IBM

1130 COBOL
Text – Volume II

Programmed Instruction

IBM

13826

1130 COBOL
Text – Volume II

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.

© Copyright International Business Machines Corporation 1971

CONTENTS

LIST OF FIGURES

LESSON 23

LESSON 23 - BRANCHING STATEMENTS (1)

INTRODUCTION

In this lesson you will learn additional ways to specify flow of control in a program using options of the PERFORM statement.

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

THRU option of the PERFORM statement
EXIT statement
TIMES option of the PERFORM statement
UNTIL option of the PERFORM statement
VARYING option of the PERFORM 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..
           PERFORM DISCOUNT.
```

Execution of the statement above would cause paragraph DISCOUNT to be executed and then control to be:

a. returned to the statement directly following the PERFORM statement.

b. passed to the statement directly following DISCOUNT.

* * *

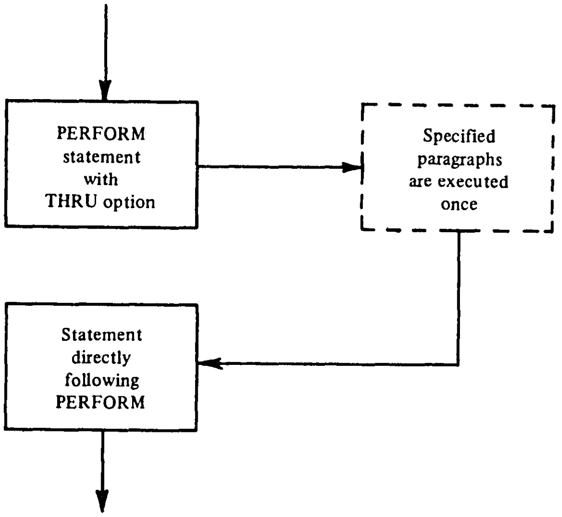
a

PERFORM Statement with the THRU Option

Format PERFORM paragraph-name-1 [THRU paragraph-name-2]

Explanation* When a PERFORM statement with the THRU option is executed, the statements from the first statement in paragraph 1 (entry point) through the last statement in paragraph 2 (exit point) are executed until the exit point is reached once, and then control is returned to the statement directly following the PERFORM statement.

Flow of Logic


PERFORM statement with THRU option
Specified paragraphs are executed once
Statement directly following PERFORM

Rules

1. An embedded GO TO statement may not transfer control out of the range of a PERFORM statement.

2. A common exit point, an EXIT statement may be specified for the range of a PERFORM statement and a transfer of control within that range.

3. An embedded PERFORM statement must have its range either totally in or totally out of the range of the original PERFORM statement; the exit point of the original PERFORM statement may not be included in the range of an embedded PERFORM statement unless it is a common exit point.

*In the explanation, paragraph-1 refers to the paragraph named paragraph-name-1 and paragraph-2 refers to the paragraph named paragraph-name-2 in the format above.

(Dashed lines show the effect of using the PERFORM statement with the THRU option. An understanding of this effect is necessary for proper use of the statement, but the programmer is not required to code the logic indicated by dashed lines.)

Figure 107

2. Refer to the format of the PERFORM statement with the THRU option in Figure 107. Two paragraph names are specified in this format. When the PERFORM statement is executed, all of the statements from the first statement in paragraph-1 through the last statement in paragraph-2 will be executed. Although the two paragraphs may be separated by intervening paragraphs, they must be in a linear sequence. Paragraphs P1, P2, P3, and P4 are in linear sequence. Which of the following statements specifies execution of paragraphs P1, P3, and P4?

   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..

           PERFORM P1 P3 P4.
```

   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 P1 THRU P4.
```

   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..

           PERFORM P1.
           PERFORM P3 THRU P4.
```

   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..

           PERFORM P1.
           PERFORM P3.
           PERFORM P4.
```

*   *   *

c,d
(a is an incorrect format; b would execute P2 also.)

---

3. The flow of logic diagram in Figure 107 shows the flow of control that occurs when the PERFORM statement with the THRU option is used. Referring to the format and flow of logic diagram in Figure 107, write a statement that would transfer control to paragraph MORTGAGE, execute MORTGAGE and the two subsequent paragraphs M-1 and M-2, and then return control to the statement directly following the statement that 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 MORTGAGE THRU M-2.
```

---

4. The range of a PERFORM statement with the THRU option includes all the statements in paragraph-1, all the statements in paragraph-2, and all the statements in intervening paragraphs that are part of the linear sequence. If THRU paragraph-name-2 were not specified, the range would be:

   a. all the statements in paragraph-1.

   b. all the statements in paragraph-2.

   c. all the statements in intervening paragraphs that are part of the linear sequence.

* * *

a

---

5. Rule 1 in Figure 107 states that an embedded GO TO statement may not transfer control out of the range of the PERFORM statement. This means that if THRU paragraph-name-2 is specified:

   a. paragraph-1 may contain a GO TO statement transferring control to paragraph-2.

   b. paragraph-2 may contain a GO TO statement transferring control to paragraph-1.

   c. paragraph-1 may contain a GO TO statement transferring control to a paragraph that is not in the linear sequence from paragraph-1 through paragraph-2.

   d. paragraph-1 may contain a GO TO statement transferring control to any paragraph within paragraph-1 through paragraph-2.

* * *

a,b,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..

           PERFORM DISCOUNT THRU PRINT.
```

The paragraphs DISCOUNT, FIGURE, ERROR-ROUTINE, and PRINT are in a linear sequence. Which of the following GO TO statements could be inserted into paragraph DISCOUNT if control is transferred to DISCOUNT by the statement above?

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 PRINT.
```

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 FIGURE.
```

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..

           GO TO READ-CARD.
```

* * *

a,b

----

7. A GO TO statement embedded in the range of a PERFORM with the THRU option can be used to bypass:

   a. any portion of the range.

   b. any portion of the range except the paragraph specified in the THRU option.

* * *

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..

       PARA1.
           .
           .
           .
       PARA2.
           .
           .
           .
       PARA3.
           .
           .
           .
       PARA4.
           .
           .
           .
           COMPUTE ...
           WRITE ...
       PARAX.
           PERFORM PARA1 THRU PARA4.
       PARA5.
           .
           .
           .
```

Match the effects below with the statements that might be included in an IF statement in PARA1 of the program segment shown above if PARAX activates execution of the sequence PARA1 through PARA4.

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..

           GO TO PARA4.
```

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..

           GO TO PARA5.
```

a. Would not return control immediately to the statement directly following PERFORM

b. Would be invalid since control cannot be transferred out of the range of a PERFORM by a GO TO statement

c. Would return control immediately to the statement directly following PERFORM

*   *   *

1) a
2) b

You have now learned to bypass any portion of the range of a PERFORM except the last paragraph. It is frequently necessary to return control immediately to the statement directly following the PERFORM, bypassing the remaining portion of the range. In the next sequence of frames you will learn to set up a special paragraph that will allow you to return control immediately to the statement directly following the PERFORM.

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 DISCOUNT THRU EXIT-PARAGRAPH.
               .
               .
               .
           DISCOUNT.
               .
               .
               .
           EXIT-PARAGRAPH.
               EXIT.
```

A paragraph containing a single statement, shown above as EXIT-PARAGRAPH, may be specified as <u>paragraph-name-2</u> in the THRU option in a PERFORM statement. Transfer of control from some point within the range of the PERFORM to the paragraph containing only the EXIT statement would then cause control to be returned to the statement directly following the PERFORM statement. To return control from paragraph DISCOUNT above to the statement following the PERFORM statement, you would write in DISCOUNT:

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..

           PERFORM EXIT-PARAGRAPH.
```

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 EXIT-PARAGRAPH.
```

*   *   *

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..

