOVE 'PASS' TO GRADE.
       GO TO MAIN-ROUTINE.
```

* * *

b

---

11. The ELSE option of an IF statement is:

a. executed only if the condition is false.

b. not executed if the condition is true.

* * *

Both

---

12. QUANTITY, MINIMUM, and DISCOUNT are numeric variables. Write a statement to specify that if QUANTITY is equal to MINIMUM, 10 is to be moved to DISCOUNT and that if QUANTITY is not equal to MINIMUM, 10 is to be added to DISCOUNT.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF QUANTITY EQUAL TO MINIMUM
               MOVE 10 TO DISCOUNT
               ELSE ADD 10 TO DISCOUNT.
```

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF ACCOUNT-NUMBER LESS THAN 1000
               ADD 1 TO COUNTER
               ELSE GO TO EXCESS.
           READ MASTER FILE.
```

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

               IF PROFIT EQUAL TO 1 GO TO FIRST-RUN
                   ELSE GO TO SECONDARY.
       TOTAL-ROUTINE.
           MOVE ZEROS TO TOTAL-1.
```

If a GO TO statement is included in an IF statement either following the condition (as statement-1) or in the ELSE option, or in both places, the sentence immediately following the IF statement may not be executed after the IF statement. When the first IF statement above is executed, execution of the READ statement will follow execution of the ADD statement if ACCOUNT-NUMBER is less than 1000. If ACCOUNT-NUMBER is not less than 1000, control will transfer to a routine called EXCESS and the READ statement will not be executed. The second IF statement above specifies that execution of the MOVE statement will immediately follow execution of the IF statement if PROFIT is:

a. not equal to 1.

b. equal to 1.

* * *

Neither (Execution of the MOVE statement can never immediately follow execution of this IF statement because both the statement following the condition and the statement in the ELSE option transfer control to another point in the program.)

14.

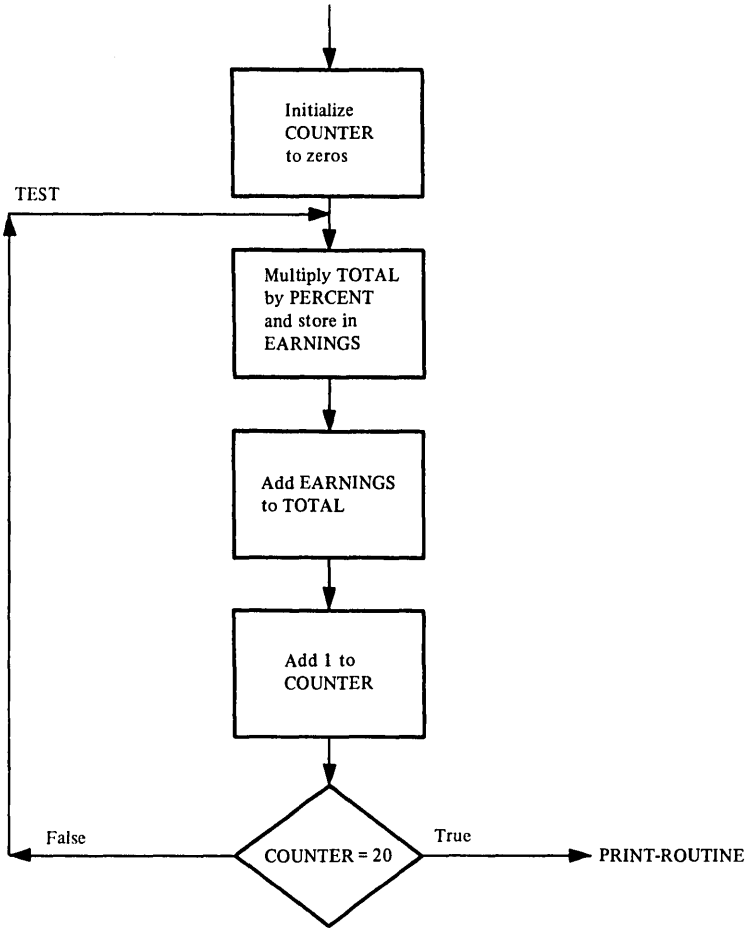

Initialize
COUNTER
to zeros
TEST
Multiply TOTAL
by PERCENT
and store in
EARNINGS
Add EARNINGS
to TOTAL
Add 1 to
COUNTER
False
COUNTER = 20
True
PRINT-ROUTINE

Figure 72

An IF statement can be used to control the number of times a portion of a program is to be executed. The flow chart above shows a portion of a program that is to be executed twenty times to compute the total on deposit after five years when interest is compounded quarterly. Code the sequence shown in the flow chart, using an IF statement to control the number of times the portion will be repeated.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        MOVE ZEROS TO COUNTER.
    TEST.
        MULTIPLY TOTAL BY PERCENT
            GIVING EARNINGS.
        ADD EARNINGS TO TOTAL.
        ADD 1 TO COUNTER.
        IF COUNTER EQUAL TO 20
            GO TO PRINT-ROUTINE
            ELSE GO TO TEST.
```

(The last two lines could also be

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

            GO TO PRINT-ROUTINE.
        GO TO TEST.
```

In any IF sentence that does not contain an ELSE option, a false condition will cause a control to be transferred to the next sentence.

---

15. A condition causes the object program to select between alternate paths of control depending on the truth value of a test. Conditions are used in IF statements. See CONDITIONS chapter in your Language Specifications Manual.

IF identifier-1 IS [NOT] { NUMERIC / ALPHABETIC } GO TO identifier-2.

A test condition is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression is evaluated.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        IF ITEM-NO IS NOT NUMERIC GO TO ALPHA.
        ADD 1 TO COUNT.
```

In the above example, if the contents of ITEM-NO are not numeric the program will branch to a paragraph named ALPHA. If the contents of ITEM-NO are numeric, the next sequential instruction will be executed. In this case, a 1 will be added to COUNT.

16.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF LAST-NAME IS ALPHABETIC GO TO PRINT-NAME.
```

The above sentence will branch to PRINT-NAME paragraph if the contents of all characters in the LAST-NAME are:

a. All numeric. Consisting of digits 0 through 9.

b. All alphabetic. Consisting of letters A through Z and/or spaces.

c. Alphanumeric. Consisting of digits and letters.

* * *

b

---

17. The sign condition determines whether or not the algebraic value of a numeric operand (i.e. an item described as numeric) is less than, greater than or equal to zero.

```
IF { identifier            }      IS [NOT]  { POSITIVE }
   { arithmetic-expression }                { NEGATIVE }
                                            { ZERO     }
```

---

18. An operand is POSITIVE if its value is greater than ZERO, NEGATIVE if it is less than ZERO and ZERO if it is equal to ZERO.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF AMOUNT IS NEGATIVE MOVE ZEROS TO AMOUNT
           ELSE COMPUTE TOTAL = AMOUNT * QTY.
```

In the segment above, which of the following is true:

a. If the AMOUNT is less than zero, ZEROS will be moved into AMOUNT.

b. If the AMOUNT is less than zero, the COMPUTE statement will be executed.

* * *

a

19.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF PRICE IS POSITIVE GO TO CALCULATE.
           GO TO ERROR-ROUTINE.
       CALCULATE.
           COMPUTE TOTAL = QTY * PRICE.
```

In the segment above control of the execution will be given to the CALCULATE paragraph if the contents of PRICE are:

a. greater than zero.

b. equal to zero.

c. zero.

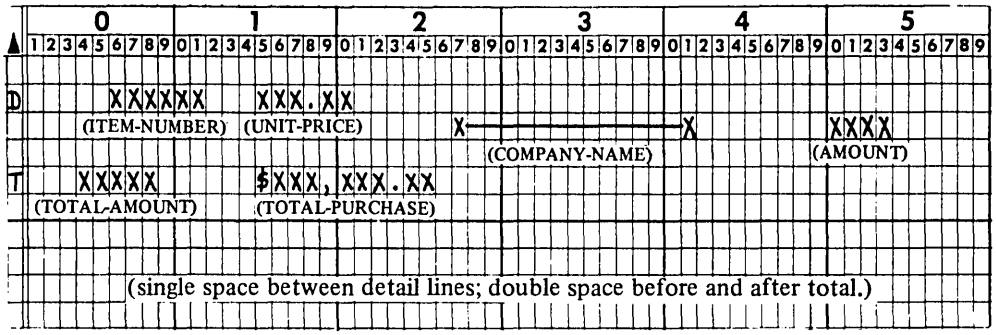
* * *

a

---

20. Read Figure 73.

Adolphe Manufacturing Company wants a program to analyze the marketability of its products. The input card file ITEM-FILE contains a record for each purchase for each item. The file is organized according to ITEM-NUMBER. The plan for the output listing that is to be produced is shown below.



ITEM-NUMBER and UNIT-PRICE will appear once, followed by the listing of purchases and the number of items purchased by each.

The illustration to the right is a sample of the cards from the file ITEM-FILE. The first card shows a sample of the data that appears in each record.

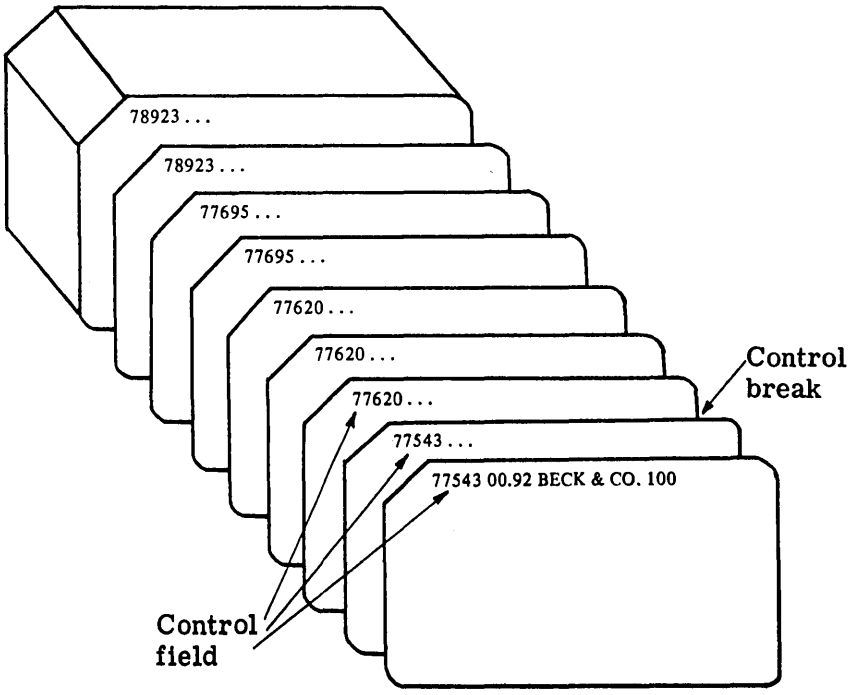


Figure 73

There may be any number of cards for each item in the problem described in Figure 73. In order to meet the specifications stated in the problem statement, all of the cards that represent orders for a particular item must be grouped together. Consequently, the programmer must be able to identify a field appearing in each card that can be used to identify the group to which it belongs. The best field to use in this case would be:

a. UNIT-PRICE

b. ITEM-NUMBER

* * *

b
(More than one item might have the same unit price. Although ITEM-NUMBER is actually the variable that will contain the values from this field, in the context of this problem the field itself will be referred to as ITEM-NUMBER.)

---

21. Figure 73 shows that ITEM-NUMBER identifies a control field. You might expect that a control field is:

    a. the field used to identify specific groups of records within a file.

    b. the name of a variable whose value is the same in every record in the file.

* * *

a

---

22. All the records with the same ITEM-NUMBER have been grouped together and the groups are in ascending order. To process the records as specified in Figure 73, the computer must identify the point where one group ends and the next begins, or where the data in the control field changes. Figure 73 shows that this point is called a:

    a. control field.

    b. control break.

* * *

b

---

23. After each group of records has been processed, totals for that group are to be printed. At this point, control is to transfer to another portion of the program. A control break:

    a. occurs when the data in the control field changes.

    b. identifies the point at which control is to transfer to another portion of the program.

* * *

Both

---

24. In order to detect a control break, the control field of each record is compared to the control field of the previously read record. You might expect that this step would be performed by a(n) ..... statement.

* * *

IF

---

25. Each time a record is read, the data in the input area from the previously read record is destroyed. Therefore, in order to compare control fields, data in the control field from the previously read card must be:

    a. moved to a working-storage variable.

    b. saved in the input area.

* * *

a

26.

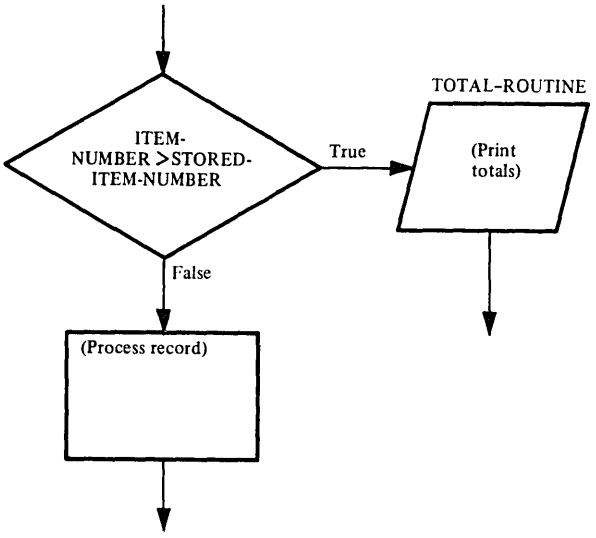


Figure 74

The flow chart segment above shows the sequencing for checking for a control break. Which of the following is true?

a. Because the records are in ascending sequence, a control break will occur when the ITEM-NUMBER read is greater than the ITEM-NUMBER previously read.

b. When a control break occurs, the record read is processed immediately.

* * *

a

---

27. Write the statement necessary to test for a control break according to the flow chart in the previous frame.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF ITEM-NUMBER GREATER THAN
               STORED-ITEM-NUMBER
               GO TO TOTAL-ROUTINE.
```

---

28. Records containing the total amount of sales for each salesman have been grouped by territory. The groups have been arranged in ascending sequence by territory number. When this file is processed, a control break will occur when the:

    a. salesman changes.

    b. territory number changes.

* * *

b

---

29. A condition that might be tested for a control break in the problem described in the preceding frame would be:

    a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SALESMAN-NUMBER GREATER THAN
               STORED-SALESMAN.
```

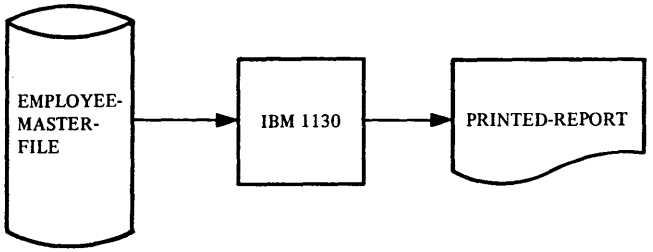
    b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           TERRITORY GREATER THAN
               STORED-TERRITORY.
```

* * *

b

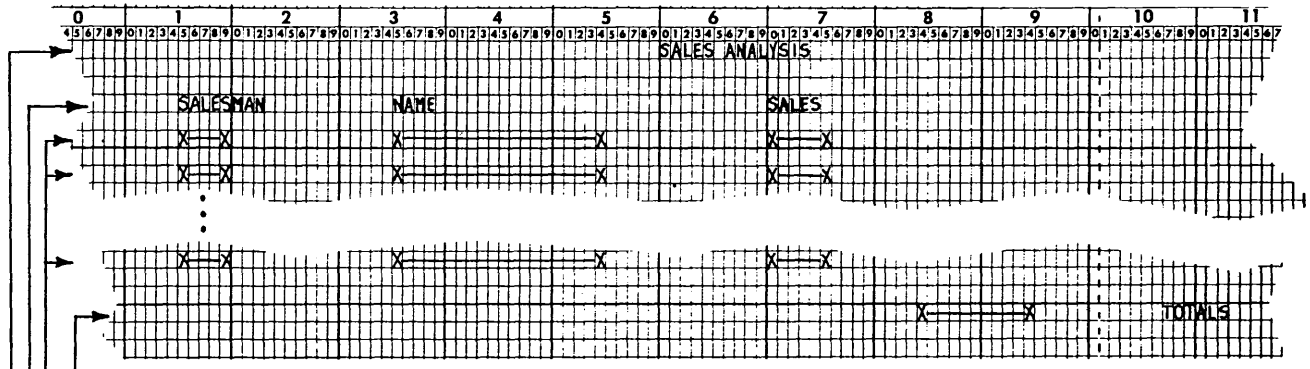
System flow chart for SALES-ANALYSIS



Input record

SALES-RECORD			
TERRITORY	SALESMAN-NUMBER	NAME	SALES

Output format



TOTAL-RECORD (triple spaced after last WORK-RECORD)
WORK-RECORD (double spaced)
HEADINGS (triple spaced)
TITLE-RECORD (at top of page)

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. SALES-ANALYSIS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT EMPLOYEE-MASTER-FILE
               ASSIGN TO DF-1-600.
           SELECT PRINTED-REPORT
               ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREA.
       DATA DIVISION.
       FILE SECTION.
       FD  EMPLOYEE-MASTER-FILE
           BLOCK CONTAINS 8 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  SALES-RECORD.
           02  TERRITORY PIC 99.
           02  SALESMAN-NUMBER PIC X(5).
           02  NAME PIC X(20).
           02  SALES PIC 9(4)V99.
       FD  PRINTED-REPORT
           LABEL RECORDS ARE OMITTED.
       01  PRINT-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       77  TERRITORY-STORED PIC 99 VALUE IS 99.
       77  TERRITORY-TOTAL PIC 9(5)V99
               VALUE IS ZEROS.
       77  FINAL-TOTAL PIC 9(7)V99
               VALUE IS ZEROS.
       77  TEST PIC 9 VALUE IS ZERO.
       01  TOTAL-RECORD.
           02  FILLER PIC X(85) VALUE IS SPACES.
           02  TOTALS PIC $$$$999.99.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  ID-WORD PIC X(6) VALUE IS 'TOTALS'.
           02  FILLER PIC X(9) VALUE IS SPACES.
       01  TITLE-RECORD.
           02  FILLER PIC X(60) VALUE IS SPACES.
           02  TITLE PIC X(14) VALUE IS
                   'SALES ANALYSIS'.
           02  FILLER PIC X(47) VALUE IS SPACES.
       01  HEADINGS.
           02  FILLER PIC X(15) VALUE IS SPACES.
           02  HEAD-1 PIC X(8) VALUE IS 'SALESMAN'.
           02  FILLER PIC X(12) VALUE IS SPACES.
           02  HEAD-2 PIC X(4) VALUE IS 'NAME'.
           02  FILLER PIC X(31) VALUE IS SPACES.
           02  HEAD-3 PIC X(5) VALUE IS 'SALES'.
           02  FILLER PIC X(46) VALUE IS SPACES.
       01  WORK-RECORD.
           02  FILLER PIC X(15) VALUE IS SPACES.
           02  SALESMAN-NUMBER PIC X(5).
           02  FILLER PIC X(15) VALUE IS SPACES.
           02  NAME PIC X(20).
           02  FILLER PIC X(15) VALUE IS SPACES.
           02  SALES PIC 9(4)V99.
           02  FILLER PIC X(54) VALUE IS SPACES.
```

Figure 75

30. As a programmer you may often be asked to prepare sales analysis reports. Figure 75 shows the system flow chart for a program named SALES-ANALYSIS. The forms of the input and output records are shown, along with the first three divisions of the program. Records containing the total amount of sales for each salesman have been grouped by territory. The groups have been arranged in ascending sequence by territory number. Each record will be listed on the 1132 printer. Totals will be printed for each territory, as well as a total for all sales, triple spaced after the total for the last territory. Follow the problem flow chart in Figure 76 and code the Procedure Division of SALES-ANALYSIS.