       PARA1.
           .
           .
           .
       PARA2.
           .
           .
           .
       PARA3.
           .
           .
           .
       PARA3A.
           .
           .
           .
           COMPUTE ...
           WRITE ...
       PARA4.
           EXIT.
       PARAX.
           PERFORM PARA1 THRU PARA4.
       PARA5.
           .
           .
           .
```

When a paragraph containing only the EXIT statement is specified in a GO TO statement that is also embedded in the range of the PERFORM, control will be returned immediately to the statement directly following the PERFORM statement. Match the following effects with the statements suggested that might be included in PARA1 in the example above.

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..

           GO TO PARA4.
```

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..

           GO TO PARA5.
```

a. Would not return control immediately to the statement directly following PERFORM

b. Would be invalid since control cannot be transferred out of the range of a PERFORM by a GO TO statement

c. Would return control immediately to the statement directly following PERFORM

*   *   *

1) c
2) b

----

11. The last statement in the last paragraph in the range of a PERFORM is the exit point for the range of that PERFORM. The EXIT statement may be used as an exit point for the range of any PERFORM statement with the THRU option. For the paragraphs DISCOUNT, SUBTRACT-DISCOUNT, and POINT-OF-EXIT which are in a linear sequence, write:

    1) a statement to specify execution of paragraph DISCOUNT, paragraph SUBTRACT-DISCOUNT, and paragraph POINT-OF-EXIT.

    2) paragraph POINT-OF-EXIT so that the statement GO TO POINT-OF-EXIT can be inserted in either DISCOUNT or SUBTRACT-DISCOUNT to cause control to return immediately to the statement directly following the PERFORM 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..

           PERFORM DISCOUNT
               THRU POINT-OF-EXIT.
```

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..

           POINT-OF-EXIT.
               EXIT.
```

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..

           POINT-OF-EXIT.
               EXIT.
```

The name of the paragraph shown above has been specified in the THRU option of a PERFORM statement. POINT-OF-EXIT may include the EXIT statement and:

a. a GO TO statement transferring control to any paragraph within the range of the PERFORM statement.

b. no other statements.

*   *   *

b

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..

           PERFORM DISCOUNT
               THRU POINT-OF-EXIT.
```

The PERFORM statement above is one that you wrote in a preceding frame. The range of this PERFORM statement includes the linear sequence of paragraphs DISCOUNT, SUBTRACT-DISCOUNT, and POINT-OF-EXIT. POINT-OF-EXIT includes the EXIT statement. Match each GO TO statement with the paragraph in which it could be placed.

1) DISCOUNT

2) SUBTRACT-DISCOUNT

3) POINT-OF-EXIT

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 POINT-OF-EXIT.
```

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 DISCOUNT.
```

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..

           GO TO SUBTRACT-DISCOUNT.
```

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..

           GO TO PRINT-DISCOUNT.
```

e. (none of these)

* * *

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

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..

           PERFORM DISCOUNT THRU P-3.


           .
           .
           .
       DISCOUNT.
           .
           .
           .
       ERRORS.
           .
           .
           .
       P-1.
           .
           .
           .
       P-2.
           .
           .
           .
       P-3.
           EXIT.
       P-4.
           .
           .
           .
```

One PERFORM statement may be embedded within the range of another PERFORM statement. Rule 3 in Figure 107 states that an embedded PERFORM must have its range totally in or totally out of the range of the original PERFORM. The rule also states that the exit point of the original PERFORM must not be included in the range of the embedded PERFORM except as a common exit point. The PERFORM statement shown above specifies that control is to be transferred to DISCOUNT in the linear sequence shown in the diagram. Decide whether it would be valid or invalid to embed each PERFORM statement below in the indicated paragraph of the linear sequence.

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 ERRORS.
```

(in paragraph P-2)

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..

           PERFORM P-1 THRU P-3.
```

(in paragraph DISCOUNT)

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..

           PERFORM P-2 THRU P-4.
```

(in paragraph ERRORS)

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..

           PERFORM P-4.
```

(in paragraph P-2)

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 P-4.
```

(in paragraph P-3)

*   *   *

1) valid (Range totally in range of original PERFORM)

2) valid (Range totally in range of original PERFORM; exit point may be included in range of embedded PERFORM if it is a common exit point.)

3) invalid (Range not totally in or out of range of original PERFORM; exit point of original PERFORM is within the range of the embedded PERFORM and is not a common exit point.)

4) valid (Range totally out of range of original PERFORM)

5) invalid (EXIT must be the only statement in the paragraph.)

15. On the basis of the range of the PERFORM statement in paragraph I-PARA of the linear sequence shown below, decide whether each of the following statements would be valid or invalid if placed in paragraph C-PARA.

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 D-PARA THRU E-PARA.
```

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..

           PERFORM E-PARA THRU J-PARA.
```

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..

           PERFORM E-PARA THRU F-PARA.
```

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..

           GO TO E-PARA.
```

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..

           GO TO H-PARA.
```

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..

           GO TO F-PARA.
```

```
     .
     .
     .
A-PARA.
     .
     .
     .
B-PARA.
     .
     .
     .
C-PARA.
     .
     .
     .
D-PARA.
     .
     .
     .
E-PARA.
     .
     .
     .
F-PARA.
     EXIT.
G-PARA.
     .
     .
     .
H-PARA.
     .
     .
     .
I-PARA.
     .
     .
     .
     PERFORM B-PARA THRU F-PARA.
J-PARA.
     .
     .
     .
```

*   *   *

1) valid (Range totally in range of original PERFORM)

2) invalid (Range not totally in or out of range of original PERFORM; exit point of original PERFORM is included in the range of the embedded PERFORM and is not a common exit point.)

3) valid (Range totally in range of original PERFORM; exit point may be included in range of embedded PERFORM if it is a common exit point.)

4) valid (Does not transfer control out of range of the PERFORM)

5) invalid (Transfers control out of the range of the PERFORM)

6) valid (Transfers control to exit point)

The THRU option of the PERFORM statement, which you have learned to use in the preceding sequence, may be used with all other options of the PERFORM statement that you will learn in succeeding frames. Regardless of the options used, when execution of the paragraphs specified in the PERFORM statement is completed, control is always returned to the statement directly following the PERFORM statement. Either one or two paragraph names may be specified in all PERFORM statements, and the EXIT statement may be used as the last paragraph. In the next sequence you will learn to use the other options of the PERFORM statement.

PERFORM Statement with the TIMES Option

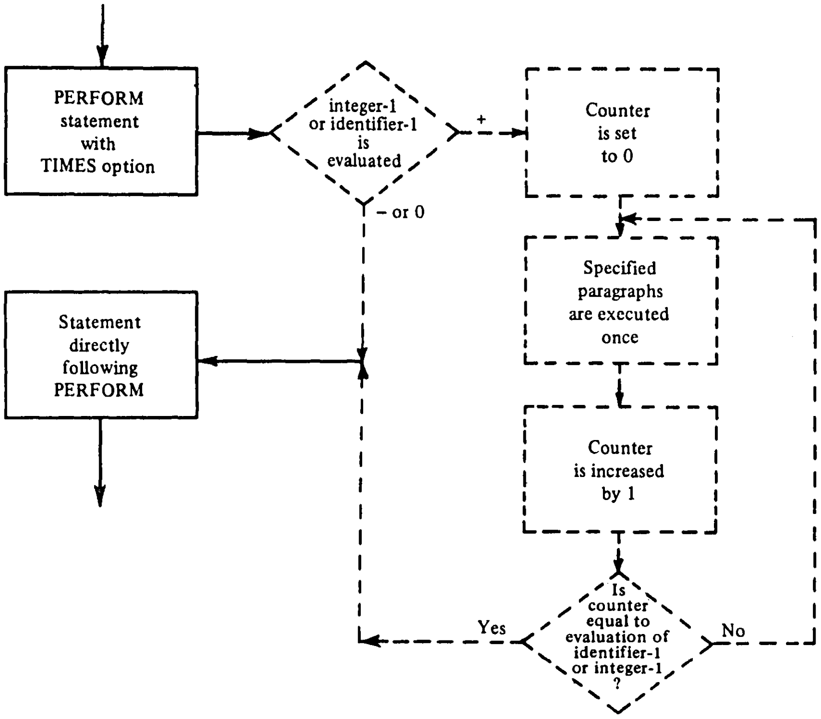
Format

PERFORM paragraph-name-1 [THRU paragraph-name-2]

$\left\{\begin{array}{l}\text{integer-1}\\ \text{identifier-1}\end{array}\right\}$ TIMES

Explanation When a PERFORM statement with the TIMES option is executed, control goes to the entry point and then continues normally until it passes through the exit point integer-1 or identifier-1 times before control is returned to the statement directly following the PERFORM statement. If the value of identifier-1 or integer-1 is negative or zero, the specified paragraphs will not be executed.

Flow of Logic


PERFORM statement with TIMES option
integer-1 or identifier-1 is evaluated
+
Counter is set to 0
- or 0
Specified paragraphs are executed once
Statement directly following PERFORM
Counter is increased by 1
Is counter equal to evaluation of identifier-1 or integer-1 ?
Yes
No

Rules

1. Identifier-1 must have an integer value.

2. A change in the value of identifier-1 after the PERFORM statement is executed but before control is returned to the statement directly following the PERFORM statement will not change the number of times the specified paragraphs are executed.

(Dashed lines show the effect of using the PERFORM statement with the TIMES option. An understanding of the effect is necessary for the proper use of the statement, but the programmer is not required to code the logic indicated by the dashed lines.)

Figure 108

16. Figure 108 shows the PERFORM statement with the TIMES option. According to the explanation and format in Figure 22, the use of the TIMES option differs from the use of the PERFORM statement with only the THRU option in that:

    a. an integer or identifier is used to specify the number of times the specified paragraphs are to be executed.

    b. paragraph-name-2 must be specified when the TIMES option is used.

* * *

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..

           PERFORM DISCOUNT 5 TIMES.

0   0    1    1    2    2    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 DISCOUNT ITEMS TIMES.
```

Match the following words from the statements shown above with the terms from the format of the PERFORM statement in Figure 108.

1) paragraph-name-1	a. 5
2) identifier-1	b. DISCOUNT
3) integer-1	c. ITEMS
	d. TIMES

* * *

1) b
2) c
3) a

---

18. According to Rule 1 in Figure 108 identifier-1 must have an integer value. If the value of the integer or identifier is zero or negative, the specified paragraphs will not be executed. Match the effects below with the values of identifier-1 or integer-1 in the TIMES option of the PERFORM statement.

1) -2 a. Invalid value (not an integer)

2) 2.8 b. Specified paragraph(s) will not be executed.

3) 26 c. Specified paragraph(s) will be executed - 2 times.

4) 0 d. Specified paragraph(s) will be executed the indicated number of times.

* * *

1) b
2) a
3) d
4) b

---

19. The flow of logic diagram in Figure 108 shows that control is returned to the statement directly following the PERFORM statement when:

a. identifier-1 or integer-1 is zero or negative.

b. the specified number of executions have been accomplished.

* * *

Either

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..

               .
               .
               .
               PERFORM ROUTINE AGE TIMES.
               .
               .
               .
           ROUTINE.
               .
               .
               .
               ADD 1 TO AGE.
               .
               .
               .
```

Rule 2 in Figure 108 states that a change in the value of identifier-1 after execution of the PERFORM statement begins does not change the number of times the paragraphs are executed. Paragraph ROUTINE in the program segment shown above will be executed:

a. AGE + 1 times since the original value of AGE is incremented by 1 in ROUTINE.

b. AGE times since a change in the value of identifier-1 does not change the number of times it is executed.

* * *

b

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..

           .
           .
           .
       INITIALIZE.
           MOVE ZEROS TO LIST-TIMES.
           ACCEPT TOTAL FROM CONSOLE.
           ACCEPT YEARS FROM CONSOLE.
           MOVE .01 TO PERCENT.
           MULTIPLY 4 BY YEARS.
               GIVING LENGTH.
           .
           .
           .
       COMPOUND.
           ADD 1 TO LIST-TIMES.
           COMPUTE TOTAL =
               TOTAL + TOTAL * PERCENT.
           MOVE TOTAL TO LIST-TOTAL.
           WRITE LIST-RECORD.
           .
           .
           .
```

Paragraph INITIALIZE sets certain values to be used in paragraph COMPOUND. Paragraph COMPOUND calculates and prints a new value of TOTAL in a numbered sequence as compound interest is calculated for the total amount and the number of years keyed in through the console typewriter. Since interest is to be compounded quarterly, COMPOUND is to be executed four times for every year in which the total amount will be available to the lending institution. Write a statement to follow the MULTIPLY statement specifying execution of COMPOUND to calculate and print values of TOTAL for as many years as necessary.

*   *   *

```
0   0    1    1    2    2    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 COMPOUND LENGTH TIMES.
```

PERFORM Statement with the UNTIL Option

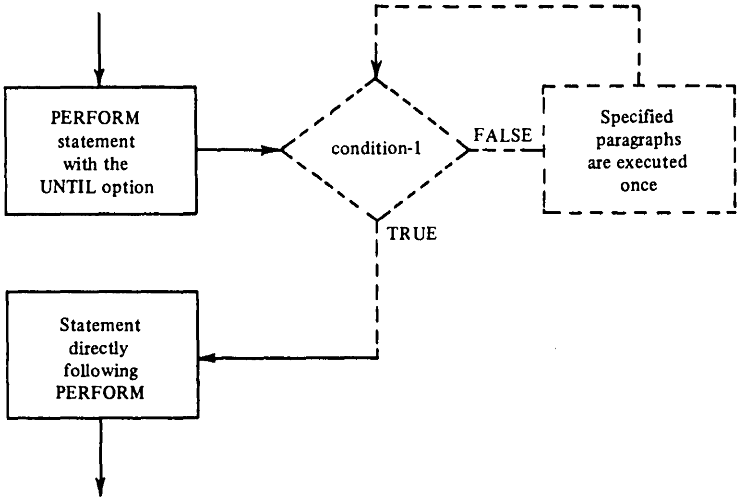
Format

PERFORM paragraph-name-1 [THRU paragraph-name-2]

UNTIL condition-1

Explanation When a PERFORM statement with the UNITL option is executed, the statements from the entry point to the exit point are executed repeatedly until condition-1 is true. Control is then returned to the statement directly following the PERFORM statement. If condition-1 is true at the time the PERFORM statement is executed, the specific paragraphs will not be executed.

Flow of Logic


PERFORM statement with the UNTIL option
condition-1
FALSE
Specified paragraphs are executed once
TRUE
Statement directly following PERFORM

Rule Condition-1 must be a relational condition or a condition-name condition.

(Dashed lines show the effect of using a PERFORM statment with the UNTIL option. An understanding of this effect is necessary for proper use of the statement, but the programmer is not required to code the logic indicated by the dashed lines.)

Figure 109

22. Figure 109 represents still another format of the PERFORM statement. A paragraph specified in a PERFORM statement with the UNTIL option is executed as many times as necessary for the condition to become true. You can infer from this that a paragraph executed because of a PERFORM statement with the UNTIL option:

a. must make some change in the value of a variable specified in condition-1.

b. might be repeated endlessly if condition-1 never became true.

* * *

Both

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..

           PERFORM REORDER
               UNTIL ORDER
               IS GREATER THAN 700.
```

When the statement above is executed, paragraph REORDER will be executed:

a. repeatedly until the value of ORDER is 701 or greater.

b. endlessly if the value of ORDER is 700 or less and is not changed within REORDER.

* * *

Both

----

24. The flow of logic diagram in Figure 109 shows that when a PERFORM statement with the UNTIL option is executed the condition is tested before the specified paragraph(s) are executed. In order to execute procedure PART-PROCESS once for every part number from 31 through 74, you would first move 31 to PART-NUMBER, then make sure that PART-PROCESS increments PART-NUMBER by 1, and finally code 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..

           PERFORM PART-PROCESS UNTIL
               PART-NUMBER IS EQUAL TO 74.
```

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 PART-PROCESS UNTIL
               PART-NUMBER IS EQUAL TO 75.
```

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..

           PERFORM PART-PROCESS UNTIL
               PART-NUMBER IS GREATER THAN 74.
```

* * *

b,c
(a would return control as soon as PART-NUMBER is equal to 74, without executing PART-PROCESS for that part.)