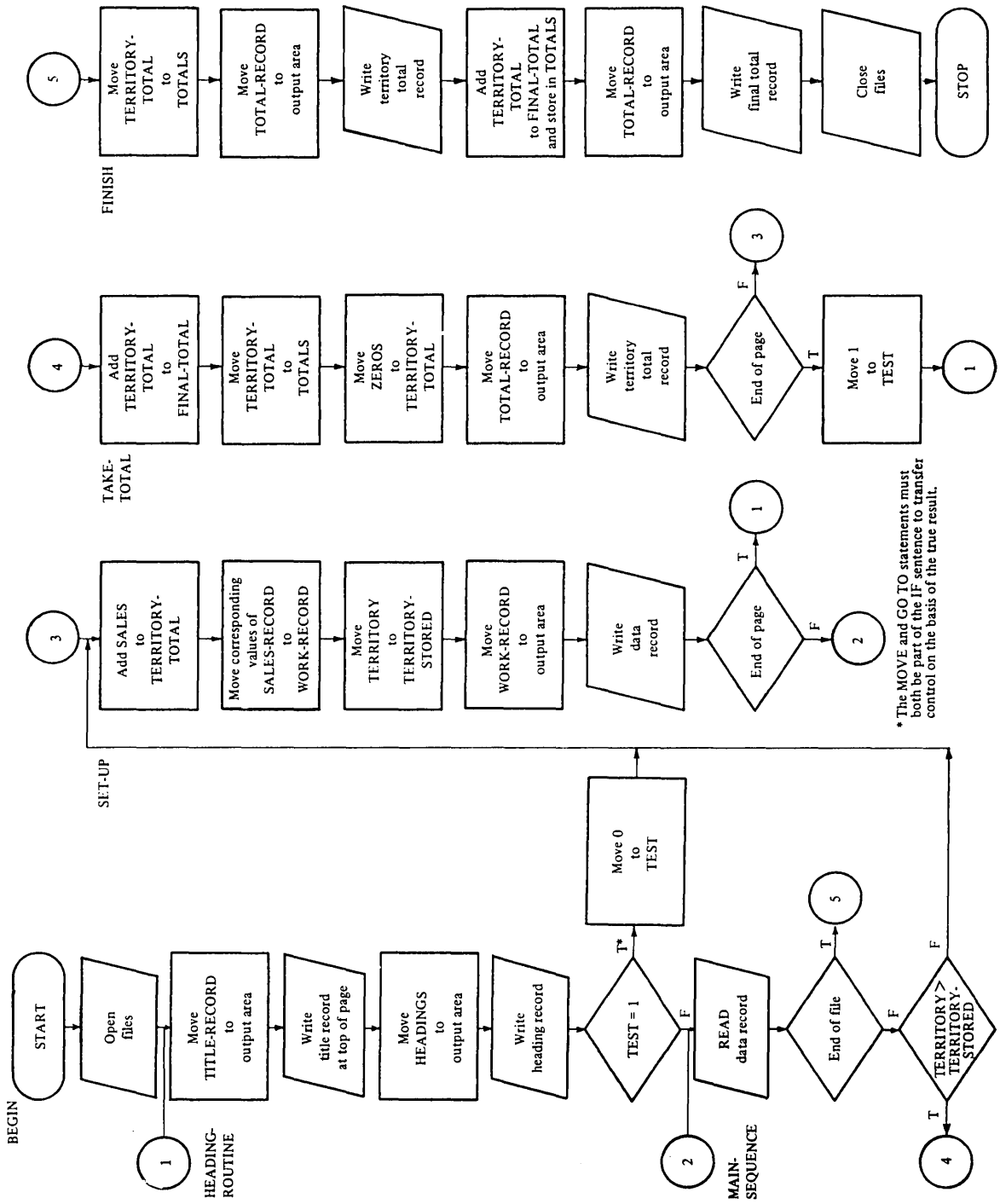

BEGIN
START
Open files
Move TITLE-RECORD to output area
Write title record at top of page
Move HEADINGS to output area
Write heading record
HEADING-ROUTINE
1
Move 0 to TEST
TEST = 1
T*
F
2
READ data record
End of file
T
5
F
TERRITORY > TERRITORY-STORED
F
T
4
MAIN-SEQUENCE
SET-UP
3
Add SALES to TERRITORY-TOTAL
Move corresponding values of SALES-RECORD to WORK-RECORD
Move TERRITORY to TERRITORY-STORED
Move WORK-RECORD to output area
Write data record
End of page
T
1
F
2
TAKE-TOTAL
4
Add TERRITORY-TOTAL to FINAL-TOTAL
Move TERRITORY-TOTAL to TOTALS
Move ZEROS to TERRITORY-TOTAL
Move TOTAL-RECORD to output area
Write territory total record
End of page
F
3
T
Move 1 to TEST
1
FINISH
5
Move TERRITORY-TOTAL to TOTALS
Move TOTAL-RECORD to output area
Write territory total record
Add TERRITORY-TOTAL to FINAL-TOTAL and store in TOTALS
Move TOTAL-RECORD to output area
Write final total record
Close files
STOP
* The MOVE and GO TO statements must both be part of the IF sentence to transfer control on the basis of the true result.

Figure 76

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT EMPLOYEE-MASTER-FILE
               OUTPUT PRINTED-REPORT.
       INITIALIZE.
           MOVE ZEROS TO TERRITORY-TOTAL.
       HEADING-ROUTINE.
           MOVE TITLE-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD.
           MOVE HEADINGS TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 4.
           IF TEST EQUAL TO 1
               MOVE ZERO TO TEST GO TO SET-UP.
       MAIN-SEQUENCE.
           READ EMPLOYEE-MASTER-FILE
               AT END GO TO FINISH.
           IF TERRITORY GREATER THAN                          (30)
               TERRITORY-STORED
               GO TO TAKE-TOTAL.
       SET-UP.
           ADD SALES OF SALES-RECORD
               TO TERRITORY-TOTAL.
           MOVE TERRITORY OF SALES-RECORD
               TO TERRITOEY OF WORK-RECORD.
           MOVE SALESMAN OF SALES-RECORD
               TO SALESMAN OF WORK-RECORD.
           MOVE NAME OF SALES-RECORD
               TO NAME OF WORK-RECORD.
           MOVE SALES OF SALES-RECORD
               TO SALES OF WORK-RECORD.
           MOVE TERRITORY TO TERRITORY-STORED.
           MOVE WORK-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 3 LINES.
               AT EOP GO TO HEADING-ROUTINE.
           GO TO MAIN-SEQUENCE.
       TAKE-TOTAL.
           ADD TERRITORY-TOTAL TO FINAL-TOTAL.
           MOVE TERRITORY-TOTAL TO TOTALS.
           MOVE ZEROS TO TERRITORY-TOTAL.
           MOVE TOTAL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 4 LINES.
               AT EOP MOVE 1 TO TEST
               GO TO HEADING-ROUTINE.
           GO TO SET-UP.
       FINISH.
           MOVE TERRITORY-TOTAL TO TOTALS.
           MOVE TOTAL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 4 LINES.
           ADD TERRITORY-TOTAL
               TO FINAL-TOTAL GIVING TOTALS.
           MOVE TOTAL-RECORD TO PRINT-RECORD.
           WRITE PRINT-RECORD
               AFTER ADVANCING 4 LINES.
           CLOSE EMPLOYEE-MASTER-FILE
               PRINTED-REPORT.
           STOP RUN.
```

If your solution appears different and you are not sure of its correctness, consult your advisor.

---

31. EXAMINE Statement. The EXAMINE statement is used to count the number of times a specified character appears in a data item and/or to replace a character with another character.

```
EXAMINE identifier TALLYING { UNTIL FIRST } literal-1
                            { ALL         }
                            { LEADING     }

        [REPLACING BY literal-2]
```

When the ALL is used an integral count is created which replaces the value of a special register TALLY. (See Language Considerations)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        EXAMINE AREA-1 TALLYING ALL 0.
```

If AREA-1 contains the number 101010 after the EXAMINE statement is executed, the special register TALLY will contain the integer 3. (There are 3 zeros in the AREA-1.)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        EXAMINE AREA-1 TALLYING ALL 1.
```

AREA-1 contains the number 110101. After the execution of the above statement the special register TALLY will contain:

a. 2

b. 4

* * *

b

EXAMINE statement	ITEM-1 (Before)	Data (After)	Resulting Value of TALLY
EXAMINE ITEM-1 TALLYING ALL 0	111010	111010	2
EXAMINE ITEM-1 TALLYING ALL 1 REPLACING BY 0	111010	000000	4
EXAMINE ITEM-1 REPLACING LEADING "*" BY SPACE	**7000	7000	+
EXAMINE ITEM-1 REPLACING FIRST "*" BY "$"	**1.94	$*1.94	+
+ unchanged			

SUMMARY:

You have learned to code and use the IF statement, in the simple form. In addition you have learned to identify control breaks in a file and to specify a certain action to be taken at the point where a control break occurs. The REPORT-WRITER feature of the COBOL language, which is not being taught in this course, greatly simplifies the problem of dealing with control breaks. If your installation is equipped for REPORT-WRITER, you may find it useful to read the section on REPORT-WRITER in your Language Specifications Manual after you have completed this course.

END OF LESSON 16

LESSON 17

LESSON 17 - DISK FILE UPDATING

INTRODUCTION

In this lesson you will write programs that require matching of records from two separate input files. You will learn to compare control fields, such as student number, from the two files in order to determine whether or not the records refer to the same student.

The technique of matching records is used frequently since changes are seldom required for every record in a master file. Some students will change majors in a particular year, for example, while many will not. Every customer does not buy something each month and every employee may not move or change his name before a file is updated.

The specific COBOL language feature that you will learn in this lesson is:

FROM option of the WRITE statement

This lesson will require approximately one half hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           READ EMPLOYEE-FILE
               AT END GO TO FINISH.
           MOVE EMPLOYEE-RECORD
               TO WORK-RECORD.
```

The first statement above would cause the input record EMPLOYEE-RECORD to be accessed from the file EMPLOYEE-FILE. The second statement would move the data in the input record to a working-storage variable called WORK-RECORD. You learned to write a single statement that would have the same effect as a READ statement and a MOVE statement. To review what you have already learned, write the single statement to produce the effect of the two statements shown above.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           READ EMPLOYEE-FILE
               INTO WORK-RECORD
               AT END GO TO FINISH.
```

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           READ INPUT-FILE INTO WORK-RECORD
               AT END GO TO STOP-ROUTINE.
```

The following statement is equivalent to the statement above.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           READ INPUT-FILE
               AT END GO TO STOP-ROUTINE.
           MOVE INPUT-RECORD
               TO WORK-RECORD.
```

---

3. A READ statement with the INTO option is used to transmit a record from an input file to:

   a. an input area associated with a previously opened file.

   b. a working-storage variable.

   c. an output area associated with a previously opened file.

* * *

b,c

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ EMPLOYEE-FILE
               AT END GO TO FINISH.
           MOVE EMPLOYEE-RECORD
               TO PAYROLL-RECORD.
           WRITE PAYROLL-RECORD.
```

Just as the INTO option of the READ statement is used to transmit a record from an input file to a working-storage variable or an output area associated with a previously opened file, the FROM option of the WRITE statement is used to transmit a record from a working-storage variable or an input area associated with a previously opened file to an output file. The statements above would transmit a record from the input file to the associated input area EMPLOYEE-RECORD, move the record from the input area to the output area PAYROLL-RECORD associated with the output file, and transmit the record from the output area to the output file. The WRITE statement below would have the same effect as the MOVE and WRITE statements above.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PAYROLL-RECORD
               FROM EMPLOYEE-RECORD.
```

Select the set of statements that would have the effect of transmitting a record from INPUT-FILE (associated with the input area INPUT-RECORD) to OUTPUT-FILE (associated with OUTPUT-AREA).

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE INTO OUTPUT-AREA
               AT END GO TO END-ROUTINE.
           WRITE OUTPUT-AREA.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           READ INPUT-FILE
               AT END GO TO END-ROUTINE.
           WRITE OUTPUT-AREA
               FROM INPUT-RECORD.
```

* * *

Either

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           WRITE OUTPUT-RECORD
               FROM WORK-RECORD.
```

The above statement would:

a. access a record and then move it to an output file.

b. move a record to an output area and then write it.

c. write a record and then move it to an output area.

* * *

b

6.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           WRITE OUTPUT-RECORD
               FROM WORK-RECORD.
```

In the statement above:

a. OUTPUT-RECORD is the output area associated with the output file.

b. WORK-RECORD may be an input area associated with a previously opened file or a working-storage variable.

* * *

Both

7.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           WRITE record-name [FROM identifier]
```

A general form of the WRITE statement is shown above. You know that record-name must be the output area associated with the output file. In the FROM option, identifier may be:

a. an input area associated with a previously opened file.

b. a working-storage variable.

* * *

Either
(Identifier must be a level 01 variable whether it is an input area or a working-storage variable.)

---

8. Write a statement to move data from the working-storage variable COMPUTATION-RECORD to the output area RESULT-PRINT and place the record in the output file PRINT-OUT.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           WRITE RESULT-PRINT
               FROM COMPUTATION-RECORD.
```

---

9. When data is moved into an output area using the FROM option, the move is made according to the rules for alphanumeric, or group moves. Which of the following rules apply to movement of data with the FROM option?

   a. Unused character positions in the receiving field are filled with spaces on the right.

   b. Excess character positions in the sending field are truncated on the right.

   c. Only corresponding elements are moved.

   d. The entire sending group is moved to the receiving group.

* * *

a,b,d

10.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PRINTOUT.
           02  FILLER PIC X(5).
           02  PART-NUMBER PIC X(7).
           02  FILLER PIC X(13).
           02  ON-HAND PIC X(3).
```

The output record PRINTOUT, shown above, is defined with FILLER items to provide for horizontal spacing.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           WRITE PRINTOUT FROM PARTS-RECORD.
```

In order for the WRITE statement above to provide the spacing indicated by the entry for PRINTOUT, PARTS-RECORD would be specified as:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PARTS-RECORD.
           02  PART-NUMBER PIC X(7).
           02  ON-HAND PIC X(3).
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PARTS-RECORD.
           02  FILLER PIC X(20).
           02  PART-NUMBER PIC X(7).
           02  FILLER PIC X(20).
           02  ON-HAND PIC X(3).
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  PARTS-RECORD.
           02  FILLER PIC X(5).
           02  CODE-ITEM PIC X(7).
           02  FILLER PIC X(13).
           02  QUANTITY PIC X(3).
           02  FILLER PIC X(25)
               VALUE IS SPACES.
```

* * *

c

11.

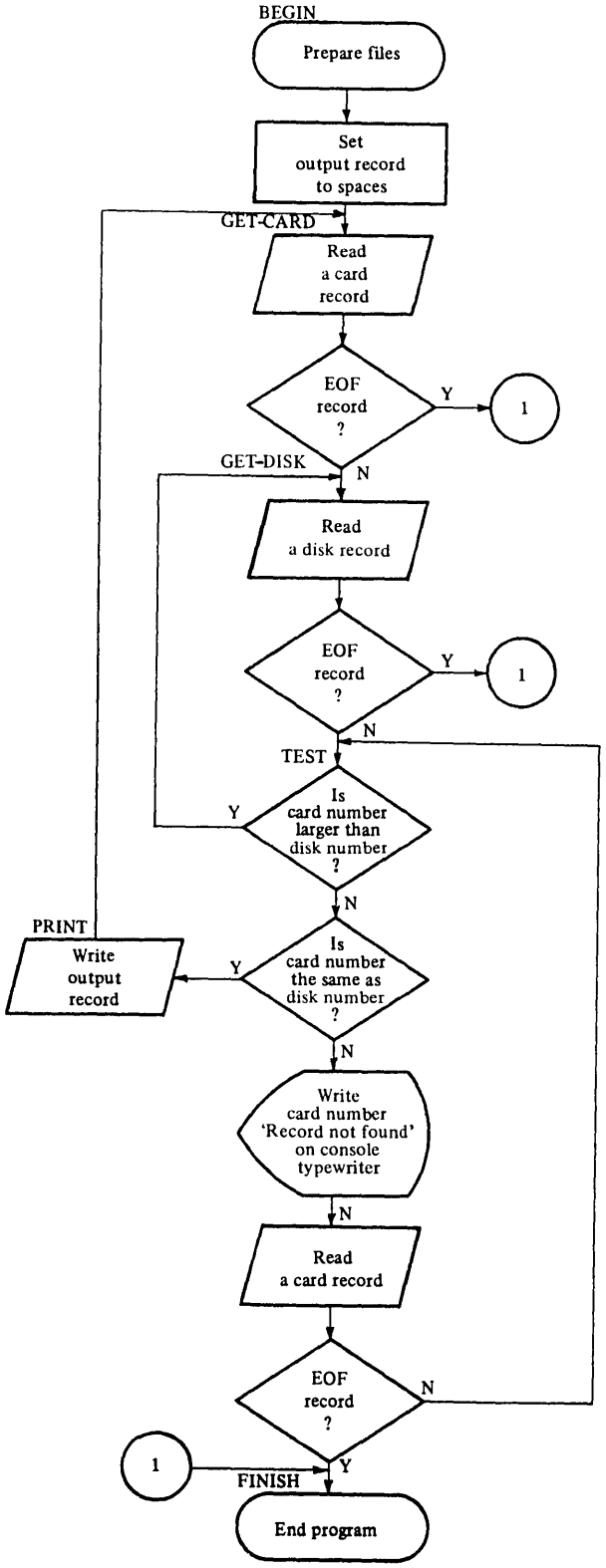
```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LOADTAPE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FREIGHT-CARD-FILE
               ASSIGN TO RD-1442.
           SELECT FREIGHT-CARD-OUT
               ASSIGN TO PU-1442.
       DATA DIVISION.
       FILE SECTION.
       FD  FREIGHT-CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  FREIGHT-LINE-RECORD.
           02  FREIGHT-LINE-NUMBER PIC X(6).
           02  FREIGHT-LINE-NAME PIC X(20).
           02  WEIGHT-RESTRICTIONS PIC X(12).
           02  FILLER PIC X(42).
       FD  FREIGHT-CARD-OUT
           LABEL RECORDS ARE OMITTED.
       01  FREIGHT-RECORD.
           02  LINE-NUMBER PIC X(6).
           02  LINE-NAME PIC X(20).
           02  RESTRICTION PIC X(12).
           02  RATING PIC X(2).
```

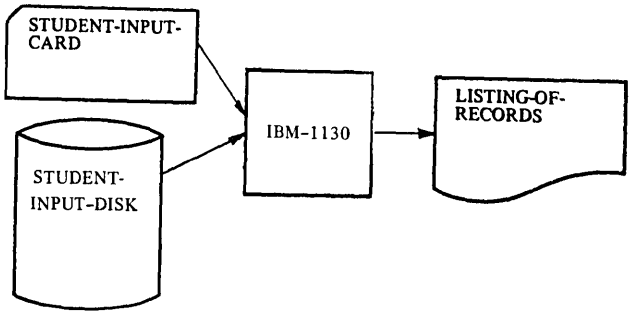
The first three divisions of a card-in-card-out program are shown above. The FILLER item on each input card contains spaces. The last two characters in the output area are to be spaces since the rating will be added later by another program. Write a Procedure Division that will read the data on the cards and punch them out again. Use the FROM option whenever possible.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT FEIGHT-CARD-FILE
               OUTPUT FREIGHT-CARD-OUT.
       CARD-TO-DISK.
           READ FREIGHT-CARD-FILE
               AT END GO TO FINISH.
           WRITE FREIGHT-RECORD
               FROM FREIGHT-LINE-RECORD.
           GO TO CARD-TO-DISK.
       FINISH.
           CLOSE FREIGHT-CARD-FILE
               FREIGHT-DISK-FILE.
           STOP RUN.
```

The following problem incorporates many of the COBOL features you have learned up to this point. Since you are not asked to do anything new in this problem, it is optional for you to code it. If you choose not to code the problem, be sure to read it carefully to make certain you understand it.

12. As a programmer for a university data-processing division, you have been asked to write a program to make a listing of the master disk entries for certain students. You have been given a deck of cards. The number of each student for whom a disk record is to be listed has been punched into a card. The last card, like the last record, is a dummy containing the number 999999999. Figure 77 gives a program flow chart, a system flow chart, and the first three divisions of a program to create the output file. Write the Procedure Divsion, using the FROM option. Remember to include OPEN and CLOSE statements when necessary.





System flow chart

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. LIST-RECORDS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENT-INPUT-CARD
               ASSIGN TO RD-1442.
           SELECT STUDENT-INPUT-DISK
               ASSIGN TO DF-2-100-X.
           SELECT LISTING-OF-RECORDS
               ASSIGN TO PR-1132-C-PUTOUT
               RESERVE NO ALTERNATE AREA.
       DATA DIVISION.
       FILE SECTION.
       FD  STUDENT-INPUT-CARD
           LABEL RECORDS ARE OMITTED.
       01  CARD-STUDENT-DATA.
           02  CARD-NUMBER PIC X(9).
           02  FILLER PIC X(71).
       FD  STUDENT-INPUT-DISK
           BLOCK CONTAINS 7 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  DISK-DATA.
           02  FILLER PIC X.
           02  DISK-NUMBER PIC X(9).
           02  NAME PIC X(20).
           02  FILLER PIC X(90).
       FD  LISTING-OF-RECORDS
           LABEL RECORDS ARE OMITTED.
       01  PRINT-LINE PIC X(121).
```

Figure 77

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT-INPUT-CARD
               STUDENT-INPUT-DISK
               OUTPUT LISTING-OF-RECORDS.
           MOVE SPACES TO PRINT-LINE.
       GET-CARD.
           READ STUDENT-INPUT-CARD
               AT END GO TO FINISH.
       GET-DISK.
           READ STUDENT-INPUT-DISK
               AT END GO TO FINISH.
       TEST.
           IF CARD-NUMBER IS GREATER THAN
               DISK-NUMBER
               GO TO GET-DISK.
           IF CARD-NUMBER IS EQUAL TO
               DISK-NUMBER
               GO TO PRINT.
           DISPLAY CARD-NUMBER
               'RECORD NOT FOUND' UPON CONSOLE.
           READ STUDENT-INPUT-CARD
               AT END GO TO FINISH.
           GO TO TEST.
       PRINT.
           WRITE PRINT-LINE
               FROM DISK-DATA.
           GO TO GET-CARD.
       FINISH.
           CLOSE STUDENT-INPUT-CARD
               STUDENT-INPUT-DISK
               LISTING OF RECORDS.
           STOP RUN.
```

(The reserved word IS is used here in the IF statements. It is an optional word and may be omitted.)

---

13. The procedure in the previous frame is an example of a problem that requires matching records from two input files. Look at the program flow chart in Figure 77 again. What would be the three possible results of comparison of CARD-NUMBER and DISK-NUMBER?

* * *

CARD-NUMBER greater than DISK-NUMBER
CARD-NUMBER equal to DISK-NUMBER
CARD-NUMBER smaller than DISK-NUMBER

---

14. The appropriate action must be taken for each possible result of a comparison. Referring to Figure 77, match the following courses of action with the result of comparisons.

1) CARD-NUMBER greater than DISK-NUMBER	a. Read another disk record and then compare again.
2) CARD-NUMBER equal to DISK-NUMBER	b. Print an error message, read another card, and then compare again.
3) CARD-NUMBER smaller than DISK-NUMBER	c. Write the output record, read a record from each input file, and then compare again.
	d. Close the files and stop execution of the program.

* * *

1) a
2) c
3) b

---

15. The program illustrated in Figure 77 is really a simplified version of matching records, since no new master file is being created. This program is merely accessing specific records from the old master file, leaving it intact for further use. If the card file contained only records of students to be added to the old master disk while creating a new disk, the new master disk would contain:

    a. the records on the old disk only.

    b. the records from the card file only.

    c. the records from both the old disk and the card file.

* * *

c

---

16. In order to match records from two input files, the records in both files must be in the same sequence by the control field. If a card file contains records to be added to a disk file, you would compare a card number to a disk number on each pass through the main routine of your program. If both files are in ascending sequence and the card number is larger than the disk number, the program should:

    a. read another disk record immediately.

    b. read another card record immediately.

    c. write the disk record on a new disk and then read another disk record.

* * *

c

---

17. Assume for the moment that you have the two input files used in Figure 77. The card file contains the numbers of the students for whom records are to be removed from a master disk file. The last card record and the last disk record are dummy records of 999999999 to ensure that both files will be completely processed. You are to decide on the logic required to create a new master disk that will not include the records of the students for whom cards are present. Which flow chart describes such a problem?

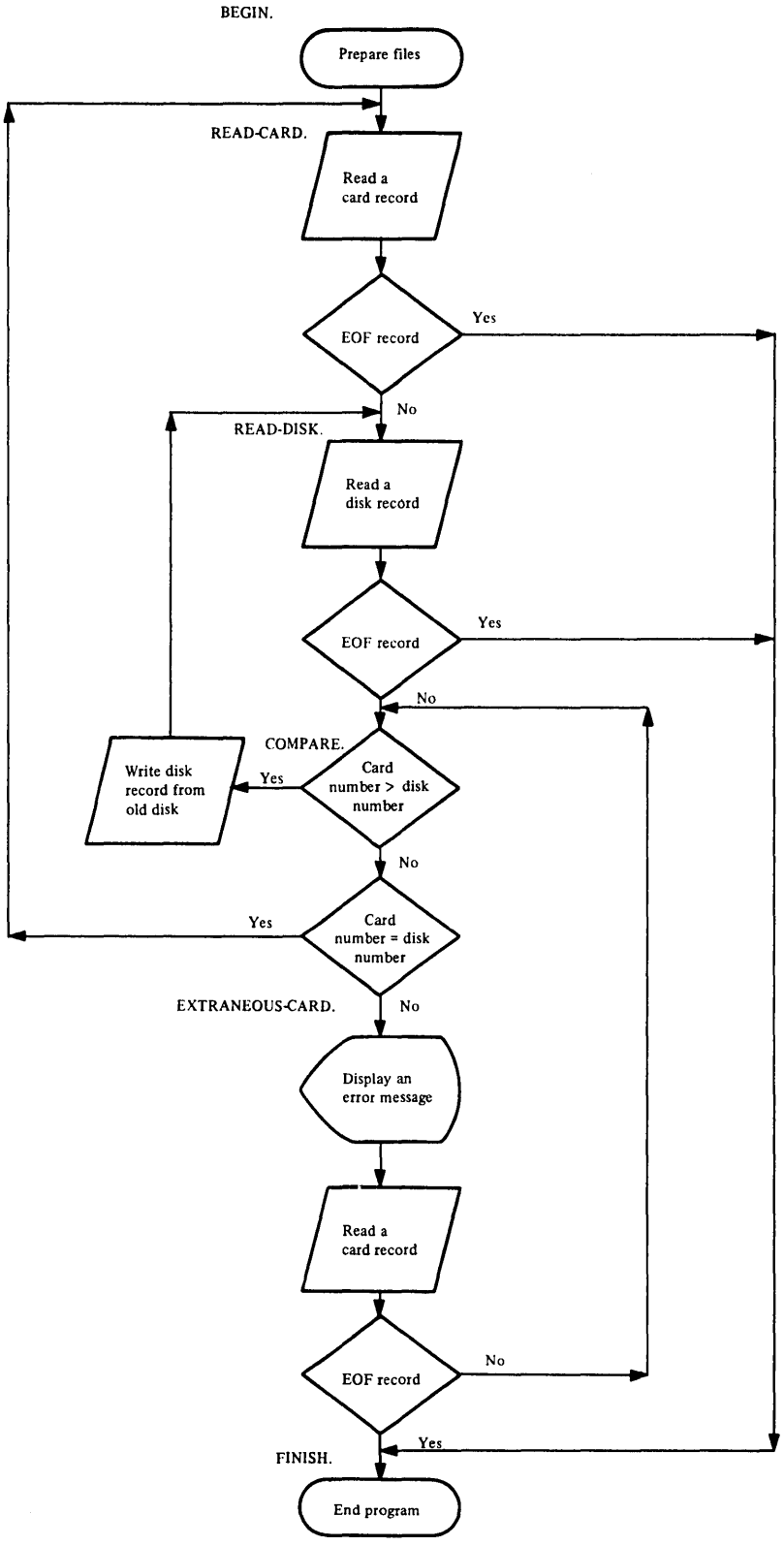

BEGIN.
Prepare files
READ-CARD.
Read a card record
EOF record
Yes
No
READ-DISK.
Read a disk record
EOF record
Yes
No
COMPARE.
Write disk record from old disk
Yes
Card number > disk number
No
Yes
Card number = disk number
EXTRANEOUS-CARD.
No
Display an error message
Read a card record
EOF record
No
Yes
FINISH.
End program

Figure 78

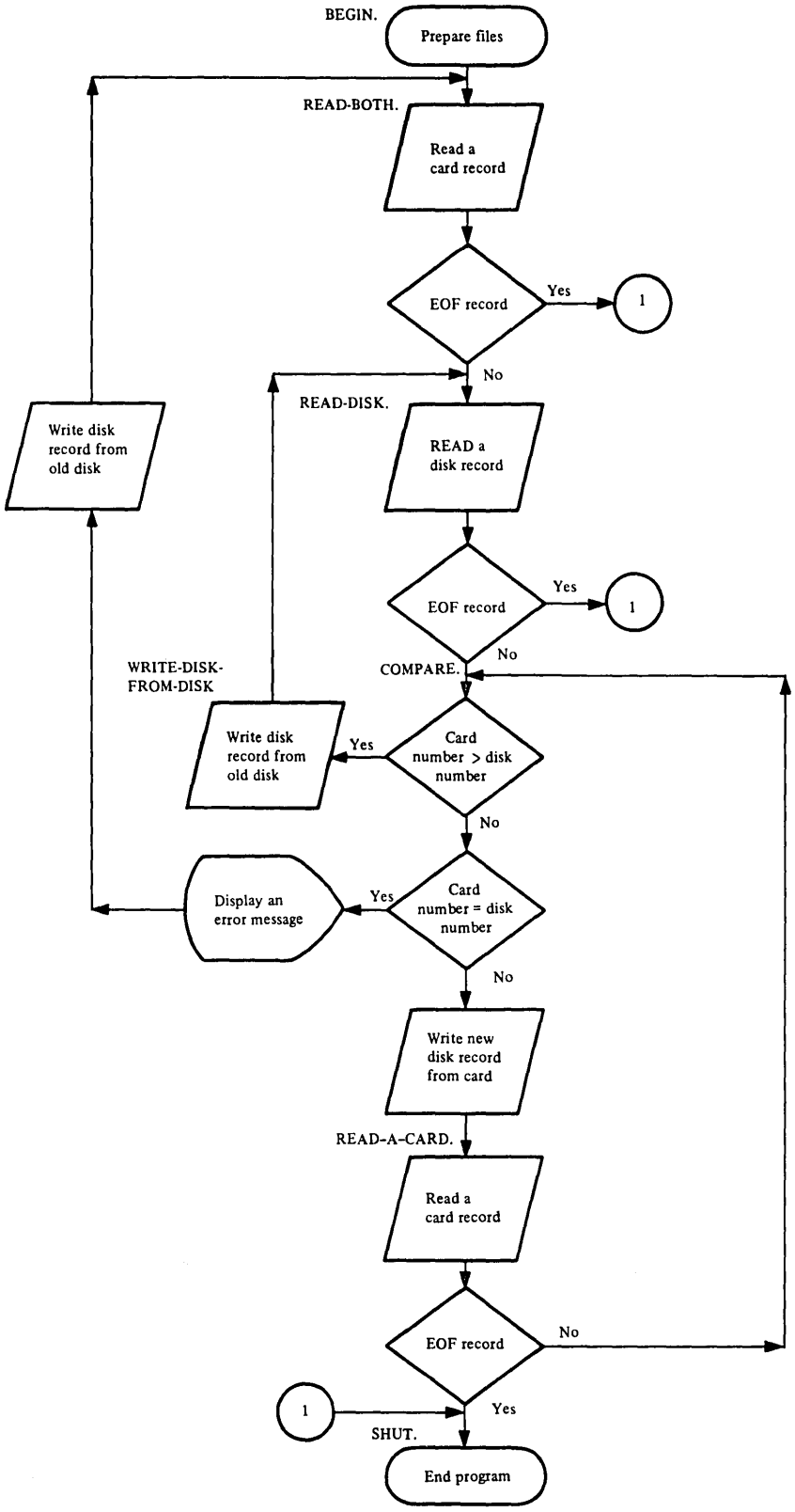
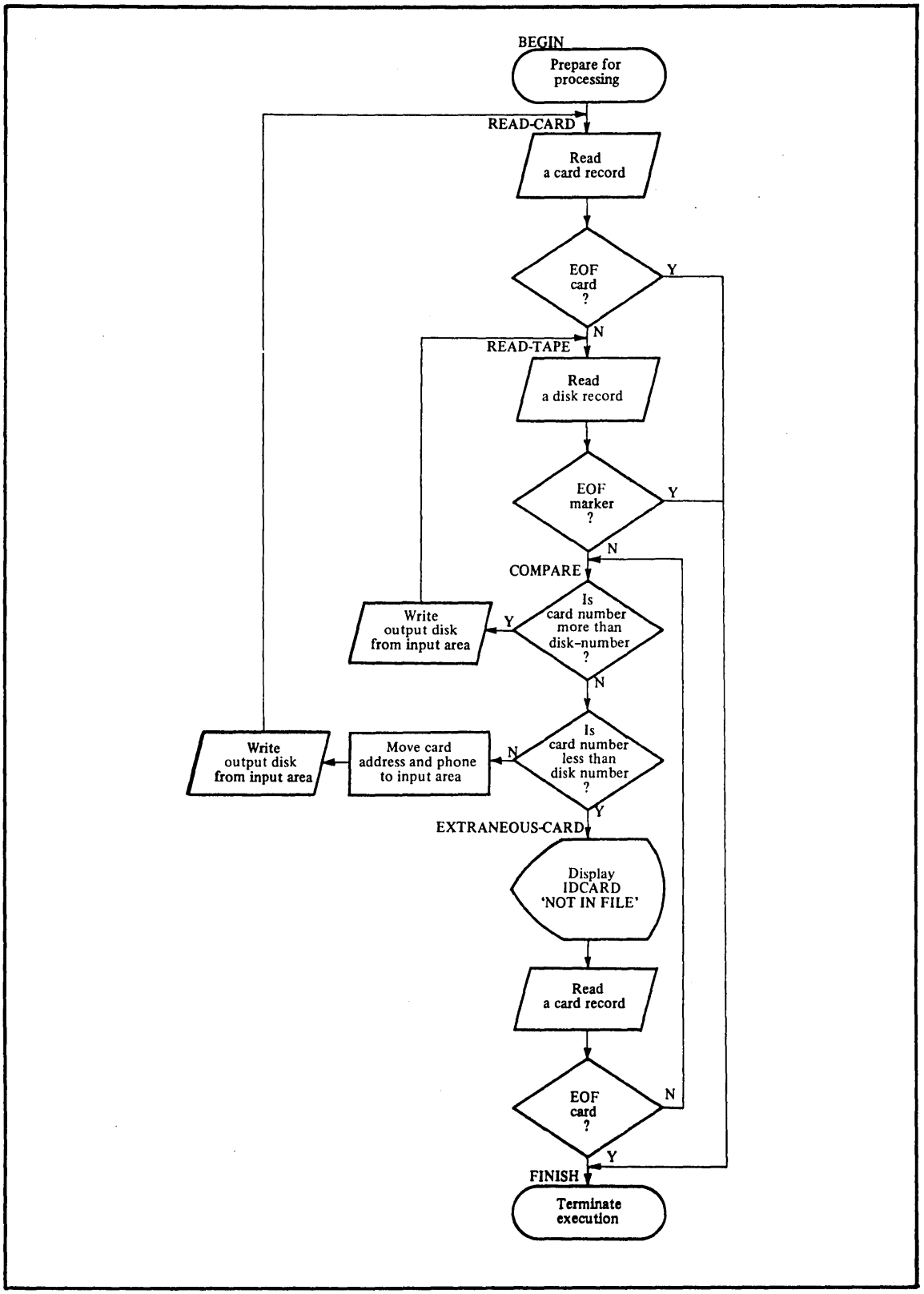


Figure 79

* * *

a
(Flow chart b would be used to add records from the card file to the new master disk file rather than remove them.)


BEGIN
Prepare for processing
READ-CARD
Read a card record
EOF card ?
Y
N
READ-TAPE
Read a disk record
EOF marker ?
Y
N
COMPARE
Is card number more than disk-number ?
Y
Write output disk from input area
N
Is card number less than disk number ?
N
Move card address and phone to input area
Write output disk from input area
Y
EXTRANEOUS-CARD
Display IDCARD 'NOT IN FILE'
Read a card record
EOF card ?
N
Y
FINISH
Terminate execution

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. REMOVALS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENT-INPUT-CARD
               ASSIGN TO RD-1442.
           SELECT STUDENT-INPUT-DISK
               ASSIGN TO DF-3-500.
           SELECT NEW-MASTER-DISK
               ASSIGN TO DF-1-400.
       DATA DIVISION.
       FILE SECTION.
       FD  STUDENT-INPUT-CARD
           LABEL RECORDS ARE OMITTED.
       01  CARD-RECORD.
           02  IDCARD PIC X(9).
           02  HANGOUT PIC X(30).
           02  PHONE PIC X(8).
           02  FILLER PIC X(33).
       FD  STUDENT-INPUT-DISK
           BLOCK CONTAINS 7 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  DISK-RECORD-IN.
           02  STUDENT-NUMBER PIC X(9).
           02  STUDENT-NAME PIC X(25).
           02  STUDENT-ADDRESS.
               03  STREET PIC X(15).
               03  CITY PIC X(10).
               03  STATE PIC X(5).
           02  STUDENT-PHONE PIC X(8).
           02  SCHOLASTIC-DATA PIC X(51).
       FD  NEW-MASTER-DISK
           BLOCK CONTAINS 7 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  DISK-RECORD-OUT.
           02  STUDENT-NUMBER-O PIC X(9).
           02  STUDENT-NAME-O PIC X(25).
           02  STUDENT-ADDRESS-O PIC X(30).
           02  STUDENT-PHONE-O PIC X(8).
           02  SCHOLASTIC-DATA-O PIC X(51).
```

Figure 80

18. Figure 80 shows the first three divisions of a program and the program flow chart for a program to update a disk file by inserting new addresses and phone numbers for students who have moved. A new disk will be created that will incorporate the data from the card file into the records from the existing disk file. Follow the program flow chart and write the Procedure Division to update the disk file, using file and variable names from the program segment in Figure 80.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT-INPUT-CARD
                STUDENT-INPUT-DISK
                OUTPUT NEW-MASTER-DISK.
       READ-CARD.
           READ STUDENT-INPUT-CARD
                AT END GO TO FINISH.
       READ-DISK.
           READ STUDENT-INPUT-DISK
                AT END GO TO FINISH.
       COMPARE.
           IF IDCARD IS GREATER THAN
                STUDENT-NUMBER
                WRITE DISK-RECORD-OUT
                FROM DISK-RECORD-IN
                GO TO READ-DISK.
           IF IDCARD IS LESS THAN
                STUDENT-NUMBER
                GO TO EXTRANEOUS-CARD.
           MOVE HANGOUT TO STUDENT-ADDRESS.
           MOVE PHONE TO STUDENT-PHONE.
           WRITE DISK-RECORD-OUT
                FROM DISK-RECORD-IN.
           GO TO READ-CARD.
       EXTRANEOUS-CARD.
           DISPLAY IDCARD 'NOT IN FILE'
                UPON CONSOLE.
           READ STUDENT-INPUT-CARD
                AT END GO TO FINISH.
           GO TO COMPARE.
       FINISH.
           CLOSE STUDENT-INPUT-CARD
                STUDENT-INPUT-DISK
                NEW-MASTER-DISK.
           STOP RUN.
```

SUMMARY:

In addition to matching records, you have learned to use the FROM option of the WRITE statement as a substitute for writing a MOVE statement. You have also seen a few of the many reasons for matching records from two input files.

END OF LESSON 17

LESSON 18

LESSON 18 - DISK FILE PROCESSING

INTRODUCTION

At the end of this lesson you will write a program to insert additional records into the proper places in a master disk file and another to remove specific records from the disk file. In these same programs you will be processing records for students transferring to and from a college but as you will observe, the same technique would be used for customer, stock items, or employees.

The COBOL language features you will learn in this lesson are not limited to matching-record problems. They are, rather, useful features that can save you time, result in more efficient programs, or accomplish an arithmetic operation.

The specific COBOL language features that you will learn in this lesson are:

PERFORM statement
SUBTRACT statement

This lesson will require approximately three quarters of an hour. The optional problems will require one additional hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           PERFORM GET-CARD.
```

The statement shown above is called a PERFORM statement. It causes control to transfer to the specified step in a program just as the GO TO statement does. The PERFORM statement, however, will cause control to be returned to the statement following it after the specified paragraph is executed. Execution of the above statement will cause:

a. execution of the GET-CARD paragraph and return of control to the statement following the PERFORM statement.

b. execution of the GET-CARD paragraph and transfer of control to the statement following the GET-CARD paragraph.

* * *

a
(b would be the effect of execution of

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           GO TO GET-CARD.
```

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       COMPARE.
           IF IDCARD IS GREATER THAN
               STUDENT-NUMBER
               WRITE DISK-RECORD-OUT
               FROM DISK-RECORD-IN
               GO TO READ-DISK.
           IF IDCARD IS LESS THAN
               STUDENT-NUMBER
               GO TO EXTRANEOUS-CARD.
           MOVE HANGOUT TO STUDENT-ADDRESS.
           MOVE PHONE TO STUDENT-PHONE.
           WRITE DISK-RECORD-OUT
               FROM DISK-RECORD-IN.
           GO TO READ-CARD.
       EXTRANEOUS-CARD.
           DISPLAY IDCARD 'NOT IN FILE'
               UPON CONSOLE.
           READ STUDENT-INPUT-CARD
               AT END GO TO FINISH.
           GO TO COMPARE.
       FINISH.
           CLOSE STUDENT-INPUT-CARD
               STUDENT-INPUT-DISK
               NEW-MASTER-DISK
           STOP RUN.
```

In the Procedure Division segment above, which you wrote in a preceding frame, the same WRITE statament is written twice in paragraph COMPARE.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WRITE-DISK.
           WRITE DISK-RECORD-OUT
               FROM DISK-RECORD-IN.
```

If paragraph WRITE-DISK were added to the segment above just prior to paragraph FINISH, each WRITE statement in paragraph COMPARE could be replaced by:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           GO TO WRITE-DISK.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           PERFORM WRITE-DISK.
```

* * *

b
(If statement a were used to replace the WRITE statement, control would pass from WRITE-DISK to FINISH, terminating the procedure.)

---

3. Figure 81 shows a Procedure Division that prints headings on each page and then displays a message to the operator before listing the records. Rewrite the WRITE statement in paragraph LISTING-ROUTINE so that the headings will be printed on each overflow page. An initialization message will be displayed only at the beginning of the program because of coding contained in the paragraph SIGNAL-OPERATOR shown in Figure 81.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN-ROUTINE.
           OPEN INPUT CUSTOMER-FILE
                OUTPUT PRINT-FILE.
       HEADING-LINE.
           MOVE HEADINGS TO PRINT-RECORD.
           WRITE PRINT-RECORD.
       SIGNAL-OPERATOR.
           DISPLAY 'PRINTOUT HAS BEGUN'.
       LISTING-ROUTINE.
           READ CUSTOMER-FILE
                AT END GO TO FINISH.
           MOVE CUST-NO TO CUSTOMER-NO.
           MOVE CUST-AMT TO CUSTOMER-AMT.
           WRITE PRINT-RECORD
                FROM LIST-RECORD
                AFTER ADVANCING 2.
                AT EOP GO TO HEADING-LINE.
           GO TO LISTING-ROUTINE.
       FINISH.
           CLOSE CUSTOMER-FILE
                PRINT-FILE.
           STOP RUN.
```

Figure 81

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE PRINT-RECORD FROM LIST-RECORD
                AFTER ADVANCING 2
                AT EOP PERFORM HEADING-LINE.
```

---

4. Match the effects with the statement types.

1) PERFORM	a. transfers control to the specified paragraph, executes it, then returns control to the statement following the cause of the transfer.
2) GO TO	b. transfers control to the specified paragraph, executes it, then continues with the next paragraph.
	c. transfers control to the specified paragraph, executes the first statement, then returns control to the statement following the cause of the transfer.

* * *

1) a
2) b

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           PERFORM paragraph-name.
```

A paragraph that is specified in this form of a PERFORM statement may not include a GO TO statement but may include a PERFORM statement. Select the paragraph that could be specified by the statement below.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           PERFORM PARAGRAPH.
```

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PARAGRAPH.
           READ CARD-FILE
               AT END GO TO HALT.
           IF CODE-SYMBOL IS EQUAL TO
               CODE-RECORD
               PERFORM ADD-ROUTINE.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PARAGRAPH.
           READ CARD-FILE
               AT END PERFORM HALT.
           IF CODE-SYSTEM IS EQUAL TO
               CODE-RECORD
               PERFORM ADD-ROUTINE.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       PARAGRAPH.
           ADD NUMBER-DELIVERED TO
               NUMBER-ON-HAND.
           GO TO PRINTOUT.
```

* * *

b

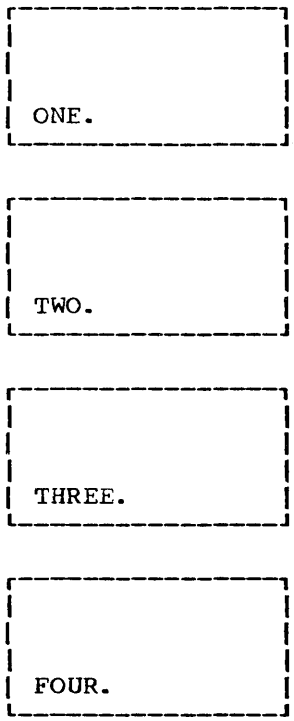
---

6. Write a statement to transfer control to a paragraph named ERROR-HANDLING, and then return control to the statement following the statement you write.