----

25. The PERFORM statement to solve the problem in the preceding frame could also have been written using the TIMES option. Refer to Figure 108 and then write an alternate statement for the problem in the preceding 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..

           PERFORM PART-PROCESS 44 TIMES.
```

----

26. Write a statement that would cause the paragraphs P-1 through P-7 to be executed repeatedly until the value of BALANCE is larger than the value of MAXIMUM-CREDIT.

* * *

```
0   0    1    1    2    2    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 P-1 THRU P-7 UNTIL
           BALANCE IS GREATER THAN
           MAXIMUM-CREDIT.
```

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..

       COMPOUND.
           ADD 1 TO LIST-TIMES.
           COMPUTE TOTAL =
               TOTAL + TOTAL * PERCENT.
           MOVE TOTAL TO LIST-TOTAL.
           WRITE LIST-RECORD.
```

You wish to specify execution of the paragraph above to find out how long it will take for the initial investment to double. Assume that the initial investment (TOTAL) was 200.00 and write a statement that will cause the appropriate number of executions.

* * *

```
0   0    1    1    2    2    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 COMPOUND
           UNTIL TOTAL IS GREATER THAN 400.
```

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..

       02 PART-NUMBER PIC 9999.
           88 SMALL VALUES ARE 0000 THRU 0299.
           88 MEDIUM VALUES ARE 0300 THRU 0699.
           88 LARGE VALUES ARE 0700 THRU 1000.
```

Part numbers are in ascending sequence in a stock file. As a programmer you wish to execute a paragraph called PART-PROCESS for all small parts and have defined the conditional variable shown above. Using a condition-name condition, write a statement to accomplish the desired 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..

           PERFORM PART-PROCESS
               UNTIL MEDIUM.
```

PERFORM Statement with the VARYING Option

Format

PERFORM paragraph-name-1 [THRU paragraph-name-2]

VARYING identifier-1

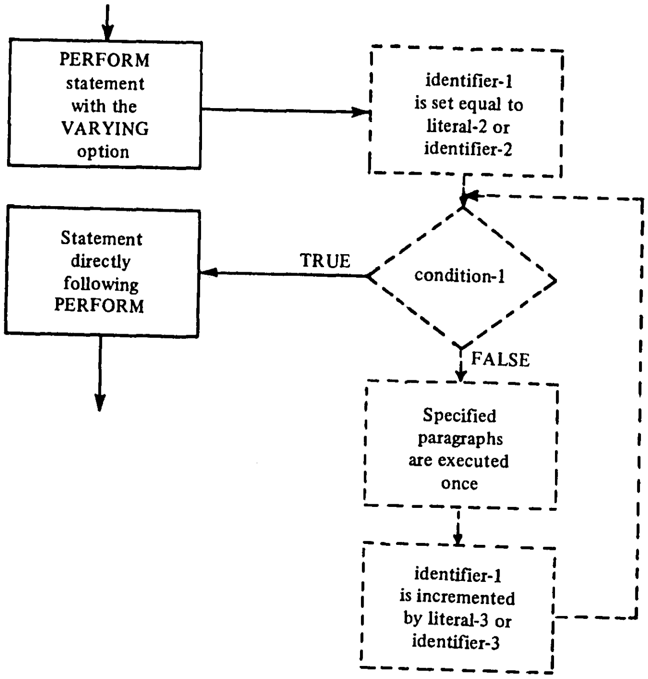
$$\underline{\text{FROM}} \left\{\begin{matrix}\text{literal-2} \\ \text{identifier-2}\end{matrix}\right\} \quad \underline{\text{BY}} \quad \left\{\begin{matrix}\text{literal-3} \\ \text{identifier-3}\end{matrix}\right\}$$

UNTIL condition-1

Explanation

When a PERFORM statement with the VARYING option is executed, the statements from the entry point to the exit point are executed repeatedly until condition-1 is true, incrementing the value of identifier-3 or literal-3 beginning at the value of identifier-2 or literal-2. If condition-1 is true at the time the PERFORM statement is executed, the specified paragraphs will not be executed.

Flow of Logic


PERFORM statement with the VARYING option
identifier-1 is set equal to literal-2 or identifier-2
Statement directly following PERFORM
TRUE
condition-1
FALSE
Specified paragraphs are executed once
identifier-1 is incremented by literal-3 or identifier-3

Rules

1. A change in the value of identifier-2 after the PERFORM statement is executed but before control is returned to the statement directly following the PERFORM statement will have no effect on the number of times the specified paragraphs are executed.

2. A change in the value of identifier-1 or identifier-3 after the PERFORM statement is executed but before control is returned to the statement directly following the PERFORM statement will affect the number of times the specified paragraphs are executed.

(Dashed lines show the effect of using the PERFORM statement with the VARYING option. An understanding of this effect is necessary for the proper use of the statement but the programmer is not required to code the logic indicated by the dashed lines.)

Figure 110

29. Figure 110 shows the PERFORM statement with the VARYING option. Which of the following statements is the correct form for a PERFORM statement with the VARYING 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..

        PERFORM P-1 THRU P-4
            VARYING ITEM FROM 7
            UNTIL ITEM EQUAL TO 12.
```

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 P-1
            VARYING ITEM
            FROM 1 BY 2 TO 7.
```

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..

        PERFORM P-1
            VARYING SHELL FROM BASIC
            BY QUOTE
            UNTIL PRICE
            IS GREATER THAN 800.00.
```

* * *

c

(a omitted BY $\left\{\begin{array}{l}\underline{\text{literal-3}}\\ \underline{\text{identifier-3}}\end{array}\right\}$;

b includes TO 7, which is not included in the format.)

SUMMARY

In this lesson you have learned to use the THRU, TIMES, UNTIL, and VARYING options of the PERFORM statement. You have learned to use the EXIT statement as an exit point for embedded PERFORM and GO TO statements.

All of the topics presented in this lesson are related to specifying flow of control within a program. It is also possible to refer to subroutines that are not contained in the main COBOL program. Such subroutines will operate on variables from the main program and may be written in either COBOL or the Basic Assembler Language. The Linkage Section must be written into the Data Division of a COBOL program when subroutines are to be used.

Sometimes the computer system does not have sufficient available core storage for a complex program. Such a problem might be solved by using subroutines, as described above.

END OF LESSON 23

LESSON 24

LESSON 24 - BRANCHING STATEMENTS (2)

INTRODUCTION

In this lesson you will learn still other methods of specifying flow of control in a program, going beyond basic use of the PERFORM statement.

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

DEPENDING ON option of the GO TO statement
ALTER statement
GO TO statement without paragraph

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..

           PERFORM DEPRECIATION
               VARYING VALUATION
               FROM SALVAGE BY RATE
               UNTIL VALUATION
               IS GREATER THAN COST.
```

On the basis of the rules in Figure 110 and the PERFORM statement above, determine which variable below, if changed, would change the number of times the paragraph is executed.

a. VALUATION

b. SALVAGE

c. RATE

*   *   *

a,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..

           PERFORM DEPRECIATION
               VARYING VALUATION
               FROM SALVAGE BY RATE
               UNTIL VALUATION
               IS GREATER THAN COST.
```

The diagram of flow of logic in Figure 110 shows how the number of times the procedure is executed is determined. Applying the diagram in Figure 110 to the statement above, you can see that VALUATION is set equal to .... and .... is tested before the paragraph is executed.

*   *   *

SALVAGE
VALUATION IS GREATER THAN COST or the condition

---

3. As soon as the condition specified in a PERFORM statement is true, control returns to the statement directly following PERFORM. As shown in the flow of logic diagram in Figure 110, control might be returned to this statement:

   a. before the paragraph is ever executed.

   b. after many executions of the paragraph.

   c. after one execution of the paragraph.

*   *   *

Any of these

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..

           PERFORM DEPRECIATION
               VARYING VALUATION
               FROM SALVAGE BY RATE
               UNTIL VALUATION
               IS GREATER THAN COST.
```

If the condition in the statement above is false when tested, the value of .... is incremented by the value of ....

*   *   *

VALUATION
RATE

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 DEPRECIATION
               VARYING VALUATION
               FROM SALVAGE BY RATE
               UNTIL VALUATION
               IS GREATER THAN COST.
```

According to the flow of logic diagram in Figure 110, as soon as VALUATION is incremented:

a. control is returned to the statement directly following PERFORM.

b. VALUATION is compared with COST.

c. paragraph DEPRECIATION is executed again.

*   *   *

b

---

6. Write a statement to execute paragraph OUTSTANDING-BALANCE, setting ENDING-BALANCE to PRESENT-BALANCE, and then incrementing by the value of SUBTRACT-PAYMENT until the value of ENDING-BALANCE is less than the value of PAYMENT.

* * *

```
0   0    1    1    2    2    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 OUTSTANDING-BALANCE
               VARYING ENDING-BALANCE
               FROM PRESENT-BALANCE
               BY SUBTRACT-PAYMENT
               UNTIL ENDING-BALANCE
               IS LESS THAN PAYMENT.
```

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..

       CONVERSION-ROUTINE.
           COMPUTE CENTIGRADE = 5 / 9 *
               (TEMPERATURE - 32).
           MOVE CENTIGRADE TO DEGREES-C.
           MOVE TEMPERATURE TO DEGREES-F.
           WRITE OUT-RECORD.
```

The paragraph above will convert a value of TEMPERATURE, which represents Fahrenheit degrees, to the centigrade scale, and then print the two values as a line on a page. As a programmer for a chemical company you might find it useful to have a table printed, listing the equivalent temperatures in both scales. Write a statement that will give you such a table for all Fahrenheit temperatures that end in 0 or 5 between -40° and 120°.

* * *

```
0   0    1    1    2    2    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 CONVERSION-ROUTINE
               VARYING TEMPERATURE
               FROM -40 BY 5
               UNTIL TEMPERATURE
               IS GREATER THAN 120.
```

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..

       MORTGAGE.
           COMPUTE INTEREST-PAYMENT =
               INTEREST * BALANCE.
           IF INTEREST-PAYMENT
               IS GREATER THAN PAYMENT
               GO TO ERROR-ROUTINE.
           COMPUTE PRINCIPAL-PAYMENT =
               PAYMENT - INTEREST-PAYMENT.
           COMPUTE BALANCE = BALANCE -
               PRINCIPAL-PAYMENT.
           MOVE MONTH TO PRINT-MONTH.
           MOVE PAYMENT TO PRINT-PAYMENT.
           MOVE INTEREST-PAYMENT
               TO PRINT-INTEREST-PAYMENT.
           MOVE BALANCE TO PRINT-BALANCE.
           WRITE PRINT-RECORD.
           GO TO EXIT-POINT.
       ERROR-ROUTINE.
           DISPLAY 'LARGER PAYMENT NEEDED'
               UPON CONSOLE.
           ADD DURATION TO MONTH.
       EXIT-POINT.
           EXIT.
```

The three paragraphs shown above occur in a program used by a real estate firm to compute and print the month, interest payment, and balance due for a mortgage. (The variables BALANCE, INTEREST, and PAYMENT have been set to appropriate values.) The value of DURATION has been set equal to the number of months of the mortgage. Write a statement to cause the paragraphs above to be executed for each month of the mortgage, using MONTH to count the number of executions.

* * *

```
0   0    1    1    2    2    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 MORTGAGE THRU EXIT-POINT
               VARYING MONTH FROM 1 BY 1
               UNTIL MONTH IS GREATER THAN
               DURATION.
```

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..

       MORTGAGE.
           COMPUTE INTEREST-PAYMENT =
               INTEREST * BALANCE.
           IF INTEREST-PAYMENT
               IS GREATER THAN PAYMENT
               GO TO ERROR-ROUTINE.
           COMPUTE PRINCIPAL-PAYMENT =
               PAYMENT - INTEREST-PAYMENT.
           COMPUTE BALANCE = BALANCE -
               PRINCIPAL-PAYMENT.
           MOVE MONTH TO PRINT-MONTH.
           MOVE PAYMENT TO PRINT-PAYMENT.
           MOVE INTEREST-PAYMENT
               TO PRINT-INTEREST-PAYMENT.
           MOVE PRINCIPAL-PAYMENT TO
               PRINT-PRINCIPAL-PAYMENT.
           MOVE BALANCE TO PRINT-BALANCE.
           WRITE PRINT-RECORD.
           GO TO EXIT-POINT.
       ERROR-ROUTINE.
           DISPLAY 'LARGER PAYMENT NEEDED'
               UPON CONSOLE.
           MOVE ZEROS TO BALANCE.
       EXIT-POINT.
           EXIT.
```

The paragraphs above are to be executed until the value of BALANCE is less than or equal to (not greater than) zero. The number of the execution is again to be printed as the value of MONTH, along with the total payment, the interest payment, and the new balance. To accomplish this, you would write 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..

           PERFORM MORTGAGE THRU EXIT-POINT
               UNTIL BALANCE
               IS NOT GREATER THAN ZERO.
```

and insert 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..

           ADD 1 TO MONTH.
```

into paragraph MORTGAGE.

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 MORTGAGE THRU EXIT-POINT
               VARYING MONTH FROM 1 BY 1
               UNTIL BALANCE
               IS NOT GREATER THAN ZERO.
```

* * *

Either

---

10. Match the following.

1) PERFORM with only the THRU option

2) PERFORM with TIMES option

a. Could be used to execute a paragraph or paragraphs once

b. Could be used to cause seven executions of a paragraph

c. Might cause no executions of the paragraph if the value of identifier-1 were zero or negative

* * *

1) a
2) a,b,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..

           GO TO MANAGER ANALYST OPERATOR
               PROGRAMMER
               DEPENDING ON JOB.
```

Read Figure 111. The statement above illustrates a GO TO statement with the DEPENDING ON option. It specifies that control will be transferred to MANAGER if the value of JOB is 1, to ANALYST if the value of JOB is 2, to OPERATOR if the value of JOB is 3, and to PROGRAMMER if the value of JOB is 4. When a GO TO statement with the DEPENDING ON option is executed:

a. control is always transferred to the first paragraph listed.

b. only one specific value of identifier will cause a transfer of control.

c. control is transferred to the paragraph whose position is represented by the value of identifier.

* * *

c

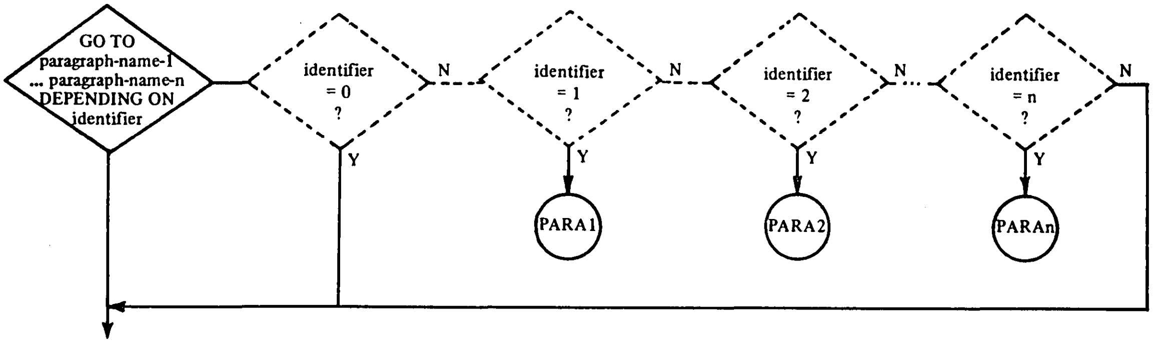
GO TO Statement with the DEPENDING ON Option

Format GO TO paragraph-name-1 [paragraph-name-2... paragraph-name-n]

DEPENDING ON identifier

Explanation When a GO TO statement with the DEPENDING ON option is executed, control is transferred to one of a series of paragraphs depending on the value of the identifier. If the value of identifier is zero or greater than the number of paragraph names specified, the GO TO statement is ignored and control goes to the next sentence in sequence.