* * *

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           PERFORM ERROR-HANDLING.
```

7.



As a programmer you wish to execute the paragraphs represented above in a different sequence, such as ONE, TWO, ONE, THREE, FOUR. To produce this sequence the statement

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           PERFORM ONE.
```

could be:

a. the last statement in TWO.

b. a statement in a separate paragraph between TWO and THREE.

* * *

Either

8.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           .
           .
           .
           WRITE PRINT
               AT EOP GO TO PAGES.
       AFTER-PAGES.
           .
           .
           .
       PAGES.
           ADD 1 TO PAGE-NUMBER.
           .
           .
           .
           GO TO AFTER-PAGES.
```

Write the segment above to use the PERFORM statement in the EOP option.

```
                          *         *         *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           .
           .
           .
           WRITE PRINT
               AT EOP PERFORM PAGES.
       AFTER-PAGES.
           .
           .
           .
       PAGES.
           ADD 1 TO PAGE-NUMBER.
           .
           .
           .
```

(The GO TO statement must be removed from paragraph PAGES.)

9.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           PERFORM TAKE-TOTAL.
```

The PERFORM statement is useful when control must be transferred to a group of statements from several points in a program. If the statement above appears in two different paragraphs in a program, the point to which control is returned after execution of the PERFORM statement will:

a. be the same in each case.

b. depend on the location of the PERFORM statement.

c. be the statement following the PERFORM that was executed.

* * *

b,c

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. BILLING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       SPECIAL-NAMES. C01 IS TO-NEXT-PAGE.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT TRANSACTION-FILE
               ASSIGN TO RD-1442.
           SELECT CUSTOMER-FILE
               ASSIGN TO DF-1-700-X.
           SELECT BILL-FILE
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE SECTION.
       FD  TRANSACTION-FILE
           LABEL RECORDS ARE OMITTED.
       01  ITEM-RECORD.
           02  CUSTOMER-NUMBER PIC X(5).
           02  ITEM-NUMBER PIC X(5).
           02  UNIT-PRICE PIC 999V99.
           02  ITEM-DESCRIPTION PIC X(20).
           02  QUANTITY PIC 999.
           02  FILLER PIC X(42).
       FD  CUSTOMER-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 6 RECORDS.
       01  CUSTOMER-RECORD.
           02  FILLER PIC X.
           02  CHARGE-ID PIC X(5).
           02  NAME PIC X(30).
           02  STREET PIC X(20).
           02  MAILING
               03  CITY PIC X(15).
               03  STATE PIC X(15).
               03  ZIP PIC X(5).
           02  FILLER PIC X(20).
       FD  BILL-FILE
           LABEL RECORDS ARE OMITTED.
       01  BILL-PRINT PIC X(63).
       WORKING-STORAGE SECTION.
       77  SAVE-NUMBER PIC X(5).
       77  SUB-AMOUNT PIC 9999V99.
       77  TOTAL-AMOUNT PIC 99999V99
               VALUE IS ZEROS.
       77  SPACING PIC X.
       77  STATE-CODE PIC X.
       01  ITEM-LINE.
           02  FILLER PIC X(4) VALUE IS SPACES.
           02  NUMBER-O PIC X(5).
           02  FILLER PIC X(4) VALUE IS SPACES.
           02  ITEM PIC X(20).
           02  FILLER PIC X(4) VALUE IS SPACES.
           02  PRICE-O PIC $999.99.
           02  FILLER PIC X(4) VALUE IS SPACES.
           02  QUANTITY-O PIC 999.
           02  FILLER PIC X(4) VALUE IS SPACES.
           02  AMOUNT PIC $9999.99.
```

```
01  PRINT-TOTAL.
    02  FILLER PIC X(52) VALUE IS SPACES.
    02  TOTAL PIC $99999.99.
01  DISCOUNT-TOTAL.
    02  FILLER PIC X(18) VALUE IS SPACES.
    02  COMMENT-1 PIC X(18)
            VALUE 'MINUS .05 DISCOUNT'.
    02  FILLER PIC X(18) VALUE IS SPACES.
    02  DISCOUNT-AMOUNT PIC $999.99.
01  TAX-TOTAL.
    02  FILLER PIC X(18) VALUE IS SPACES.
    02  COMMENT-2 PIC X(12)
            VALUE 'PLUS .04 TAX'.
    02  FILLER PIC X(24) VALUE IS SPACES.
    02  TAX-AMOUNT PIC $999.99.
01  NAME-LINE.
    02  FILLER PIC X(10) VALUE IS SPACES.
    02  NAME-OUT PIC X(30).
01  STREET-LINE.
    02  FILLER PIC X(10) VALUE IS SPACES.
    02  STREET-OUT PIC X(20).
01  ADDRESS-LINE.
    02  FILLER PIC X(10) VALUE IS SPACES.
    02  ADDRESS-OUT PIC X(35).
01  END-LINE.
    02  FILLER PIC X(10) VALUE IS SPACES.
    02  COMMENT-3 PIC X(25)
            VALUE IS 'PLEASE PAY WITHIN 30 DAYS'.
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT TRANSACTION-FILE
        CUSTOMER-FILE
        OUTPUT BILL-FILE.
INPUT-ROUTINE.
    READ TRANSACTION-FILE
        AT END GO TO HALT.
    MOVE CUSTOMER-NUMBER TO SAVE-NUMBER.
MATCH.
    READ CUSTOMER-FILE
        AT END GO TO HALT.
    IF CUSTOMER-NUMBER GREATER THAN
        CHARGE-ID GO TO MATCH.
    IF CUSTOMER-NUMBER LESS THAN
        CHARGE-ID GO TO MESSAGE-HALT.
WRITE-MAILING-DATA.
    MOVE NAME TO NAME-OUT.
    WRITE BILL-PRINT FROM NAME-LINE
        AFTER ADVANCING TO-NEXT-PAGE.
    MOVE STREET TO STREET-OUT.
    WRITE BILL-PRINT FROM STREET-LINE
        AFTER ADVANCING 2 LINES.
    IF STATE EQUAL TO 'MICHIGAN'
        MOVE 1 TO STATE-CODE
        ELSE MOVE 0 TO STATE-CODE.
    MOVE MAILING TO ADDRESS-OUT.
    WRITE BILL-PRINT FROM ADDRESS-LINE
        AFTER ADVANCING 2 LINES.
    MOVE 6 TO SPACING.
```

```
WRITE-ITEM-LINE.
    MULTIPLY UNIT-PRICE BY QUANTITY
        GIVING SUB-AMOUNT.
    MOVE SUB-AMOUNT TO AMOUNT.
    ADD SUB-AMOUNT TO TOTAL-AMOUNT.
    MOVE ITEM-NUMBER TO NUMBER-0.
    MOVE ITEM-DESCRIPTION TO ITEM-0.
    MOVE UNIT-PRICE TO PRICE-0.
    MOVE QUANTITY TO QUANTITY-0.
    WRITE BILL-PRINT FROM ITEM-LINE
        AFTER ADVANCING SPACING LINES.
CHECK-NEXT-CARD.
    READ TRANSACTION-FILE
        AT END GO TO END-ROUTINE.
    IF CUSTOMER-NUMBER EQUAL TO
        SAVE-NUMBER MOVE 1 TO SPACING
        GO TO WRITE-ITEM-LINE.
    MOVE CUSTOMER-NUMBER TO SAVE-NUMBER.
    PERFORM CALCULATIONS.
    PERFORM ADVANCE.
    GO TO MATCH.
MESSAGE-HALT.
    DISPLAY CUSTOMER-NUMBER
        'NOT FOUND IN FILE.'
        UPON CONSOLE.
    GO TO HALT.
END-ROUTINE.
    PERFORM CALCULATIONS.
    WRITE BILL-PRINT FROM END-LINE
        AFTER ADVANCING 3 LINES.
HALT.
    CLOSE TRANSACTION-FILE BILL-FILE
        CUSTOMER-FILE.
    STOP RUN.
ADVANCE.
    WRITE BILL-PRINT FROM END-LINE
        AFTER ADVANCING 3 LINES.
    MOVE ZEROS TO TOTAL-AMOUNT.
```

Figure 82

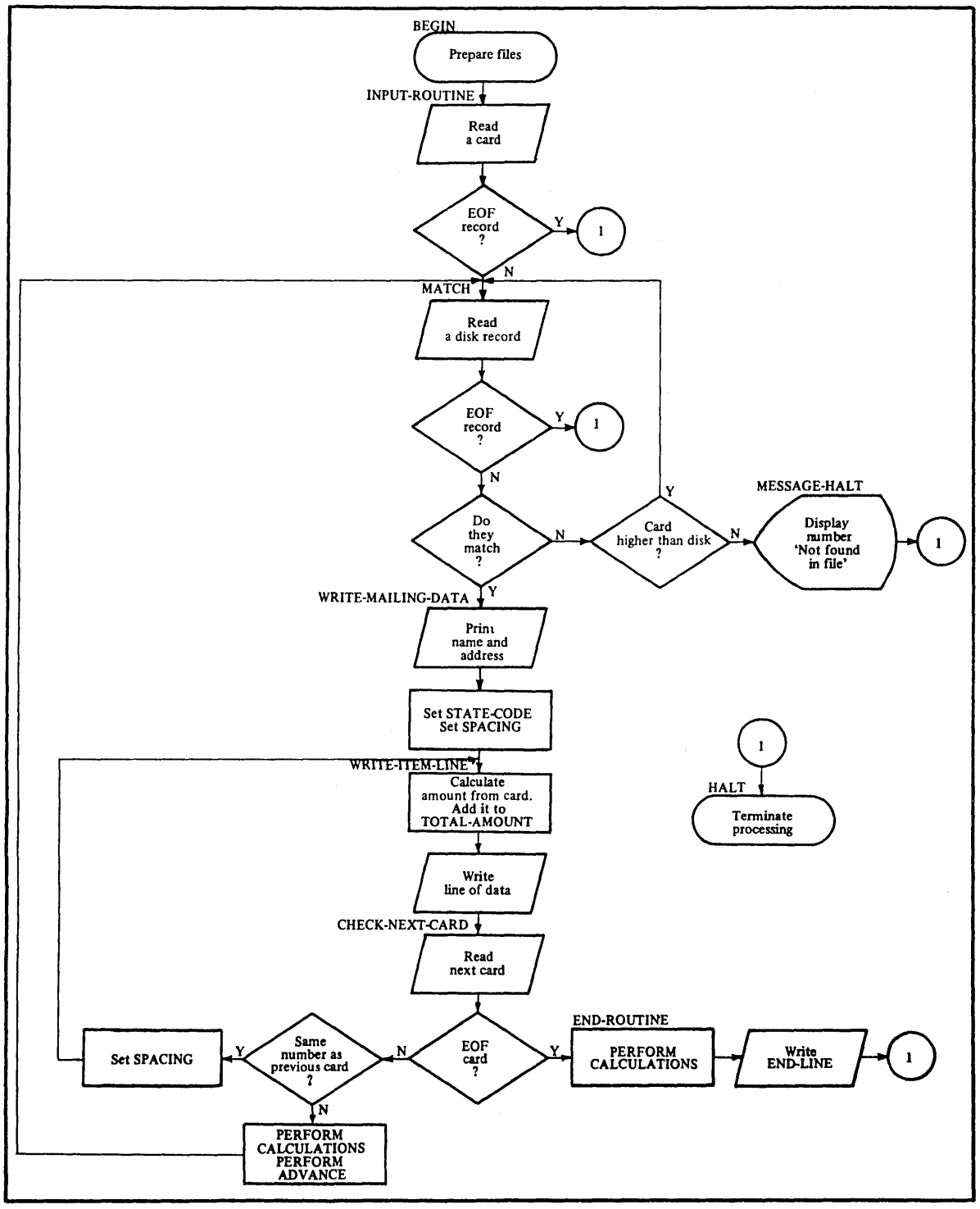

BEGIN
Prepare files
INPUT-ROUTINE
Read a card
EOF record ?
Y
1
N
MATCH
Read a disk record
EOF record ?
Y
1
N
Do they match ?
N
Card higher than disk ?
Y
N
MESSAGE-HALT
Display number 'Not found in file'
1
Y
WRITE-MAILING-DATA
Print name and address
Set STATE-CODE Set SPACING
WRITE-ITEM-LINE
Calculate amount from card. Add it to TOTAL-AMOUNT
Write line of data
CHECK-NEXT-CARD
Read next card
EOF card ?
N
Y
END-ROUTINE
PERFORM CALCULATIONS
Write END-LINE
1
Same number as previous card ?
Y
Set SPACING
N
PERFORM CALCULATIONS PERFORM ADVANCE
1
HALT
Terminate processing

Figure 83

10. Figure 82 shows a portion of a program used by a Michigan dealer to calculate and print monthly bills for customers. Figure 83 is a flow chart that is included to help you follow the program in Figure 82. The master disk contains the numbers and mailing information for all customers. The input cards each contain data for one purchase by one customer with the cards in ascending sequence by customer. More than one card may be present for each customer, or there may be no cards for some customers since not every customer makes a purchase each month.

    The PERFORM statement is used in paragraphs CHECK-NEXT-CARD and END-ROUTINE in this program to execute paragraph CALCULATIONS, which you will write in the next frame. A statement that could be used in paragraph CALCULATIONS is the:

    a. GO TO statement.

    b. PERFORM statement.

    c. IF statement.

* * *

b,c

---

11. Figure 84 shows a detailed flow chart for paragraph CALCULATIONS. Write paragraph CALCULATIONS, after checking the Data Division of program BILLING (Figure 78) for the variable names.

* * *

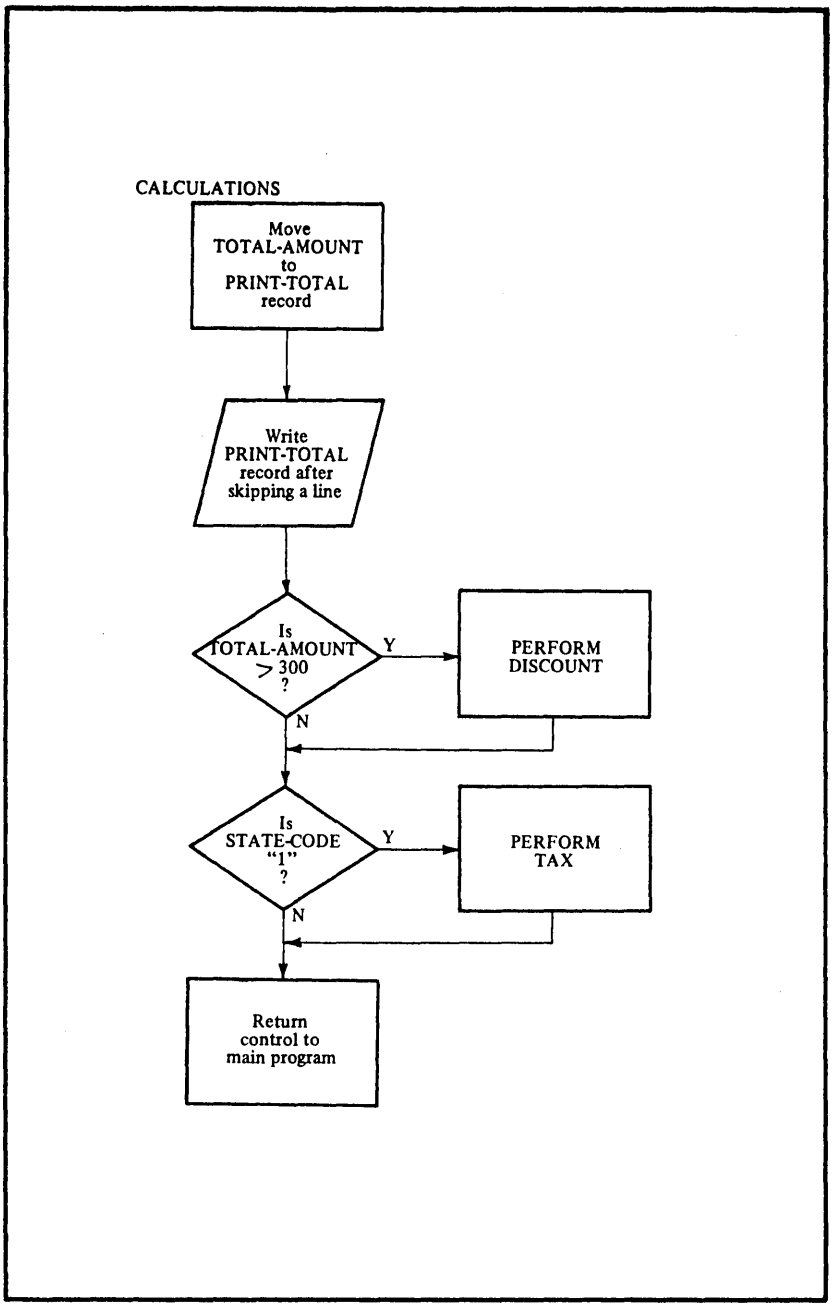


Figure 84

```
0    0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CALCULATIONS.
           MOVE TOTAL-AMOUNT TO TOTAL.
           WRITE BILL-PRINT
               FROM PRINT-TOTAL
               AFTER ADVANCING 2 LINES.
           IF TOTAL-AMOUNT IS GREATER THAN 300
               PERFORM DISCOUNT.
           IF STATE-CODE IS EQUAL TO 1
               PERFORM TAX.
```

---

12. Figure 85 gives flow charts for paragraphs TAX and DISCOUNT. First write paragraph TAX to calculate and print the tax.

* * *

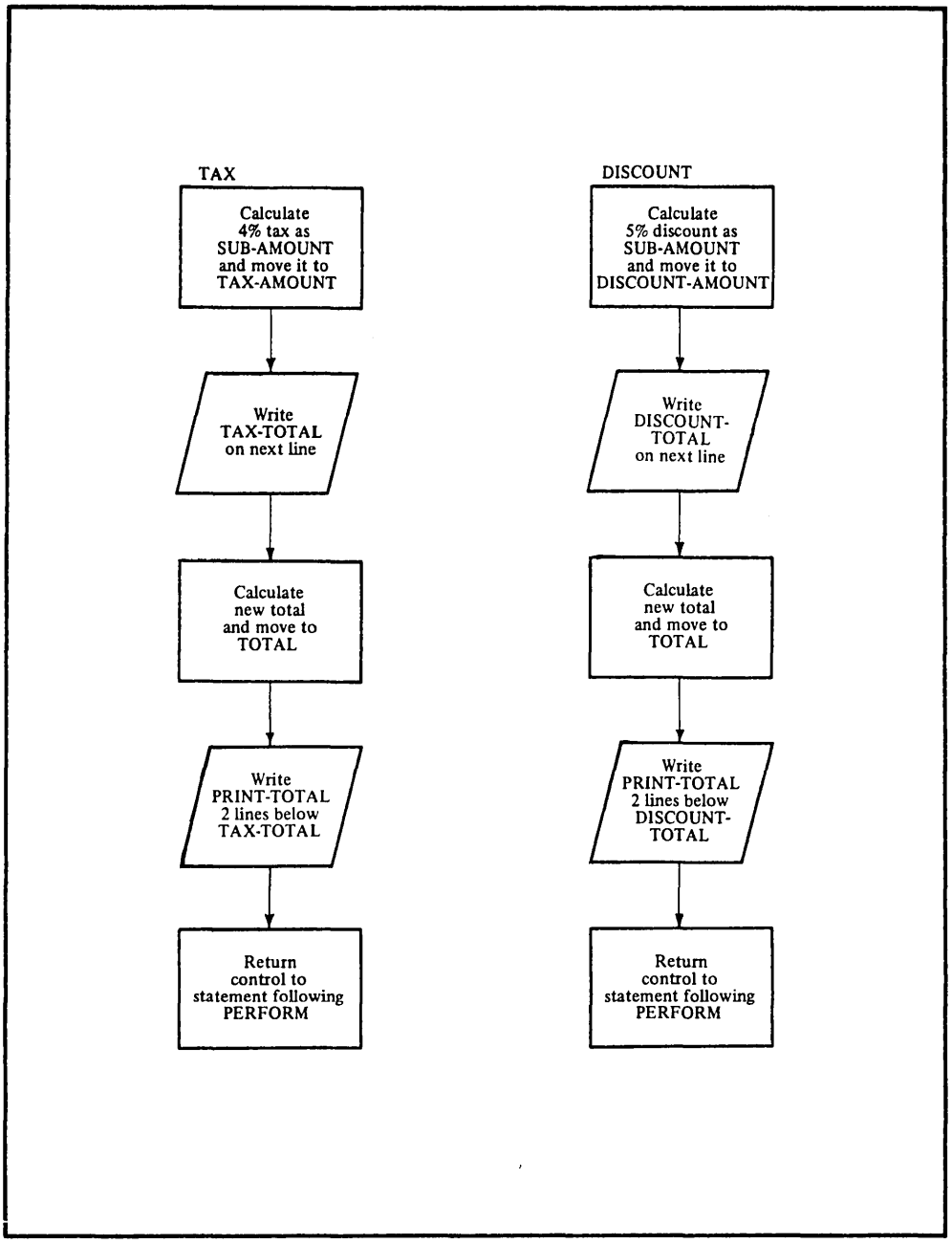


Figure 85

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
TAX.
    MULTIPLY .04 BY TOTAL-AMOUNT
        GIVING SUB-AMOUNT.
    MOVE SUB-AMOUNT TO TAX-AMOUNT.
    WRITE BILL-PRINT FROM TAX-TOTAL
        AFTER ADVANCING 1 LINE.
    ADD SUB-AMOUNT TO TOTAL-AMOUNT.
    MOVE TOTAL-AMOUNT TO TOTAL.
    WRITE BILL-PRINT FROM PRINT-TOTAL
        AFTER ADVANCING 2 LINES.
```

(In the MULTIPLY statement, you cannot specify "GIVING TAX-AMOUNT" because the value of SUB-AMOUNT is needed for use in the ADD statement. You cannot specify "GIVING TOTAL" in the ADD statement because the value of TOTAL-AMOUNT is used in the main program.)