Flow of Logic



Rules

1. Identifier must represent a positive or unsigned integer.
2. The value of identifier represents the number of the paragraph-name to which control will be transferred.

(Dashed lines show the effect of using the GO TO statement with the DEPENDING ON option. An understanding of this effect is necessary for proper use of the statement, but the programmer is not required to code the logic indicated by dashed lines.)

Figure 111

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..

           GO TO HOUR-RATE WEEK-RATE
               MONTH-RATE
               DEPENDING ON PAYCODE.
```

Rule 1 in Figure 111 states that the identifier in the GO TO statement with the DEPENDING ON option must represent a positive or unsigned integer. If the integer is zero or is larger than the number of paragraph names listed, the GO TO statement is ignored. Match the effects of the statement above with the assumed values of PAYCODE.

1) 9

2) 3

3) 1

4) 0

a. Control is transferred to HOUR-RATE.

b. Control is transferred to WEEK-RATE.

c. Control is transferred to MONTH-RATE.

d. Control is transferred to PAYCODE.

e. Control passes to the statement following the GO TO statement.

* * *

1) e
2) c
3) a
4) e

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 INDICATOR IS EQUAL TO 1
               GO TO UPDATE-ROUTINE.
           IF INDICATOR IS EQUAL TO 2
               GO TO ADD-ROUTINE.
           IF INDICATOR IS EQUAL TO 3
               GO TO DELETE-ROUTINE.
```

The example above shows IF statements can be used to transfer control depending on the value of a variable. Write a GO TO statement with the DEPENDING ON option that will have the same effect as the three IF 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..

           GO TO UPDATE-ROUTINE
               ADD-ROUTINE DELETE-ROUTINE
               DEPENDING ON INDICATOR.
```

----

14. Which of the following statements would cause a transfer of control to paragraph MATH when the value of MAJOR is 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 MAJOR IS EQUAL TO 2
               GO TO MATH.
```

    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 MATH DEPENDING ON
               MAJOR IS EQUAL TO 2.
```

    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..

           GO TO ENGLISH MATH ANTHRO
               DEPENDING ON MAJOR.
```

*   *   *

a,c

----

15. Refer to Figure 111 and write a statement that will transfer control to paragraph TECHNICAL if the value of TYPE-OF-STUDENT is 4, to paragraph NONDEGREE if the value of TYPE-OF-STUDENT is 3, to UNDERGRADUATE if the value is 2, and to GRADUATE if the value is 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..

           GO TO GRADUATE UNDERGRADUATE
               NONDEGREE TECHNICAL
               DEPENDING ON TYPE-OF-STUDENT.
```

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..

       03 PAYMENT-HISTORY PIC 9.
          88 BAD VALUE 1.
          88 POOR VALUE 2.
          88 SLOW VALUE 3.
          88 AVERAGE VALUE 4.
          88 GOOD VALUE 5.
          88 EXCELLENT VALUE 6.
          88 NONE VALUE 7.
```

Write a statement that will cause transfer of control to one of the paragraphs BAD-ROUTINE through NONE-ROUTINE based on the respective values of PAYMENT-HISTORY.

* * *

```
0   0    1    1    2    2    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 BAD-ROUTINE POOR-ROUTINE
               SLOW-ROUTINE AVERAGE-ROUTINE
               GOOD-ROUTINE EXCELLENT-ROUTINE
               NONE-ROUTINE
               DEPENDING ON PAYMENT-HISTORY.
```

(The condition names are not needed when a GO TO statement with the DEPENDING ON option is used.)

A simple GO TO statement (one without the DEPENDING ON option) may also be used to transfer control to different points in a program if the program alters the paragraph name specified in the GO TO statement. The ALTER statement may be used to alter a paragraph name in a GO TO statement.

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..

           FIX.
               GO TO IN-STATE.
```

ALTER Statement

Format

ALTER paragraph-name-1 TO [PROCEED TO] paragraph-name-2

Explanation — The ALTER statement is used to change a name specified in the GO TO statement so that control will be transferred to a different point in the program.

Rules

1. Paragraph-name-1 must be the name of a paragraph that contains only one statement; a GO TO statement without the DEPENDING CN Option.

2. Paragraph-name-2 must be the name of a paragraph to which control is to be transferred by the GO TO statement in paragraph-1 during future executions of that GO TO statement.

Figure 112

Read the explanation and rules in Figure 112. Which of the ALTER statements below could alter the GO TO statement above so that execution of paragraph FIX would cause control to be transferred to paragraph OUT-STATE?

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..

        ALTER FIX TO PROCEED
            TO OUT-STATE.
```

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..

        ALTER IN-STATE TO PROCEED
            TO OUT-STATE.
```

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..

        ALTER FIX TO PROCEED
            TO IN-STATE.
```

* * *

a

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..

    SWITCH-PARAGRAPH.
        GO TO PENALTY.
    ALTER-PARAGRAPH.
```

In the segment above the GO TO statement is the only statement in SWITCH-PARAGRAPH. Write a statement that will cause execution of SWITCH-PARAGRAPH to transfer control 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..

        ALTER SWITCH-PARAGRAPH TO
            PROCEED TO DISCOUNT.
```

----

19. Which of the following paragraphs could be changed by an ALTER 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..

       PARA1.
           GO TO Q R S
               DEPENDING ON LETTER.
```

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..

       PARA2.
           GO TO Q.
```

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..

       PARA3.
           WRITE ERROR-RECORD.
           GO TO Q.
```

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..

       PARA4.
           ALTER Q TO PROCEED TO S.
```

* * *

b
(In a the GO TO statement contains the DEPENDING ON option; in c the GO TO statement is not the only statement in the paragraph; in d there is no GO TO statement to be altered.)

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..

           GO TO.
```

A GO TO statement that will be altered before it is executed may have the form shown above. A GO TO statement that specifies no paragraph name:

a. must be the only statement in a paragraph.

b. may be executed any time before it is altered.

* * *

a

----

21. Refer to Figure 112 and decide which of the following versions of paragraph END-ROUTINE could be specified as paragraph-name-1 in an ALTER 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..

       END-ROUTINE.
           ADD 1 TO COUNTER.
           GO TO.
```

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..

       END-ROUTINE.
           ADD 1 TO COUNTER.
           GO TO GET-ANOTHER-CARD.
```

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..

       END-ROUTINE.
           GO TO.
```

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..

       END-ROUTINE.
           GO TO GET-ANOTHER-CARD.
```

*　　*　　*

c,d

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..

       PARAGRAPH-1.
           GO TO BYPASS-PARAGRAPH.
               .
               .
               .
       BYPASS-PARAGRAPH.
               .
               .
               .
           ALTER PARAGRAPH-1
               TO PROCEED TO PARAGRAPH-2
               .
               .
               .
       PARAGRAPH-2.
               .
               .
               .
```

The segment above contains an ALTER statement. Which of the following describes the effect of execution of the segment above?

a. After the ALTER statement is executed, execution of PARAGRAPH-1 will cause control to be transferred to BYPASS-PARAGRAPH.

b. Before the ALTER statement is executed, execution of PARAGRAPH-1 will cause control to be transferred to PARAGRAPH-2.

* * *

Neither (The opposite is true.)

23.