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    SUBTRACT CHECK FROM ACCOUNT.
```

The flow chart for paragraph DISCOUNT is very similar to the one you used in writing TAX. One crucial difference, however, is that the amount of discount is to be subtracted from the total bill, while the amount of tax was to be added. The effect of execution of the statement above if the value of CHECK were 1000 and the value of ACCOUNT were 20700 would be that the value of:

a. CHECK changes to 19700

b. ACCOUNT changes to 19700

* * *

b

14.

```
          ⎡identifier-1⎤   ⎡identifier-2⎤
SUBTRACT  ⎢            ⎥   ⎢            ⎥  ...
          ⎣literal-1   ⎦   ⎣literal-2   ⎦

    FROM identifier-m
```

The format of the SUBTRACT statement is shown above. The rules for usage of identifiers and literals are the same as for the ADD statement. Which SUBTRACT statements are of the correct format?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SUBTRACT 10, CHECK FROM ACCOUNT.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SUBTRACT CHECK FROM 10, ACCOUNT.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SUBTRACT CHECK FROM 10.
```

* * *

a
(b specifies two items after FROM; c specifies a literal after FROM. The commas are optional.)

---

15. The options that you learned to use with other arithmetic statements can also be used in the SUBTRACT statement. Write a statement to subtract the value of PAYMENTS from the value of BALANCE. The result of the subtraction is to be stored in NEW-BALANCE.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SUBTRACT PAYMENTS FROM BALANCE
               GIVING NEW-BALANCE.
```

---

16. Now write paragraph DISCOUNT to complete program BILLING. Figure 85 gives the flow chart for this procedure.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DISCOUNT.
           MULTIPLY .05 BY TOTAL-AMOUNT
               GIVING SUB-AMOUNT.
           MOVE SUB-AMOUNT TO DISCOUNT-AMOUNT.
           WRITE BILL-PRINT
               FROM DISCOUNT-TOTAL
               AFTER ADVANCING 1 LINE.
           SUBTRACT SUB-AMOUNT
               FROM TOTAL-AMOUNT.
           MOVE TOTAL-AMOUNT TO TOTAL.
           WRITE BILL-PRINT FROM PRINT-TOTAL
               AFTER ADVANCING 2 LINES.
```

(In the MULTIPLY statement, you cannot specify "GIVING DISCOUNT-AMOUNT" because the value of SUB-AMOUNT is needed for use in the ADD statement. You cannot specify "GIVING TOTAL" in the SUBTRACT statement because the value of TOTAL-AMOUNT is used in the main program.)

---

17. As a programmer you will not be given a flow chart in most cases. You will normally develop your own flow charts for programming tasks. Your next task is to draw a flow chart for a program that involves adding records to a disk file. Later you will be asked to write a program, so be sure to include all of the steps in your flow chart.

    A community college that has many transfer students keeps a record of the residence and scholastic history of each student, along with the year he entered the college and his unique number, on a master disk. A card file contains data on the incoming transfer students. The data for each student is arranged on a single card. If an input card contains a number which is already in the file, a message containing the card number and the literal 'IS IN FILE.NOT ADDED.' is to be displayed, and a new card is to be read. The last record in each input file is a dummy record with the number 999999999. A new disk file is to be created containing all old disk records and all card records. When an end-of-file record is encountered in either file the program is to be terminated. Both input files are in ascending sequence, and the output file is to be produced in this sequence. Draw a flow chart for the problem.

* * *

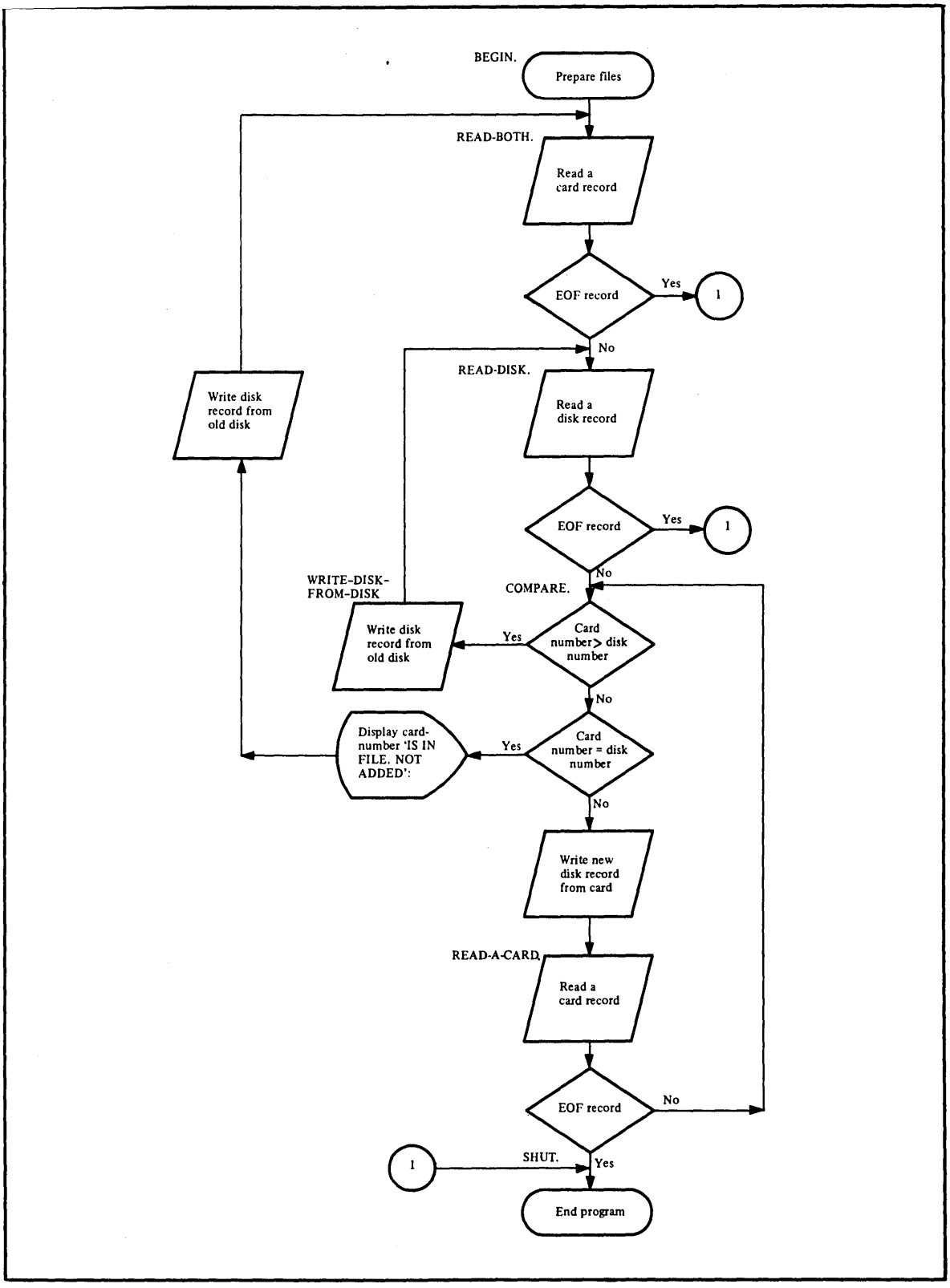

BEGIN.
Prepare files
READ-BOTH.
Read a card record
EOF record
Yes
1
No
READ-DISK.
Read a disk record
EOF record
Yes
1
No
COMPARE.
Card number > disk number
Yes
Write disk record from old disk
WRITE-DISK-FROM-DISK
No
Card number = disk number
Yes
Display card-number 'IS IN FILE, NOT ADDED':
No
Write new disk record from card
READ-A-CARD.
Read a card record
EOF record
No
1
SHUT.
Yes
End program
Write disk record from old disk

Figure 86

(If you are not familiar with flowcharting techniques, review the prerequisite course Fundamentals of Programming. The paragraph names are included only for your use in the next frame.)

The following problem incorporates the matching record techniques you have been studying in this lesson. Since you are not asked to be anything new in this problem, it is optional for you to code it. If you choose not to code the solution, be certain to read it carefully to understand how the matching records problem is solved.

---

18. Figure 87 shows the first three divisions for the program represented by the flow chart created in the preceding frame. Write a Procedure Division for the program using the flow chart that you drew or the solution given to the preceding problem.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. MATCH.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENT-MASTER
               ASSIGN TO DF-1-600-X.
           SELECT NEW-STUDENT-MASTER
               ASSIGN TO DF-2-800-X.
           SELECT UPDATE-DATA
               ASSIGN TO RD-1442.
       DATA DIVISION.
       FILE SECTION.
       FD  STUDENT-MASTER
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 5 RECORDS.
       01  STUDENT-DATA.
           02  PERSONAL.
               03  YEAR-IN PIC XX.
               03  STUDENT-NUM PIC X(9).
               03  FILLER PIC X(35).
           02  SCHOLASTIC PIC X(14).
       FD  NEW-STUDENT-MASTER
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 5 RECORDS.
       01  STUDENT-DATA-NEW.
           02  YEAR PIC XX.
           02  S-NUMBER PIC X(9).
           02  FILLER PIC X(74).
       FD  UPDATE-DATA
           LABEL RECORDS ARE OMITTED.
       01  TRANSFER.
           02  IN-OR-OUT PIC XX.
           02  CARD-NUMBER PIC X(9).
           02  FILLER PIC X(69).
```

Figure 87

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT-MASTER
                UPDATE-DATA
                OUTPUT NEW-STUDENT-MASTER.
       READ-BOTH.
           READ UPDATE-DATA
                AT END GO TO SHUT.
       READ-DISK
           READ STUDENT-MASTER
                AT END GO TO SHUT.
       COMPARE.
           IF CARD-NUMBER IS GREATER THAN
                STUDENT-NUM
                GO TO WRITE-DISK-FROM-DISK.
           IF CARD-NUMBER IS EQUAL TO
                STUDENT-NUM
                DISPLAY CARD-NUMBER
                     'IS IN FILE. NOT ADDED.'
                     UPON CONSOLE
                WRITE STUDENT-DATA-NEW
                     FROM STUDENT-DATA
                GO TO READ-BOTH
                 ELSE WRITE STUDENT-DATA-NEW
                     FROM TRANSFER.
       READ-A-CARD.
           READ UPDATE-DATA
                AT END GO TO SHUT.
           GO TO COMPARE.
       WRITE-DISK-FROM-DISK.
           WRITE STUDENT-DATA-NEW
                FROM STUDENT-DATA.
           GO TO READ-DISK.
       SHUT.
           CLOSE STUDENT-MASTER
                NEW-STUDENT-MASTER
                UPDATE-DATA.
           STOP RUN.
```

(There are many possible solutions to this problem as there are to most programming assignments. If your solution appears different and you are not sure of its correctness, check with your advisor.)

The following program incorporates the matching record techniques you have been studying in this lesson. Since you are not asked to do anything new in this problem, it is optional for you to code it. If you choose not to code the solution, be certain to read it carefully to understand how the matching records problem is solved.

---

19. As a programmer you may be required to update a disk by removing records from it. The three divisions in Figure 87 could also be used in a program to remove records. Assume now that each input card contains the number of a student who has transferred out of the community college; again, the last record in each input file is a dummy record with the number 999999999. Draw a flow chart and then write a Procedure Division to create a new disk containing only those records on the old disk for which there are no matching cards.

* * *

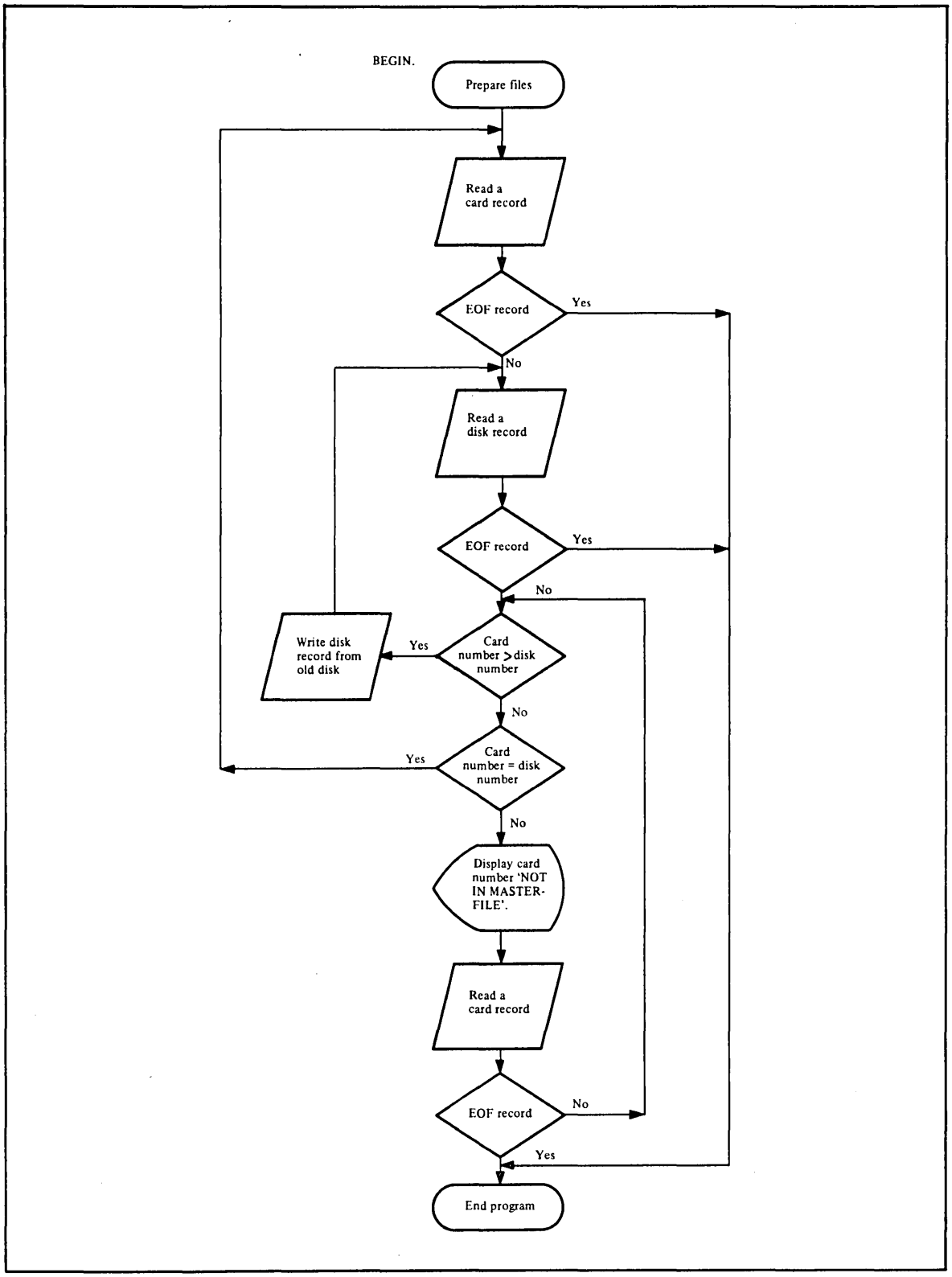

BEGIN.
Prepare files
Read a card record
EOF record
Yes
No
Read a disk record
EOF record
Yes
No
Write disk record from old disk
Yes
Card number > disk number
No
Yes
Card number = disk number
No
Display card number 'NOT IN MASTER-FILE'.
Read a card record
EOF record
No
Yes
End program

Figure 88

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENT-MASTER
                UPDATE-DATA
                OUTPUT NEW-STUDENT-MASTER.
       READ-CARD.
           READ UPDATE-DATA
                AT END GO TO SHUT.
       READ-DISK
           READ STUDENT-MASTER
                AT END GO TO SHUT.
       COMPARE.
           IF CARD-NUMBER IS EQUAL TO
                STUDENT-NUM
                GO TO READ-CARD.
           IF CARD-NUMBER IS GREATER THAN
                STUDENT-NUM
                WRITE STUDENT-DATA-NEW
                FROM STUDENT-DATA                    (4)
                GO TO READ-TAPE.
           DISPLAY CARD-NUMBER
                'NOT IN MASTER FILE.'
                UPON CONSOLE.
           READ UPDATE-DATA
                AT END GO TO SHUT.
           GO TO COMPARE.
       SHUT.
           CLOSE STUDENT-MASTER
                NEW-STUDENT-MASTER
                UPDATE-DATA.
           STOP RUN.
```

SUMMARY:

The two procedures that you have just written could easily have been combined into a single, more complex, program. It is frequently desirable to add some records to a disk file, delete other records, and update or change still other records. This can be done by using a field in the transaction file to hold a code such as 1 for add, 2 for delete, and 3 for change. A different procedure would then be followed for each code. Branching to different procedures would be accomplished after testing the code field each time a transaction card is read. The logic in this lesson was kept simple so that you could concentrate on one specific reason for matching records at a time.

In addition to matching records you have learned to use the FROM option of the WRITE statement as a substitute for writing a MOVE statement, the PERFORM statement to transfer control temporarily, and the SUBTRACT statement.

END OF LESSON 18

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 19

LESSON 19 - CONDITIONAL STATEMENTS (1)

INTRODUCTION

In preceding lessons you have learned to specify mutually exclusive paths of control based on such circumstances as the number in the record in the card file being greater than the number in the record in the disk file. This circumstance is represented by the following expression:

CARD-NUMBER GREATER THAN DISK-NUMBER

This expression is tested in an IF statement at some point in the program and the appropriate path of control will be selected depending on whether the expression is true or false. Expressions such as the one shown above are called test conditions because they are tested to determine whether they are true or false.

In this lesson you will learn to specify various kinds of test conditions. Specific COBOL language features that you will learn in this lesson are:

Relational operators
Relational conditions
Valid comparisons
Picture character S
NOT logical operator

This lesson will require approximately one half hour.

1.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF STUDENT-NUMBER EQUAL TO ID-NUMBER
    GO TO READ-CARD.
```

The COBOL statement above specifies a test condition. The test condition is:

a. STUDENT-NUMBER EQUAL TO ID-NUMBER

b. IF STUDENT-NUMBER EQUAL TO ID-NUMBER

c. GO TO READ-CARD

* * *

a
(b is the reserved word IF plus the condition.)

Relational and Logical Operators

Type of Operation	Operator (operation symbol)	Operation
Relational	IS GREATER THAN	Is greater than
	IS LESS THAN	Is less than
	IS EQUAL TO	Is equal to
Logical	OR	Logical inclusive OR (either or both are true)
	AND	Logical conjunction (both are true)
	NOT	Logical negation

Figure 89

2. The test condition STUDENT-NUMBER EQUAL TO ID-NUMBER contains a relational operator. Figure 89 shows that the relational operator in the test condition above is ..... .

* * *

EQUAL TO

3.

```
{identifier-1}                              {identifier-2}
{literal-1   } relational-operator          {literal-2   }
{arithmetic- }                              {arithmetic- }
{expression-1}                              {expression-2}
```

TOTAL GREATER THAN 1000

Figure 90

A test condition containing a single relational operator is called a relational condition. The form of a relational condition and an example of a test condition are shown above. The test condition is:

a. a relational condition.

b. used to compare the value of a variable with the value of a literal.

c. used to specify a relationship between two items.

* * *

All of these

---

4. TOTAL GREATER THAN 1000

The result of a relational condition is either true or false. If the value of TOTAL were 2000, the result of the relational condition above would be ..... .

* * *

true

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF COUNTER LESS THAN 1000
    GO TO READ1
    ELSE GO TO READ2.
```

The appearance of a test condition in an IF statement causes the test condition to be evaluated. When the IF statement above is executed:

a. control will transfer to READ1 if COUNTER is equal to 2000.

b. the specified relational condition will be evaluated.

c. if COUNTER is equal to 500, the result of the relational condition will be false.

* * *

b

---

6. AMOUNT GREATER THAN
MAXIMUM-BALANCE

Write a statement to cause evaluation of the test condition above. Specify that, if the result is true, the paragraph with the name REJECT is to be performed and that, if the result is false, the next sentence is to be executed. The PERFORM statement is used to depart from the normal sequence of procedures to execute a statement(s) a specified number of times until a predetermined condition is satisfied.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF AMOUNT GREATER THAN MAXIMUM-BALANCE
              PERFORM REJECT.
```

7.

first-operand relational-operator second-operand

relational-condition

Figure 91

According to the diagram above, in the relational condition

SALARY-CODE EQUAL TO 0

the first operand is ..... and the second operand is .....
.

* * *

```
SALARY-CODE
0
```

---

8. Figure 92 shows the operands that can be used in valid relational conditions. Operands in a relational condition can be:

Types of Valid Comparisons

First Operand \ Second Operand		Group	Elementary			
			Alphanumeric	Alphabetic	Numeric	Literal
Group		C	C	C	C	C
Elementary	Alphanumeric	C	C	C	C	C
	Alphabetic	C	C	C	I	C
	Numeric	C	C	I	N	C
	Literal	C	C	C	C	I

C-Compared logically (one character at a time, according to collating sequence shown earlier)
N-Compared algebraically (numeric values are compared)
I-Invalid comparison

Example:

IF TOTAL GREATER THAN MAXIMUM GO TO MESSAGE

first operand, operator second, operand

condition

Explanation:

To use this chart find the data type (determined by the picture) of the first operand in the column headed First Operand. Then find the data type of the second operand across the top of the figure opposite Second Operand. Extend imaginary lines into the figure from the data types of the first and second operands. In the block where these two lines intersect is a letter that tells you how the values are compared.

Figure 92

a. group variables.

b. literals.

* * *

Either
(A more comprehensive table of permissible comparisons can be found in the Language Specifications Manual.

---

9. Figure 92 indicates that a valid relational condition can contain:

   a. two literals as operands.

   b. an alphanumeric operand and an alphabetic operand.

                    *   *   *

b

10.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WORKING-STORAGE SECTION.
01  WORK-RECORD.
    02  SHOP-NAME PIC A(10).
    02  WORK-NUMBER PIC XXXXX.
    02  EMPLOYEE-TIME PIC 999V99.
    02  WORK-RATE PIC 999V99.
01  SECOND-RECORD.
    02  NAME PIC A(10).
    02  WORKNUMBER PIC XXXXX.
    02  TIME-EMPLOYEE PIC 999V99.
    02  RATE-WORK PIC 999V99.
```

According to the Data Division entries above and Figure 92, which of the following is a valid comparison?

   a. WORK-NUMBER GREATER THAN
      WORKNUMBER

   b. NAME LESS THAN WORK-RATE

   c. WORK-NUMBER EQUAL TO NAME

                    *   *   *

a, c

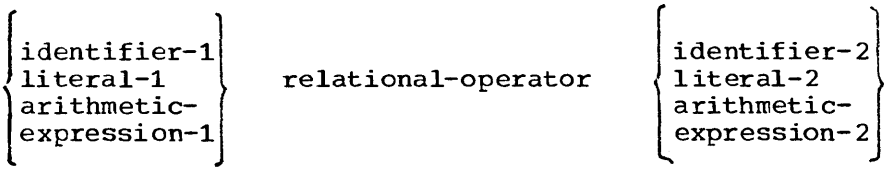


Figure 93

The form of the relational condition above indicates that either operand can be an arithmetic expression. In this context, arithmetic expression refers to a combination of numeric variables, numeric literals, and arithmetic operators. This type of relational condition will not be covered here because it is generally best to have arithmetic operations performed before the relational condition is to be evaluated.

11.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       WORKING-STORAGE SECTION.
       01  WORK-RECORD.
           02  SHOP-NAME PIC A(10).
           02  WORK-NUMBER PIC XXXXX.
           02  EMPLOYEE-TIME PIC 999V99.
           02  WORK-RATE PIC 999V99.
       01  SECOND-RECORD.
           02  NAME PIC A(10).
           02  WORKNUMBER PIC XXXXX.
           02  TIME-EMPLOYEE PIC 999V99.
           02  RATE-WORK PIC 999V99.
```

There are two types of relational conditions: comparison of numeric operands and comparison of nonnumeric operands. Numeric operands are compared algebraically. For example, the value 02 would be greater than -10, and -10 would be greater than -11. According to the Data Division entries above, which of the following will be compared algebraically?

a. EMPLOYEE-TIME GREATER THAN
   WORK-RATE

b. WORK-NUMBER EQUAL TO
   WORKNUMBER

c. RATE-WORK LESS THAN NAME

* * *

a
(If a relational condition contains a numeric operand and a nonnumeric operand, it is treated as a nonnumeric comparison.)

---

12. In order for the value of a numeric variable to include a sign, the letter S must be specified preceding the first 9 in the picture for that variable. The S, like the V, does not represent a character position. Write a PICTURE clause that would permit a variable to have integer values from -12 to +12.

* * *

PIC S99.
(When a relational condition is evaluated, any unsigned value other than zero is considered positive. Zero is a unique value, and any preceding sign is ignored.)

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WORKING-STORAGE SECTION.
       77  COMP-NUMBER PIC S999V99.
       77  REGIONAL-TOTAL PIC S999V99.
       01  RECORD-1.
           02  RECORD-NUMBER PIC XXXXX.
           02  RECORD-TOTAL PIC S999V99.
           02  RECORD-BALANCE PIC S999V99.
           02  BALANCE-LIMIT PIC S999V99.
```

Variables	Values
COMP-NUMBER	-997/01
REGIONAL-TOTAL	+903/65
RECORD-NUMBER	77321
RECORD-TOTAL	+336/70
RECORD-BALANCE	+000/00
BALANCE-LIMIT	-000/00

Figure 94

Refer to the pictures and values above and determine whether the result of each relational condition below is true or false.

1) RECORD-TOTAL LESS THAN
   COMP-NUMBER

2) REGIONAL-TOTAL GREATER THAN
   COMP-NUMBER

3) RECORD-BALANCE GREATER THAN
   BALANCE-LIMIT

* * *

1) false
2) true
3) false

14.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
WORKING-STORAGE SECTION.
77  COMP-NUMBER PIC S999V99.
77  REGIONAL-TOTAL PIC S999V99.
01  RECORD-1.
    02  RECORD-NUMBER PIC XXXXX.
    02  RECORD-TOTAL PIC 999.
    02  RECORD-IDENT PIC A(4).
```

The second type of relational condition is the comparison of nonnumeric operands. Any comparison which includes a nonnumeric operand is a nonnumeric comparison. Nonnumeric comparisons are made with respect to the EBCDIC collating sequence. According to the data description entries above, which of the following comparisons will be made with respect to the collating sequence?

a. REGIONAL-TOTAL EQUAL TO
   COMP-NUMBER

b. RECORD-TOTAL GREATER THAN
   RECORD-NUMBER

c. RECORD-IDENT LESS THAN
   RECORD-NUMBER

* * *

b, c
(Any character in the picture of a variable other than a 9, a V, or an S indicates that the variable is nonnumeric. Any edited variable with a $ or . in its picture is nonnumeric, as are variables with X's or A's.)

---

15. Figure 7 shows the collating sequence of the COBOL character set in descending order. The first character in the list is considered to have the highest value; the last character is considered to have the lowest value. If the value of MARK were 5, the result of the relational condition

    MARK LESS THAN 'A'

    would be ..... .

* * *

false

---

16. If the nonnumeric operands of a relational condition are the same length, the characters in corresponding positions are compared from left to right. If an unequal pair of characters is encountered, the operand containing the character higher in the collating sequence is considered to be the greater operand.

    NAME1 EQUAL TO NAME2

    When the relational condition above is evaluated, NAME1 has the value BACKER and NAME2 has the value BARKER. According to Figure 1, the result will be:

    a. true

    b. false because the value of NAME1 is greater than the value of NAME2.

    c. false because the value of NAME2 is greater than the value of NAME1.

                    *   *   *

c (Figure 7 shows that the letter Z has the highest value of the letters of the alphabet and that A has the lowest.)

---

17. NAME1 GREATER THAN NAME2

    When a relational condition with nonnumeric operands of unequal size is evaluated, the shorter operand is padded with blanks on the right to make it equal in length to the longer operand. Then the operands are compared in the normal way beginning with the leftmost characters. When the relational condition above is evaluated, the value of NAME1 is CARR and the value of NAME2 is CARVER. Figure 7 indicates that the result of this relational condition will be ..... .

                    *   *   *

false

18.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       01  TRANSACT-RECORD.
           02  CLIENT-NUMBER PIC X(5).
           02  ITEM-ID PIC A(5).
           02  UNIT-PRICE PIC 999V99.
           02  QUANTITY PIC 999.
           02  ITEM-DESCRIPTION PIC X(62).
       01  TRANSACTIONS.
           02  CLIENT-ID PIC X(5).
           02  PART-ID PIC A(5).
           02  AMOUNT-EACH PIC 999V99.
           02  NUMBER-ORDERED PIC 999.
           02  PART-DESCRIPTION PIC X(62).
```

Variables	Values	Variables	Values
CLIENT-NUMBER	55471	CLIENT-ID	47762
ITEM-ID	AACFG	PART-ID	XAACN
UNIT-PRICE	077/25	AMOUNT-EACH	172/50
QUANTITY	500	NUMBER-ORDERED	025

Figure 95

Refer to the pictures and values above and match the word or words below with the correct relational condition. (You may also refer to Figures 7 and 92 if necessary.)

1) CLIENT-NUMBER GREATER THAN
   CLIENT-ID

2) PART-ID LESS THAN QUANTITY

3) UNIT-PRICE GREATER THAN
   AMOUNT-EACH

4) NUMBER-ORDERED LESS THAN
   QUANTITY

a. Invalid

b. Compared algebraically

c. Compared with respect to the collating sequence

d. True

e. False

* * *

1) c, d
2) a (An alphabetic item cannot be compared with a numeric item.)
3) b, e
4) b, d

---

19. Write an IF statement to specify that if ACCOUNT is greater than BASE, control is to be transferred to a routine called CALCULATE. If ACCOUNT is equal to or less than BASE, another record is to be read from INFILE. When the last card has been read from INFILE, control is to transfer to FINISH.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF ACCOUNT GREATER THAN BASE
              GO TO CALCULATE
              ELSE READ INFILE
                  AT END GO TO FINISH.
```

20.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF ACCOUNT NOT GREATER THAN BASE
    READ INFILE AT END GO TO FINISH
    ELSE GO TO CALCULATE.
```

The IF statement you wrote in the previous frame could also be written as shown above. NOT is a logical operator that causes logical negation. In the IF statement above, the relational condition will be true if ACCOUNT is not greater than BASE.

PRICE NOT EQUAL TO LIST-PRICE

The relational condition above will be true if PRICE is:

a. equal to LIST-PRICE.

b. greater than LIST-PRICE.

c. less than LIST-PRICE.

* * *

b, c

---

21. The logical operator NOT can be used with any relational operator. It must always be preceded by and followed by a space. Write a statement using the NOT logical operator to specify that if PART-NUMBER is equal to or greater than ORDER-NUMBER, the record STATEMENT is to be written.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
IF PART-NUMBER NOT LESS THAN
    ORDER-NUMBER WRITE STATEMENT.
```

You know that an IF statement causes a relational condition to be evaluated and the appropriate path of control to be selected based on whether the relational condition is true or false. As a programmer you might be required to write a program in which the basis for the selection of the appropriate path of control will depend on a set of circumstances instead of a single relational condition. A possible solution is to use a compound condition in a single IF statement.

SUMMARY:

You have now completed Lesson 19 in which you have learned to specify various types of conditions in IF statements. You have learned to use a relational condition to specify a relationship between two variables or a variable and a literal. You have learned to use the logical operator NOT to specify negation. The class condition, which tests whether a variable is alphabetic or numeric, and the sign condition, which tests whether a numeric variable is positive, negative, or zero, will not be taught in this course. For information on the uses of these conditions, refer to the Language Specifications Manual after completing this course.

END OF LESSON 19

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 20

LESSON 20 - CONDITIONAL STATEMENTS (2)

INTRODUCTION

In this lesson you will learn to use the logical operators AND and OR to specify compound conditions. You will also learn to use condition-name conditions to test a variable that has one of a specific set of variables.

Specific COBOL language features you will learn in this lesson are:

AND and OR logical operators
Compound conditions
Condition-name conditions
Level 88

This lesson will require approximately one half hour.

1. A compound condition is formed by using the logical operators other than NOT. Figure 89 includes a list of the logical operators. The logical operators that can be used to form a compound condition are ..... .

* * *

OR and AND

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF COST GREATER THAN 100 AND
               SYMBOL EQUAL TO IDENTIFICATION
               GO TO FINISH.
```

A compound condition can be two relational conditions connected by OR or AND. The IF statement above contains the compound condition

```
COST GREATER THAN 100 AND SYMBOL
    EQUAL TO IDENTIFICATION
```

which, when evaluated, will give one result.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF NAME1 EQUAL TO NAME2 AND
               ITEM EQUAL TO 10 PERFORM READ1.
```

The compound condition in the IF statement above:

a. is NAME1 EQUAL TO NAME2.

b. will give two results when evaluated.

* * *

Neither (The compound condition is NAME1 EQUAL TO NAME2 AND ITEM EQUAL TO 10, which will give one result when evaluated.)

3.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF NAME1 EQUAL TO NAME2 AND
             ITEM EQUAL TO 10 PERFORM READ1.
```

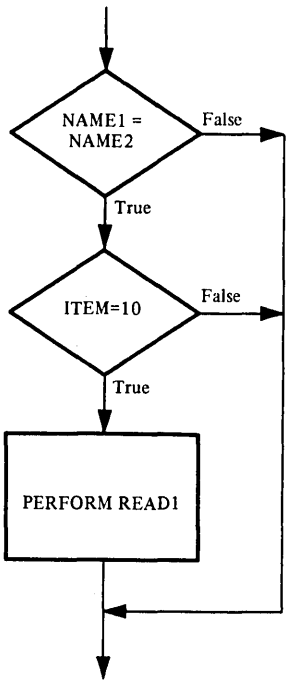


Figure 96

The logical operator AND can connect two relational conditions to form one compound condition that will give one result when evaluated. The flow chart above diagrams the action specified by the IF statement. As the word AND implies, the flow chart shows that in order for a compound condition specifying AND to be true:

a. both relational conditions must be true.

b. either the first or the second relational condition must be true.

* * *

a

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF NAME1 EQUAL TO NAME2 AND
               ITEM EQUAL TO 10 PERFORM READ1.
```

When a compound condition consists of two relational conditions connected by the logical operator AND, the relational conditions are evaluated first. If both relational conditions are true, the compound condition is true; otherwise it is false. When the statement above is executed NAME1 is equal to NAME2 and ITEM is equal to 5. The next statement executed will be:

a. the first statement in paragraph READ1.

b. the first statement in the sentence sequentially following the IF statement.

* * *

b

---

5. Write a single IF statement to specify that if GROSS is greater than 200 and if DEPENDENTS is not equal to zero, the value 1 is to be moved to INDICATOR, otherwise the value 0 is to be moved to INDICATOR.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF GROSS GREATER THAN 200 AND
               DEPENDENTS NOT EQUAL TO 0
               MOVE 1 TO INDICATOR
               ELSE MOVE 0 TO INDICATOR.
```

(The figurative constant ZERO could be used instead of the symbol 0.)

---

6. The logical operator OR can be used in the same way as AND to form a compound condition. An example of a compound condition using OR is:

a. NUMBER-1 OR GREATER THAN
NUMBER-2

b. OR NUMBER-1 GREATER THAN
NUMBER-2

c. NUMBER-1 GREATER THAN
NUMBER-2 OR SWITCH EQUAL TO 1

* * *

c

7.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF NUMBER-1 GREATER THAN NUMBER-2
              OR SWITCH EQUAL TO 1
              ADD 1 TO NEW-COUNT.
```

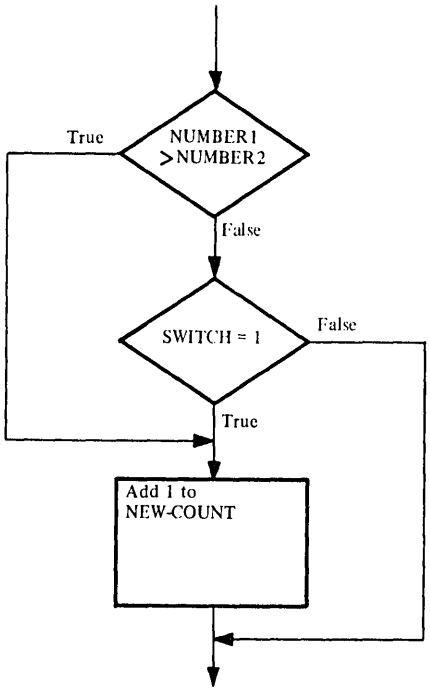


Figure 97

The flow chart above diagrams the action specified by the statement. As the word OR implies, a compound condition specifying OR will be true:

a. if either the first relational condition or the second relational condition is true.

b. only if both relational conditions are true.

* * *

a

---

8. A compound condition containing the logical operator OR will be true if either the first or second or both relational conditions are true. Which of the following statements specifies that the value 1 will be moved to SWITCH-A if A-NUMBER or B-NUMBER or both equal 2?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF A-NUMBER EQUAL TO 2 AND
               B-NUMBER EQUAL TO 2
               MOVE 1 TO SWITCH-A.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF A-NUMBER EQUAL TO 2 OR
               B-NUMBER EQUAL TO 2
               MOVE 1 TO SWITCH-A.
```

* * *

b

9.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF GROSS LESS THAN 500 OR
               PAYCODE EQUAL TO 1 PERFORM PAY-2
               ELSE PERFORM PAY-1.
```

When the statement above is executed and GROSS has the value:

1) 250 and PAYCODE has the value 1, the routine called ..... will be performed.

2) 250 and PAYCODE has the value 0, the routine called ..... will be performed.

3) 600 and PAYCODE has the value 1, the routine called ..... will be performed.

4) 600 and PAYCODE has the value 2, the routine called ..... will be performed.

* * *

1) PAY-2
2) PAY-2
3) PAY-2
4) PAY-1

---

10. Write a statement to specify that if TOTAL-SALES exceeds 5000 or CUSTOMER-CODE is equal to 1, or both, control will transfer to PREFERRED. Otherwise, 1 is to be added to REJECTS.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF TOTAL-SALES GREATER THAN 5000 OR
              CUSTOMER-CODE EQUAL TO 1
              GO TO PREFERRED
              ELSE ADD 1 TO REJECTS.
```

11.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          IF (A EQUAL TO B OR C EQUAL TO D)
              AND (W EQUAL TO X OR Y EQUAL TO Z)
              GO TO EQUATE
              ELSE GO TO BEGIN.
```

A compound condition can be two or more compound conditions connected by AND or OR. An example of an IF statement containing this type of compound condition is shown above. When the statement above is executed the simplest compound conditions will be evaluated first, then the results will be used to arrive at the final result based on whether AND or OR is used. When the IF statement above is executed, A is equal to B and C is equal to D. W is not equal to X and Y is not equal to Z. After the IF statement is executed, control will be transferred to a routine called ..... .

* * *

BEGIN
(An important consideration in writing complex test conditions is the order in which the conditions involved are evaluated. Information on the order of evaluation is given in the Language Specifications Manual under Compound Conditions.)

The relational condition is a type of test condition. Another type of test condition that you will find useful in programming is the condition-name condition. In general, the condition-name condition is used in an IF statement that tests whether a variable has a specific value or one of a specific set of values.

12.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       01  EMPLOYEE-RECORD.
           02  EMPLOYEE-NUMBER
           02  EMPLOYEE-NAME
           02  MARITAL-STATUS
           02  DEPENDENTS
           02  PAY-RATE
           02  PAY-CODE
```

In the partial record description above, PAY-CODE will always contain one of the values 1, 2, or 3. These values indicate whether the employee is paid by the hour, week, or month, respectively. The relational condition

PAY-CODE EQUAL TO 1

could be used in an IF statement to determine whether an employee is paid by the hour. A variation of the IF statement that might also be used for this purpose is shown below.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF HOURLY GO TO HOUR-RATE.
```

In this example HOURLY is:

a. equivalent to PAY-CODE EQUAL TO 1.

b. another way of expressing a relational condition.

* * *

Both

13.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF HOURLY GO TO HOUR-RATE.
```

As used in the statement above, HOURLY is the name of a condition that expresses the relational condition PAY-CODE EQUAL TO 1. If PAY-CODE contains the value 2, the employee is paid by the week. The condition in either of the following statements could be used to determine whether the employee is paid by the week.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF PAY-CODE EQUAL TO 2
               GO TO WEEK-RATE.

           IF WEEKLY GO TO WEEK-RATE.
```

In the statements above:

a. WEEKLY is a relational condition.

b. PAY-CODE is a condition name.

c. PAY-CODE is equivalent to WEEKLY.

* * *

None of these (WEEKLY is a condition-name condition that is equivalent to the relational condition PAY-CODE EQUAL TO 2.)