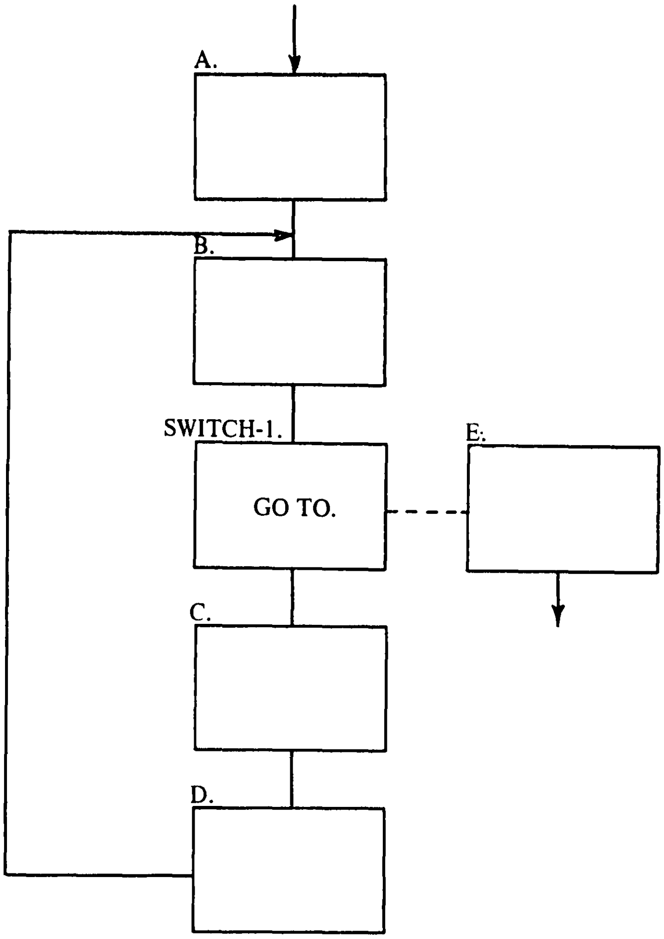


Figure 113

The paragraphs in the flow chart segment above are to be executed in the sequence A, B, SWITCH-1, C, D, B, SWITCH-1, E. The dashed line on the flow chart indicates the direction of flow of control the second time paragraph SWITCH-1 is executed. To specify a paragraph name for the GO TO statement in paragraph SWITCH-1 you must insert an ALTER statement into paragraph:

a. A to specify transfer of control to paragraph C when the GO TO is first executed.

b. SWITCH-1 before the GO TO statement to specify transfer of control each time the GO TO statement is executed.

c. SWITCH-1 after the GO TO statement to specify transfer of control to paragraph E when the GO TO statement is executed the second time.

* * *

a

24.

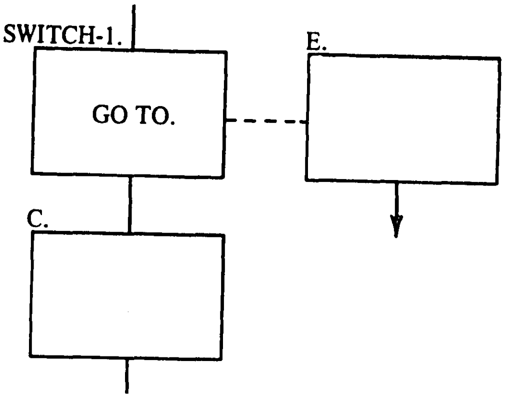


Figure 114

Write the statement to be inserted into paragraph A of the flowchart shown in Figure 113 to provide for transter of control in SWITCH-1 as indicated by the solid line in the flow chart 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..

       ALTER SWITCH-1 TO PROCEED TO C.

------------------------------------------------------------------------
```

25.

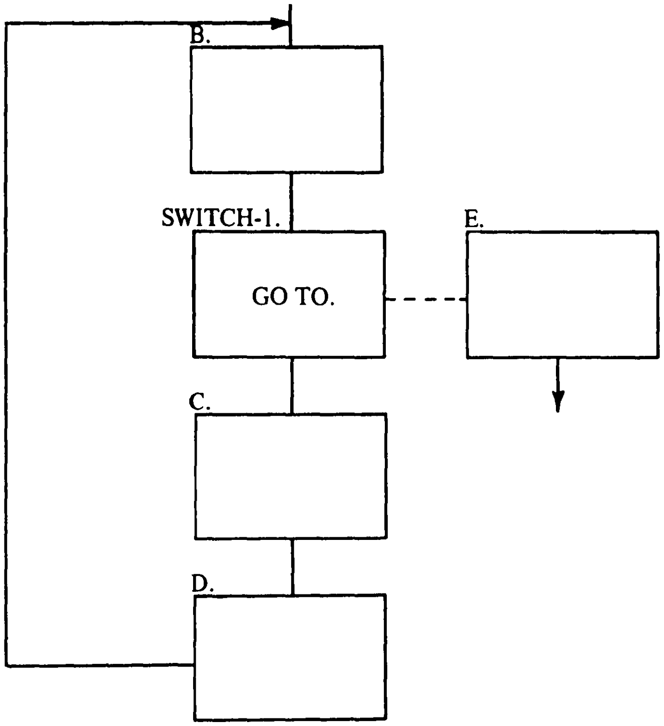


Figure 115

Write the last two statements for paragraph D in the flow chart above to alter SWITCH-1 paragraph to provide for transfer of control in SWITCH-1 as indicated by the dashed 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..

        ALTER SWITCH-1 TO PROCEED TO E.
        GO TO B.
```

----

26. University Tog Shops, Inc. maintains a file containing information on all of its member shops. The file needs updating every year. An input card file contains data on shops that have closed, identified by a 1 in column 80 of a card, shops that have opened, identified by a 2, shops that have changed managers, identified by a 3, and general income information, identified by a 4. Different routines are to be executed for each identification number. To solve this problem, you would use:

    a. an ALTER statement as soon as a card is read.

    b. a GO TO statement with the DEPENDING ON option as soon as a card is read.

* * *

b

---

27. In a program you are writing, you are to transfer control to a message routine after the main sequence is executed a second time. You could accomplish this by inserting:

    a. an ALTER statement to be executed at the end of the first execution of the main sequence.

    b. a GO TO statement as the only statement in a paragraph.

    c. a GO TO statement with the DEPENDING ON option to be executed after the first execution of the main sequence.

* * *

a,b
(Both a and b would be required.)

SUMMARY

In this lesson you have learned to use the DEPENDING ON option of the GO TO statement and to use the ALTER statement in conjunction with a GO TO statement that may or may not specify a paragraph name.

END OF LESSON 24

LESSON 25

LESSON 25 - DATA FORMATS

INTRODUCTION

In this lesson, some techniques for improving the efficiency of a program will be presented. You will learn some situations that cause extra instructions to be generated by the compiler and how to avoid these situations by defining data in specific ways in data description entries.

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

USAGE clause
DISPLAY option of the USAGE clause
COMP (COMPUTATIONAL) option of the USAGE clause

Programming techniques that you will learn to use in this lesson will involve:

Decimal point alignment
Types of moves

This lesson will require approximately three quarters of an hour.

USAGE IS DISPLAY
DISPLAY statement

The reserved word DISPLAY can be specified either in a USAGE clause or in a DISPLAY statement. In the initial sequence of frames you will learn to use the word DISPLAY in the USAGE clause.

---

1. Thus far in the course, every data item has been represented in core storage in the same way. The USAGE clause may be used to specify how a data item is to be stored in core storage. The USAGE clause:

   a. determines the manner in which a data item is represented in core storage.

   b. may be omitted from a program.

* * *

Both

---

2. The DISPLAY option of the USAGE clause has the same effect as the omission of the USAGE clause. The DISPLAY option of the USAGE clause could be specified for:

   a. alphabetic data.

   b. alphanumeric data.

   c. numeric data.

   d. edited data.

* * *

All of these
(USAGE IS DISPLAY is seldom specified in a program, since omission of the clause has the same effect.)