14.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  PAY-CODE PIC 9.
               88  HOURLY VALUE IS 1.
               88  WEEKLY VALUE IS 2.
               88  MONTHLY VALUE IS 3.
```

When a condition-name condition is to be used in place of a relational condition, the condition names must be described in the Data Division, assigned specific values, and associated with their related variable. The data description entries above show how PAY-CODE is associated with the condition names to be used to represent its values. This example shows that:

a. condition names have the level number 88.

b. the VALUE clause is used to assign to the condition name the value the name is to represent.

* * *

Both
(Each level 88 entry must contain a VALUE clause, even if it is written in the File Section. The conditional variable is an elementary item, and as such must be defined with a PICTURE clause.)

You have now learned all of the level numbers that are included in this course. Figure 98 in your Programmed Instruction - Illustrations is a summary of level numbers for your use at any time.

Level Numbers

Number	In Area	Purpose
01	A	Record description entries.
02 thru 49	A or B	Subdivision of level 01 entries. May be group or elementary items.
77	A	Independent data item (Not a subdivision; not subdivided)
88	A or B	Condition name (must use the VALUE clause)

For additional information on level numbers and their uses, consult the Language Specifications Manual.

Figure 98

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          02  PAY-CODE PIC 9.
              88  HOURLY VALUE IS 1.
              88  WEEKLY VALUE IS 2.
              88  MONTHLY VALUE IS 3.
```

The entry above defines PAY-CODE as a conditional variable that will contain the values represented by the associated condition names. This example shows that:

a. the conditional variable is defined with a PICTURE clause.

b. the conditional variable will contain one of the values assigned to its associated condition names by the VALUE clause.

c. PAY-CODE is a condition name.

* * *

a, b

---

16. The input area in a program contains the conditional variable MARITAL-STATUS. The column in a card that corresponds to this variable will contain M, S, or D indicating that the employee is married, single, or divorced, respectively. As is the case wherever non-numeric constants are used, such constants should be enclosed in quotation marks when used in VALUE clauses. Example: 'M'. Which of the following correctly shows how MARITAL-STATUS could be associated with appropriate condition names?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  MARITAL-STATUS PIC A.
               88  'M' VALUE IS MARRIED.
               88  'S' VALUE IS SINGLE.
               88  'D' VALUE IS DIVORCED.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  MARITAL-STATUS PIC A.
               03  MARRIED VALUE IS 'M'.
               03  SINGLE VALUE IS 'S'.
               03  DIVORCED VALUE IS 'D'.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  MARITAL-STATUS PIC A.
               88  MARRIED VALUE IS 'M'.
               88  SINGLE VALUE IS 'S'.
               88  DIVORCED VALUE IS 'D'.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  MARITAL-STATUS.
               88  MARRIED PIC A.
               88  SINGLE PIC A.
               88  DIVORCED PIC A.
```

* * *

c

---

17. JOB-TYPE is a level 02 elementary variable that will contain one of the values 1, 2, or 3, specifying a workshop worker, an office worker, or a factory worker, respectively. Write the data description entry for JOB-TYPE and associate with it appropriate condition names that can be used to represent its values.

```
                         *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  JOB-TYPE PIC 9.
               88  WORKSHOP-WORKER VALUE IS 1.
               88  OFFICE-WORKER VALUE IS 2.
               88  FACTORY-WORKER VALUE IS 3.
```

---

18. Using the data description entries you wrote in the preceding frame, write the statement(s) necessary to specify that if the employee is a workshop worker, control will transfer to a routine called WORKSHOP; if he is an office worker, control will transfer to a routine called OFFICE; if he is a factory worker, control is to go to the next statement in sequence.

```
                         *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF WORKSHOP-WORKER GO TO WORKSHOP.
           IF OFFICE-WORKER GO TO OFFICE.
```

19.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           02  PART-NUMBER PIC 9999.
               88  SMALL VALUES ARE 0000 THRU 0299.
               88  MED VALUES ARE 0300 THRU 0699.
               88  LARGE VALUES ARE 0700 THRU 1000.

           IF SMALL GO TO SMALL-SCALE.
```

In a preceding frame you wrote data description entries to cause single values to be represented by condition names. It is also possible to represent a range of values with a condition name by using the THRU option of the VALUE clause. The data description entries above show that the conditional variable PART-NUMBER can contain any value from 0000 to 1000. Within this range of values, three ranges are defined and represented by condition names. According to the data description entries for PART-NUMBER, when the IF statement above is executed, control will transfer to a routine called SMALL-SCALE if PART-NUMBER contains any of the values 0000 through 0299.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           IF LARGE ADD 2 TO FACTOR.
```

According to the data description entries for PART-NUMBER, when the statement above is executed the value of FACTOR will be increased by 2 if PART-NUMBER contains the value:

a. 0500.

b. 0700.

c. 0800.

d. 1000.

* * *

b, c, d (VALUES ARE is equivalent to VALUE IS; the phrases are interchangeable when they are used with the THRU option. The words IS and ARE are optional and may be omitted from any VALUE clause.)

---

20. GRADE is a level 02 variable that will contain values from 000 to 100. The values from 000 to 059 are failing grades, the values 060 through 069 are poor, the values 070 through 079 are passing, the values 080 through 089 are good, and the values 090 through 100 are excellent. Write the data description entry for GRADE and associate with it appropriate condition names that can be used to represent its values.

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           02  GRADE PIC 999.
               88  FAILING VALUES ARE 000 THRU 059.
               88  POOR VALUES ARE 060 THRU 069.
               88  PASSING VALUES ARE 070 THRU 079.
               88  GOOD VALUES ARE 080 THRU 089.
               88  EXCELLENT VALUES ARE 090 THRU 100.
```

---

21. Refer to the data description entries you wrote for the last frame and write a statement to specify that if the value of GRADE is less than 060, a routine called ACTION is to be performed.

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF FAILING PERFORM ACTION.

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           (IF GRADE LESS THAN 60 PERFORM ACTION
```

would also be correct.)

SUMMARY:

You have expanded your flexibility in testing capabilities by learning how to specify compound conditions. You have also learned how to test a variable that has one of a specific set of values.

END OF LESSON 20

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 21

LESSON 21 - CHANNEL SKIPPING AND ARITHMETIC

INTRODUCTION

As a COBOL programmer you may write programs for business data-processing problems that specify precise arrangement of output data. In previous lessons you learned to specify vertical and horizontal spacing to meet output requirements. When output data is to be recorded on preprinted forms, such as checks or invoices, it is often necessary to use the technique called channel skipping. In this lesson you will learn to specify channel skipping in the AFTER ADVANCING option of the WRITE statement.

Business data-processing problems usually require arithmetic calculations. You have already learned to specify addition, multiplication, and subtraction. In this lesson you will learn to specify division. You will also learn to specify combinations of the four basic operations in a single statement.

Specific COBOL language features that you will learn to use in this lesson are:

DIVIDE statement
COMPUTE statement
Channel skipping

This lesson will require approximately three quarters of an hour.

1. The COBOL statement used to specify division is the DIVIDE statement. The simplest form of the DIVIDE statement is:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{numeric-literal-1} \end{matrix} \right\} \underline{\text{INTO}} \text{ identifier-2}$$

The result of a DIVIDE statement, the quotient, is stored according to the rules for a MULTIPLY, SUBTRACT, or ADD statement. The quotient is stored in:

a. identifier-1

b. numeric-literal-1

c. identifier-2

* * *

c

2.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DIVIDE 5 INTO TOTAL.
```

The DIVIDE statement above specifies the same operation as TOTAL/5. The COBOL statement used to express the operation X/Y is:

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DIVIDE X INTO Y.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DIVIDE X BY Y.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DIVIDE Y INTO X.
```

* * *

c

---

3. Write a DIVIDE statement to specify that the value of RESULT is to be divided by the value of FACTOR.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           DIVIDE FACTOR INTO RESULT.
```

4.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       77  FACTOR PIC 99.
       77  RESULT PIC 999V99.

           DIVIDE FACTOR INTO RESULT.
```

The result of a DIVIDE statement is padded or truncated as necessary when it is stored in the appropriate variable. If the values of FACTOR and RESULT are 02 and 12500, respectively, after execution of the DIVIDE statement above, the value stored in:

a. FACTOR will be 06250.

b. RESULT will be 62500.

c. FACTOR will be 62500.

d. RESULT will be 06250.

* * *

d

5.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       77  DIVISOR PIC 99.
       77  DIVIDEND PIC 99V99.

           DIVIDE DIVISOR INTO DIVIDEND.
```

The values of DIVISOR and DIVIDEND are 03 and 0200, respectively. After execution of the DIVIDE statement above, the value stored in DIVIDEND will be ..... .

* * *

0066

6.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       77  BALANCE PIC 9999V99.
       77  MONTHS PIC 9V9.
```

The ROUNDED option is used in a DIVIDE statement to specify that the quotient is to be rounded. Write a statement to divide BALANCE by MONTHS and round the quotient to the nearest cent. (The ROUNDED option follows the name of the variable in which the result is to be stored.)

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DIVIDE MONTHS INTO BALANCE ROUNDED.
```

7.

DIVIDE { identifier-1 / numeric-literal-1 } INTO { identifier-2 / numeric-literal-2 }

GIVING identifier-3

The GIVING option can also be used in a DIVIDE statement as shown above. You would expect that when the form of the DIVIDE statement shown above is used, the value of:

a. identifier-2 is unchanged upon exection of the statement.

b. the quotient is stored in the variable indicated by identifier-3.

* * *

Both

---

8. Write a statement to divide BALANCE by MONTHS and store the rounded result in PAYMENTS (the ROUNDED option always follows the name of the variable in which the result is to be stored).

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           DIVIDE MONTHS INTO BALANCE
               GIVING PAYMENTS ROUNDED.
```

---

9. Write a statement to divide PROFIT by 500. The rounded quotient is to be stored in PORTION.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          DIVIDE 500 INTO PROFIT GIVING
              PORTION ROUNDED.
```

---

10. Any time an attempt is made to divide by zero, the size error condition will occur. In order to avoid invalid data resulting from this condition:

    a. appropriate instructions should be included in the SIZE ERROR option in the DIVIDE statement.

    b. the ROUNDED option should be specified.

* * *

a

It is often necessary to perform several operations in order to obtain a single result. For example, the area of a circle is r^2. Calculating the area would require two MULTIPLY statements. But the same result can be obtained by a single COMPUTE statement.

---

11. Several arithmetic operations can be combined in one COMPUTE statement. A COMPUTE statement can:

    a. reduce the number of statements in the source program.

    b. combine two or more arithmetic operations into a single statement.

* * *

Both
(The COMPUTE statement produces a more efficient object program than other arithmetic statements.)

12.

COMPUTE identifier-1 = { identifier-2 | numeric-literal | arithmetic expression }

According to the form of the COMPUTE statement shown above, which of the following could be correct?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE 5 = TOTAL.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE TOTAL = 5.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE TOTAL = 'TOTALB1'.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE TOTAL = TOTALB1.
```

* * *

b, d
(The symbol = must have at least one space preceding and one space following it.)

13.

COMPUTE identifier-1 = { identifier-2 / numeric literal }

MOVE { identifier-2 / numeric-literal } TO identifier-1

A COMPUTE statement and a MOVE statement of the forms shown above are equivalent.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    COMPUTE TOTAL = BALANCE.
```

If the value of BALANCE is 3742 and the value of TOTAL is 2005, when the statement above is executed, the value of:

a. BALANCE will be changed to 2005.

b. TOTAL will be changed to 3742.

c. BALANCE will be changed to the value of TOTAL.

d. TOTAL will be changed to the value of BALANCE.

* * *

b, d

14.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    COMPUTE SWITCH = 1.
```

When the statement is executed, the value of SWITCH will be changed to ..... .

* * *

1

15.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
       DATA DIVISION.
       FILE SECTION.
       FD  OUTPUT-DISK-FILE
           LABEL RECORDS ARE STANDARD.
       01  DETAIL-RECORD.
           02  ITEM-ID PIC X(6).
           02  RATES PIC $$$.99.
           02  AMOUNT PIC $$$.99.
           02  PRIORITY PIC 99.
       WORKING-STORAGE SECTION.
       77  CLASS PIC 99.
       77  QUANTITY PIC 999V99.
       77  ASSESS PIC 999V99.
```

The picture for identifier-1 in a COMPUTE statement (the identifier to the left of the equal sign) may contain edit characters. The picture for any identifier to the right of the equal sign may contain no edit characters but may contain a V or S; that is, an identifier to the right of the equal sign must be a numeric variable. According to the data description entries above, which of the following statements is correct?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           COMPUTE AMOUNT = QUANTITY.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           COMPUTE QUANTITY = AMOUNT.
```

c.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           COMPUTE RATES = ASSESS.
```

d.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           COMPUTE ASSESS = PRIORITY.
```

e.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
           COMPUTE RATES = 00000.
```

* * *

a, c, d, e

---

16. Write a statement to cause the value of INDICATOR to be changed to 3.

```
                          *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE INDICATOR = 3.

           or

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MOVE 3 TO INDICATOR.
```

---

17. An arithmetic expression may also appear to the right of the equal sign in a COMPUTE statement. An arithmetic expression is a valid combination of numeric literals, numeric variables, and arithmetic operators. Figure 99 shows the arithmetic operators.

Arithmetic Operators

Hierarchy of Evaluation	Operator	Meaning	Example	
			Arithmetic Expression	Cobol Expression
1	+ –	unary plus sign unary minus sign	+2 –2	+2 -2
2	**	exponentiation	3^2	3 ** 2
3	* /	multiplication division	3 × 2 3 ÷ 2	3 * 2 3 / 2
4	+ –	addition subtraction	3 + 2 3 – 2	3 + 2 3 - 2

1) Parentheses modify the order of evaluation; operations enclosed in parentheses are performed first, beginning with the innermost pair of parentheses.

2) When 2 operators of same level of hierarchy appear in the same expression, the operations are performed from left to right.

3) Every operator must be preceded by and followed by a space, except for unary signs.

Figure 99

The arithmetic expression NUMBER1 * NUMBER2 specifies that:

a. NUMBER1 and NUMBER2 are to be added.

b. NUMBER2 is to be divided by NUMBER1.

* * *

Neither (NUMBER1 and NUMBER2 are to be multiplied.)

---

18. Figure 99 shows that the signs + and - can be unary plus and unary minus signs, respectively, or they can specify addition and subtraction. A unary plus or minus sign precedes a single numeric literal or identifier and specifies that the value of the literal or identifier it precedes is to be multiplied by +1 or -1, respectively.

    The value of FACTOR is -2. The value of the arithmetic expression - FACTOR is ..... .

* * *

2

---

19. The COBOL arithmetic expression

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          D / B ** 2 + C
```

    is equivalent to the algebraic expression $D/B^2 + C$. Refer to Figure 99 and show the algebraic equivalent of

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          FACTOR * .5 - TOTAL ** 3.
```

* * *

$.5 \times FACTOR - TOTAL^3$

---

20. Refer to Figure 99 if necessary and write the COBOL expression that is equivalent to XY^2.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          X * Y ** 2.
```

---

21. Figure 99 shows the hierarchy of evaluation, which specifies the order in which operations are performed when more than one operator appears in an expression.

    List the operations in the order in which they will be performed in the expression

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          FACTOR * 5 - TOTAL ** 3.

                           *         *         *
```

**(exponentiation; TOTAL is cubed.)
* (multiplication; 5 and FACTOR are multiplied.)
- (subtraction; TOTAL is subtracted from the product of 5 and FACTOR.)

---

22. Multiplication and division are on the same level in the hierarchy. When these operators appear in the same expression, the operations will be performed from left to right. The same is true for addition and subtraction. Refer to Figure 99 and list the operations in the order in which they will be performed in the expression

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

          TOTAL - FRACTION / 2 * 3 + 1.

                           *         *         *
```

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

---

23. When parentheses are included in an expression, the operation specified within the innermost parentheses is performed first. For example, in the expression

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    ((A + B) * C) ** 2
```

A and B will be added, their sum multiplied by C, and the product then squared.

List the operations in the order in which they will be performed in the expression

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    (X / (2 * Y)) ** 3.
```

* * *

* (multiplication)
/ (division)
** (exponentiation)

---

24. Write a COBOL expression to add TOTAL1 and TOTAL2 and multiply the sum by the product of 2 and TOTAL3.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    (TOTAL1 + TOTAL2) * 2 * TOTAL3
```

25.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
```

```
    COMPUTE TOTAL = COST - YEARS * RATE.
```

When a COMPUTE statement containing an arithmetic expression is executed, the variable specified by the identifier to the left of the equal sign is set equal to the value of the result of the evaluation of the arithmetic expression. If the values of COST, YEAR, and RATE are 10000, 10, and 50 respectively, the value of TOTAL after execution of the statement above will be ..... .

* * *

9500

---

26. Which of the following statements would be used to find the product of the value of RATE and the value of DEPOSIT and store this product in BALANCE?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           MULTIPLY RATE BY DEPOSIT
               GIVING BALANCE.
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           CALCULATE BALANCE = RATE * DEPOSIT.
```

* * *

a (b contains the word CALCULATE instead of COMPUTE.)

---

27. Write a statement to set STANDARD equal to the result of COMMISSION subtracted from the product of MONTHS and AMOUNT.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE STANDARD = MONTHS * AMOUNT
                   - COMMISSION.
```

28.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       77  TAG PIC 99V99.
       77  PERCENT PIC V99.
       77  BILL PIC $99.99.
```

Figure 100 shows the options that can be used in each arithmetic statement. Write a COMPUTE statement to round the value of the product of TAG and PERCENT to the nearest cent and store it in the edited variable BILL.

Summary of Arithmetic Statements and Their Options

Arithmetic Statements \ Allowable Options	GIVING variable-name	variable-name ROUNDED	SIZE ERROR statement	REMAINDER variable-name
ADD { identifier-1 / numeric-literal-1 } [identifier-2 / numeric-literal-2] ... TO identifier-m	X *	X	X	
SUBTRACT { identifier-1 / numeric-literal-1 } [identifier-2 / numeric-literal-2] ... FROM identifier-m	X	X	X	
MULTIPLY { identifier-1 / numeric-literal-1 } BY identifier-2	X	X	X	
DIVIDE { identifier-1 / numeric-literal-1 } INTO identifier-2	X	X	X	X
COMPUTE identifier-1 = { identifier-2 / numeric-literal-1 / arithmetic-expression }		X	X	

* The reserved word TO is omitted when the GIVING option is specified.

Figure 100

```
                    *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           COMPUTE BILL ROUNDED = TAG * PERCENT.
```

You now know how to specify addition, multiplication, division, and subtraction in a COBOL program. You also know how to use a COMPUTE statement to specify any combination of arithmetic operations. A summary of arithmetic statements and the options that may be used in them is given in Figure 100. You may refer to Figure 100 whenever you are coding arithmetic statements.

In the next sequence you will learn to specify channel skipping, a technique for providing vertical spacing in a printed report. Channel skipping requires the use of a carriage control tape, which is looped and mounted in a special compartment on the carriage of the printer. The printing line to be used is determined by an instruction in the program that directs the carriage to move to a punch in the appropriate channel of the carriage control tape; the corresponding line is the printing line.

29.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE OUTPUT-RECORD
               AFTER ADVANCING TO-FIRST-LINE.
```

When you used the AFTER ADVANCING option of the WRITE statement to advance to the first printing line of the next page, you used channel skipping. The mnemonic name specified in the ADVANCING option represented a specific channel on the carriage control tape.

In the statement above a specific channel is specified by:

a. OUTPUT-RECORD.

b. AFTER ADVANCING.

c. TO-FIRST-LINE.

```
                    *       *       *
```

c

---

30. You will recall that the use of a mnemonic name in the AFTER ADVANCING option required that the name be associated with a system name. Which of the following would associate the mnemonic name TO-NEXT-PAGE with a system name?

a.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       WORKING-STORAGE SECTION.
       77  TO-NEXT-PAGE PIC X(13).
```

b.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CONFIGURATION-SECTION.
       SPECIAL-NAMES.
           C01 IS TO-NEXT-PAGE.
```

* * *

b

31.

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE OUTPUT-RECORD
               AFTER ADVANCING TO-NEXT-PAGE.
```

In the preceding frame, the mnemonic name TO-NEXT-PAGE is associated with the system name C01 in the SPECIAL-NAMES paragraph. The system name C01 represents channel 1 on the carriage control tape. The statement above will cause the printer to advance to the line corresponding to the punch in channel 1 of the carriage control tape. If the punch in channel 1 corresponds to the third line on the page, the statement above will cause the record to be written on:

a. line 1 of a new page.

b. line 3 of a new page.

* * *

b
(Line 3 would be the first printing line of the page.)

---

32. The system name C01 represents channel 1 on the carriage control tape. A carriage control tape has 12 channels. You might expect that a mnemonic name could be associated with:

    a. C02 to specify channel 2 on the carriage control tape.

    b. any system name in the sequence C01 through C12.

* * *

Either

---

33. Write the section of a COBOL program necessary to associate the mnemonic name TO-TOTAL-LINES with the system name representing channel 3 of the carriage control tape.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           C03 IS TO-TOTAL-LINES.
```

---

34. Refer to the preceding frame, and write a statement that would cause the contents of BILLING-RECORD to be written on the line corresponding to the punch in channel 3 of the carriage control tape.

* * *

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE BILLING-RECORD
               AFTER ADVANCING TO-TOTAL-LINES.
```

---

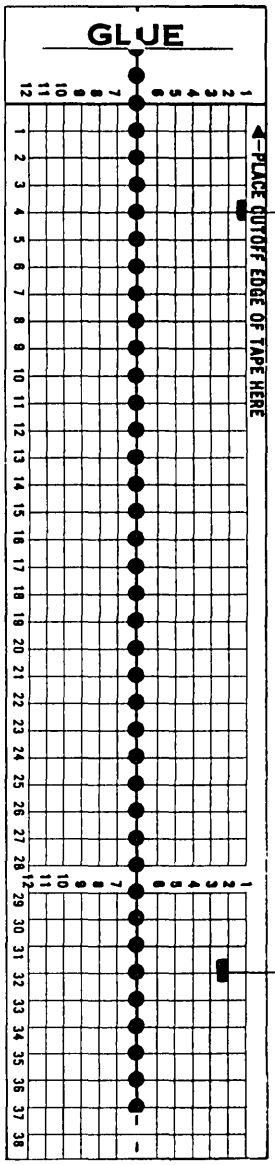
35. The diagram in Figure 101 shows the carriage control tape in position with a printed page. The punches in the tape for channels 1 and 2 are shown and related to the information on the corresponding lines of the page. The diagram also shows where specific records are to be printed on the page. The form has been designed so the heading should appear four lines from the top of the page and the totals on the 32nd line. The working-storage variables and the output area used to produce this report are defined in the Data Division shown in Figure 101. Use the diagram and Data Division and code the following entries.

    1) The section that would associate the mnemonic names TO-NAME-LINE and TO-TOTAL-LINE with the system names representing the channels containing punches

    2) A statement that would cause the values in NAME-LINE to be printed on the fourth line of the page

    3) A statement that would cause the first detail line (record in DETAIL-LINE) to be printed with the spacing indicated in the illustration

    4) A statement that would cause the total line (record in TOTAL-LINE) to be printed on line 32



ALBRITE CHEMICALS 707721

81753	DOUBLE BLOCK HOIST	54.70
81624	50 FOOT ROPE HOIST	14.90

TOTAL $69.60

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       DATA DIVISION.
       FILE-SECTION.
       FD  OUTPUT-FILE
           LABEL RECORDS ARE OMITTED.
       01  INVOICE-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       01  NAME-LINE.
           02  NAME PIC X(20).
           02  FILLER PIC X(6).
           02  FIRM-ID PIC X(6).
           02  FILLER PIC X(89).
       01  DETAIL-LINE.
           02  ITEM-NUMBER PIC X(5).
           02  FILLER PIC X(10).
           02  DESCRIPTION PIC X(25).
           02  FILLER PIC X(5).
           02  PRICE PIC 999.99.
           02  FILLER PIC X(70).
       01  TOTAL-LINE.
           02  FILLER PIC X(30).
           02  TITLE PIC X(5) VALUE 'TOTAL'.
           02  FILLER PIC X(8).
           02  TOTAL PIC $$$$$.99.
           02  FILLER PIC X(70).
```

Figure 101

* * *

1)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           C01 IS TO-NAME-LINE.
           C02 IS TO-TOTAL-LINE.
```

2)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE INVOICE-RECORD
               FROM NAME-LINE
               AFTER ADVANCING TO-NAME-LINE.
```

3)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE INVOICE-RECORD
               FROM DETAIL-LINE
               AFTER ADVANCING 2 LINES.
```

4)

```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           WRITE INVOICE-RECORD
               FROM TOTAL-LINE
               AFTER ADVANCING TO-TOTAL-LINE.
```

SUMMARY

In this lesson you have learned to use the DIVIDE and COMPUTE arithmetic statements. At this point you are able to specify, in COBOL, any computation that might be required in a program. You have also learned to use channel skipping for vertical placement of printed data.

END OF LESSON 21

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 22

LESSON 22 - PROGRAM CODING EXAMPLE

INTRODUCTION

This lesson consists of coding a program called MONTHLY-BILLING. You will find that many of the precepts you learned in the previous lessons are applied here.

This lesson will require approximately three quarters of an hour.

1. Figure 102 gives channel numbers and mnemonic names for use in a billing procedure. Write a SPECIAL-NAMES paragraph to associate the system name for each channel given in Figure 102 with the appropriate mnemonic name.

Vertical Spacing Guide for PRINT-FILE

Channel	Mnemonic name	Data to be printed following skip to channel	
1	TO-NAME-LINE	NAME-LINE STREET-LINE CITY-STATE-LINE	single spaced
2	TO-DETAIL-LINE	DETAIL-LINE DETAIL-LINE . . .	single spaced
3	TO-SUBTOTAL-LINE	SUBTOTAL-LINE TAX-LINE TOTAL-LINE	single spaced

Figure 102

```
                                *       *       *
0    0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       SPECIAL-NAMES.                                          (29)
           C01 IS TO-NAME-LINE.
           C02 IS TO-DETAIL-LINE.
           C03 IS TO-SUBTOTAL-LINE.
```

2.

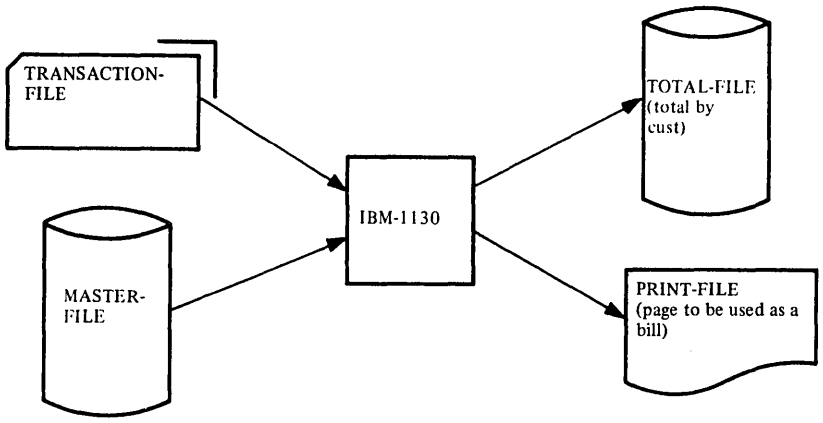


Figure 103

The system flow chart above shows the files used in a program called MONTHLY-BILLING. In this program a record from the master file is to be accessed for each customer in the transaction file. The address from the master file is to be printed on the bill, and then the data from each transaction card for that customer is to be processed (unit price * quantity = VOLUME-PRICE-WORK) and printed. The value of a variable SUBTOTAL-WORK will be increased each time a card record is processed, and when all data cards for one customer have been processed, the value of this variable will be printed as a subtotal. Tax will be calculated and printed, and then added to the subtotal. The total will be printed to complete the bill. Following this, the customer's number, his subtotal and his tax will be placed in the output disk file for subsequent use by another program. A bill is then printed for the next customer, and the sequence is repeated until the transaction file is completed.

Figure 102 gives a guide to vertical spacing for the bill, which is written in PRINT-FILE. Figure 102 indicates that, except for channel skipping, single spacing is to be used throughout.

```
0  0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. MONTHLY-BILLING.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       SPECIAL-NAMES.
           C01 IS TO-NAME-LINE.
           C02 IS TO-DETAIL-LINE.
           C03 IS TO-SUBTOTAL-LINE.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MASTER-FILE
               ASSIGN TO DF-1-600-X.
           SELECT TRANSACTION-FILE
               ASSIGN TO RD-1442.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
           SELECT TOTAL-FILE
               ASSIGN TO DF-2-800-X.
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER-FILE
           BLOCK CONTAINS 8 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  CUSTOMER-MASTER.
           02  PERSONAL-DATA.
               03  NAME PIC X(20).
               03  CUSTOMER-NUMBER PIC X(6).
               03  STREET PIC X(15).
           02  FILLER PIC X(24).
       FD  TRANSACTION-FILE
           LABEL RECORDS ARE OMITTED.
       01  PURCHASE-RECORD.
           02  CUSTOMER-NUMBER PIC X(6).
           02  ITEM-NUMBER PIC X(6).
           02  DESCRIPTION PIC X(15).
           02  UNIT-PRICE PIC 999V99.
           02  QUANTITY PIC 99.
           02  FILLER PIC X(46).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  BILL PIC X(121).
       FD  TOTAL-FILE
           BLOCK CONTAINS 10 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  TOTAL-RECORD.
           02  CUSTOMER-NUMBER PIC X(6).
           02  SUBTOTAL-T PIC 99999V99.
           02  TAX-T PIC 999V99.
```

```
WORKING-STORAGE SECTION.
77  VOLUME-PRICE-WORK PIC 9999V99.
77  SUBTOTAL-WORK PIC 99999V99.
77  TAX-WORK PIC 999V99.
01  NAME-LINE.
    02  FILLER PIC X(9) VALUE SPACES.
    02  NAME PIC X(20).
    02  CUSTOMER-NUMBER PIC X(6).
    02  FILLER PIC X(86) VALUE SPACES.
01  STREET-LINE.
    02  FILLER PIC X(9) VALUE SPACES.
    02  STREET-O PIC X(15).
    02  FILLER PIC X(97) VALUE SPACES.
01  DETAIL-LINE.
    02  FILLER PIC X(5) VALUE SPACES.
    02  ITEM-NUMBER PIC X(6).
    02  FILLER PIC X(5) VALUE SPACES.
    02  DESCRIPTION PIC X(15).
    02  FILLER PIC X(5) VALUE SPACES.
    02  UNIT-PRICE PIC 999.99.
    02  FILLER PIC X(5) VALUE SPACES.
    02  QUANTITY PIC 99.
    02  FILLER PIC X(5) VALUE SPACES.
    02  VOLUME-PRICE PIC 9,999.99.
    02  FILLER PIC X(59) VALUE SPACES.
01  SUBTOTAL-LINE.
    02  FILLER PIC X(52) VALUE SPACES.
    02  SUBTOTAL PIC $$$,$99.99.
    02  FILLER PIC X(59) VALUE SPACES.
01  TAX-LINE.
    02  FILLER PIC X(48) VALUE SPACES.
    02  CONSTANT1 PIC XXX VALUE 'TAX'.
    02  FILLER PIC X(5) VALUE SPACES.
    02  TAX PIC 999.99.
    02  FILLER PIC X(59) VALUE SPACES.
01  TOTAL-LINE.
    02  FILLER PIC X(42) VALUE SPACES.
    02  CONSTANT-2 PIC X(5) VALUE 'TOTAL'.
    02  FILLER PIC X(5) VALUE SPACES.
    02  TOTAL PIC $$$,$99.99.
    02  FILLER PIC X(59) VALUE SPACES.
```

(The optional word IS has been omitted from the VALUE clause)

Figure 104

Figure 104 contains the complete coding for the first three divisions of the program. The SPECIAL-NAMES paragraph is the one that you wrote in the preceding frame. Because numeric-edited variables cannot be used in calculations, it was necessary to include several working-storage numeric variables along with input variables that can be used in the required calculations. Elementary variables in records that will be printed contain some editing symbols. These variables may be used to store values.

Figure 105 is a program flow chart for the Procedure Division of MONTHLY-BILLING. Follow the flow chart and write the Procedure Division to complete MONTHLY-BILLING.

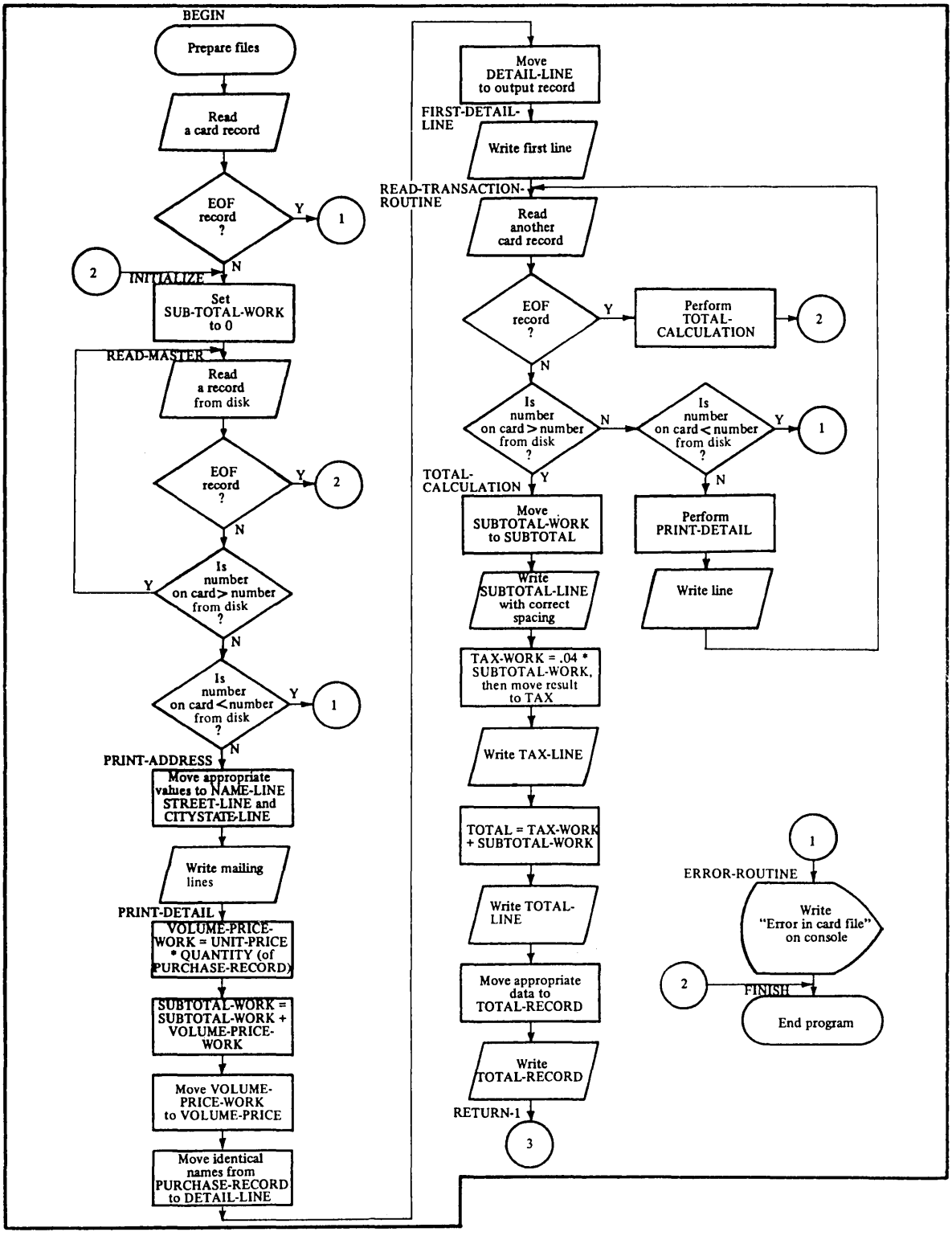

BEGIN
Prepare files
Read a card record
EOF record ?
Y
1
N
2
INITIALIZE
Set SUB-TOTAL-WORK to 0
READ-MASTER
Read a record from disk
EOF record ?
Y
2
N
Is number on card > number from disk ?
Y
N
Is number on card < number from disk ?
Y
1
N
PRINT-ADDRESS
Move appropriate values to NAME-LINE STREET-LINE and CITYSTATE-LINE
Write mailing lines
PRINT-DETAIL
VOLUME-PRICE-WORK = UNIT-PRICE * QUANTITY (of PURCHASE-RECORD)
SUBTOTAL-WORK = SUBTOTAL-WORK + VOLUME-PRICE-WORK
Move VOLUME-PRICE-WORK to VOLUME-PRICE
Move identical names from PURCHASE-RECORD to DETAIL-LINE
Move DETAIL-LINE to output record
FIRST-DETAIL-LINE
Write first line
READ-TRANSACTION-ROUTINE
Read another card record
EOF record ?
Y
Perform TOTAL-CALCULATION
2
N
Is number on card > number from disk ?
N
Is number on card < number from disk ?
Y
1
N
TOTAL-CALCULATION
Y
Move SUBTOTAL-WORK to SUBTOTAL
Write SUBTOTAL-LINE with correct spacing
TAX-WORK = .04 * SUBTOTAL-WORK, then move result to TAX
Write TAX-LINE
TOTAL = TAX-WORK + SUBTOTAL-WORK
Write TOTAL-LINE
Move appropriate data to TOTAL-RECORD
Write TOTAL-RECORD
RETURN-1
3
Perform PRINT-DETAIL
Write line
1
ERROR-ROUTINE
Write "Error in card file" on console
2
FINISH
End program

Figure 105

```
                        *       *       *

0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT MASTER-FILE
                TRANSACTION-FILE
                OUTPUT PRINT-FILE
                TOTAL-FILE.
           READ TRANSACTION-FILE
                AT END GO TO ERROR-ROUTINE.
       INITIALIZE.
           MOVE ZEROS TO SUBTOTAL-WORK.
       READ-MASTER.
           READ MASTER-FILE
                AT END GO TO FINISH.
           IF CUSTOMERNUMBER GREATER THAN
                CUSTOMER-NUMBER OF PERSONAL-DATA
                GO TO READ-MASTER.
           IF CUSTOMERNUMBER LESS THAN
                CUSTOMER-NUMBER OF PERSONAL-DATA
                GO TO ERROR-ROUTINE.
       PRINT-ADDRESS.
           MOVE NAME OF PERSONAL-DATA
                TO NAME OF NAME-LINE.
           MOVE CUSTOMER-NUMBER OF PERSONAL-DATA
                TO CUSTOMER-NUMBER OF NAME-LINE.
           MOVE STREET OF PERSONAL-DATA
                TO STREET OF NAME-LINE.
           MOVE CITY-STATE OF PERSONAL-DATA
                TO CITY-STATE OF NAME-LINE.
           MOVE STREET TO STREET-O.
           MOVE CITY-STATE TO CITY-STATE-O.
           WRITE BILL FROM NAME-LINE
                AFTER ADVANCING TO-NAME-LINE.        (31)
           WRITE BILL FROM STREET-LINE
                AFTER ADVANCING 1 LINE.
           WRITE BILL FROM CITYSTATE-LINE
                AFTER ADVANCING 1 LINE.
       PRINT-DETAIL.
           COMPUTE VOLUME-PRICE-WORK =              (11)
                UNIT-PRICE OF PURCHASE-RECORD
                * QUANTITY OF PURCHASE-RECORD.
           ADD VOLUME-PRICE-WORK
                TO SUBTOTAL-WORK.
           MOVE VOLUME-PRICE-WORK
                TO VOLUME-PRICE.
           MOVE CUSTOMER-NUMBER OF PURCHASE-RECORD
                TO CUSTOMER-NUMBER OF DETAIL-LINE.
           MOVE ITEM-NUMBER OF PURCHASE-RECORD
                TO ITEM-NUMBER OF DETAIL-LINE.
           MOVE DESCRIPTION OF PURCHASE-RECORD
                TO DESCRIPTION OF DETAIL-LINE.
           MOVE UNIT-PRICE OF PURCHASE-RECORD
                TO UNIT-PRICE OF DETAIL-LINE.
           MOVE QUANTITY OF PURCHASE-RECORD
                TO QUANTITY OF DETAIL-LINE.
           MOVE DETAIL-LINE TO BILL.
       FIRST-DETAIL-LINE.
           WRITE BILL
                AFTER ADVANCING TO-DETAIL-LINE.      (31)
       READ-TRANSACTION-ROUTINE.
           READ TRANSACTION-FILE
                AT END PERFORM TOTAL-CALCULATION
                GO TO FINISH.
           IF CUSTOMERNUMBER GREATER THAN
```

```
            CUSTOMER-NUMBER OF PERSONAL-DATA
            GO TO TOTAL-CALCULATION.
        IF CUSTOMERNUMBER LESS THAN
            CUSTOMER-NUMBER OF PERSONAL-DATA
            GO TO ERROR-ROUTINE.
        PERFORM PRINT-DETAIL.
        WRITE BILL
            AFTER ADVANCING 1 LINE.
        GO TO READ-TRANSACTION-ROUTINE.
    TOTAL-CALCULATION.
        MOVE SUBTOTAL-WORK TO SUBTOTAL.
        WRITE BILL FROM SUBTOTAL-LINE                (31)
            AFTER ADVANCING TO-SUBTOTAL-LINE.
        COMPUTE TAX-WORK = .04 *
            SUBTOTAL-WORK.
        MOVE TAX-WORK TO TAX.
        WRITE BILL FROM TAX-LINE
            AFTER ADVANCING 1 LINE.
        COMPUTE TOTAL = TAX-WORK +
            SUBTOTAL-WORK.
        WRITE BILL FROM TOTAL-LINE
            AFTER ADVANCING 1 LINE.
        MOVE CUSTOMER-NUMBER
            OF PERSONAL-DATA
            TO CUSTOMER-NUMBER
            OF TOTAL-RECORD.
        MOVE SUBTOTAL-WORK TO SUBTOTAL-T.
        MOVE TAX-WORK TO TAX-T.
        WRITE TOTAL-RECORD.
    RETURN-1.
        GO TO INITIALIZE.
    ERROR-ROUTINE.
        DISPLAY 'ERROR IN CARD FILE'
            UPON CONSOLE.
    FINISH.
        CLOSE MASTER-FILE
            TRANSACTION-FILE
            PRINT-FILE TOTAL-FILE.
        STOP RUN.
```

Figure 106

SUMMARY

The problem you just completed is a fair summation of many of the major COBOL precepts you have studied up to this point. Perhaps you can now see more clearly the progress that you have made.

END OF LESSON 22

THIS PAGE INTENTIONALLY LEFT BLANK

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

1130 COBOL Text Volume I Printed in U.S.A. SH20-0930-0