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  NAME PIC X(20) USAGE IS DISPLAY
```

The entry above indicates that the USAGE clause is specified in the .... Division.

* * *

Data

---

4. USAGE IS DISPLAY

The USAGE clause shown above specifies that each character position in the picture of its associated variable will require one word of storage. The DISPLAY option of the USAGE clause:

a. specifies that a variable with picture 99V99 will require 5 words of storage.

b. has the same effect as the omission of the USAGE clause.

* * *

b
(V does not represent a character position.)

The term DISPLAY variable refers to a variable that has USAGE IS DISPLAY specified in its data description entry, or has no USAGE clause at all. The term displayed value refers to a value of a variable that was specified in a DISPLAY statement. It is important to note, however, that each variable for which the USAGE clause has not been specified is frequently referred to as a DISPLAY variable. A numeric variable for which the USAGE clause has not been specified may be referred to as a numeric DISPLAY variable.

---

5. Values of DISPLAY variables are stored one character per word. Values of numeric DISPLAY variables, called external decimal values, contain a representation of an algebraic sign in the left half of the rightmost position of a word. Examples of the storage of values of DISPLAY variables are shown in the table below. Z = Zone, equivalent to hexadecimal F.

PICTURE	Value	Storage in Hexadecimal
X(6)	PT-109	D7E360F1F0F9 (EBCDIC CODE)
9(5)	11995	Z1Z1Z9Z9F5 F represents "unsigned"
S999	+720	Z7Z2C0 C represents "+ sign"
S9V99	-125	Z1Z2D5 D represents "- sign"

Referring to the table above, show how the values in the following table will be stored if either USAGE IS DISPLAY is specified or the USAGE clause is omitted.

PICTURE	Value	Storage
X(7)	2551112	
9(7)	2551112	
S9(4)	+1969	
S9V99	-198	

* * *

Storage

```
F2 F5 F5 F1 F1 F1 F2
Z2 Z5 Z5 Z1 Z1 Z1 Z2
Z1 Z9 Z6 C9
Z1 Z9 D8
```

(An unsigned representation, F, is placed in the rightmost word or numeric DISPLAY variable which contains no S in its picture.)

---

6. Numeric values (variables whose picture contain only 9's, and an S or V if necessary) may be stored in a more compact way if COMPUTATIONAL is specified in the USAGE clause. (COMPUTATIONAL is usually written COMP since the form is shorter.) Values of COMP are stored as binary data.

   A binary data item has a decimal equivalent that consists of numeric characters 0 through 9 plus a sign. It occupies one word, two words, or four words, corresponding to decimal lengths of 1 through 4 digits, 5 through 9 digits, and 10 through 18 digits respectively. The leftmost bit of the storage area is the operational sign.

   USAGE IS COMPUTATIONAL must be specified for binary data.

PICTURE	VALUE	STORAGE			
		sign	WORD		
S9999	+1234	0000	0100	1101	0010
S9999	-1234	1111	1011	0010	1110
9999	-1234	0000	0100	1101	0010

   The sign position of a binary field, internally a '1' in the leftmost position, means the number is negative. A '0' in the leftmost position of a binary field means the number is positive.

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..

       WORKING-STORAGE SECTION.
       77  PRICE PIC S99V99 USAGE IS COMP.
       77  QUANTITY PIC 999 USAGE IS COMP.
       77  EXTEND PIC X9(5)V99 USAGE IS COMP.
```

Refer to the data description entries above, and match each phrase below with the variables that it describes.

1) PRICE
2) QUANTITY
3) EXTEND

a. Binary.
b. DISPLAY variables.
c. Requires one word of storage.
d. Requires two words of storage.
e. Numeric field.

*   *   *

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

---

8. USAGE IS COMP.

COMP may be specified only for numeric variables. Which of the following data description entries could be correct?

a. 02 ESTATE PIC 999V99
USAGE IS COMP.

b. ESTATE PIC 999V99
USAGE IS DISPLAY.

c. EMPLOYER PIC X(22)
USAGE IS COMP.

d. EMPLOYER PIC X(22)
USAGE IS DISPLAY.

*   *   *

a,b,d
(The picture of a COMPUTATIONAL item may contain only 9's, the operational sign S, the implied decimal point V, and one or more P's.)

Internal Representation of Numeric Items

Item	Value	Description	Internal Representation
External Decimal	-1234	DISPLAY PICTURE 9999	00Z1 00Z2 00Z3 00F4 word
		DISPLAY PICTURE S9999	00Z1 00Z2 00Z3 00D4 word Note: Internally the 00D4 which represents -4 is the same bit configuration as the EBCDIC character M.
Binary	+1234	COMPUTATIONAL PICTURE s9999	0000 0100 1101 0010 s word
External Decimal	+1234	DISPLAY PICTURE S9999	00Z1 00Z2 00Z3 00C4 word
		DISPLAY PICTURE 9999	00Z1 00Z2 00Z3 00F4 word
Binary	-1234	COMPUTATIONAL PICTURE S9999	1111 1011 0010 1110 word
		COMPUTATIONAL PICTURE 9999	0000 0100 1101 0010 word

Z = zone, equivalent to hexadecimal F (bit configuration is 1111)
Hexadecimal F = nonprinting plus sign
Hexadecimal C = internal equivalent of the plus sign
(bit configuration is 1100)
Hexadecimal D = internal equivalent of the minus sign
(bit configuration is 1101)
S = the sign position of a numeric field; internally a '1' in position S means the number is negative, whereas a '0' in position S means the number is positive

Figure 116

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..

        WORKING-STORAGE SECTION.
        77  PRICE PIC S99V99 USAGE IS COMP.
        77  QUANTITY PIC 999 USAGE IS COMP.
        77  EXTEND PIC S9(5)V99.
```

Figure 116 explains the internal representations that are necessary in arithmetic operations. Refer to Figure 116 and the working storage variables above to select the effects of execution of the multiply 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..

            MULTIPLY PRICE BY QUANTITY
            GIVING EXTEND.
```

a. PRICE and QUANTITY are converted to COMP.

b. the result is converted to COMP before being placed in EXTEND.

* * *

Neither
(PRICE and QUANTITY are not converted. The COMP result is converted to DISPLAY before being placed in EXTEND.)

----

10. DIVIDE EXTEND INTO QUANTITY.

When the DIVIDE statement above is executed:

a. EXTEND will be converted to COMP.

b. QUANTITY will be converted to DISPLAY.

c. the result will be placed in the result field without conversion.

* * *

a,c
(Since both the result and the result field, QUANTITY, are COMP there is no conversion.)

The next sequence of frames will explain the effects of arithmetic operations, including conversions required for numeric variables, output requirements for result fields, alignment of decimal point, and effects of the sign in arithmetic operations.

----

11. In writing an efficient program a programmer should specify USAGE clauses to provide for as few conversions as possible. He should specify COMP for variables that will be:

    a. used in arithmetic operations.

    b. specified in arithmetic statements.

    c. specified in COMPUTE statements.

* * *

Any of these

----

12. ALIGNMENT of decimal points is another consideration in arithmetic operations. when you do an addition or subtraction problem by hand, you are careful to align the decimal points. The compiler must do this also, if the pictures of the variables do not provide for alignment. Any extra tasks to be performed by the compiler, such as conversion or alignment, decrease the efficiency of a 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..

           ADD TAX TO SUBTOTAL.
```

Which of the three data descriptions for TAX and SUBTOTAL below would result in the most efficient execution of the statement above?

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..

       77  TAX PIC S99V99
           USAGE IS COMP.
       77  SUBTOTAL PIC S9999
           USAGE IS COMP.
```

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..

       77  TAX PIC S99V999
           USAGE IS COMP.
       77  SUBTOTAL PIC S999V99.
           USAGE IS COMP.
```

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..

       77  TAX PIC S99V99.
           USAGE IS COMP.
       77  SUBTOTAL PIC S999V99.
           USAGE IS COMP.
```

* * *

c

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 DISCOUNT FROM BALANCE.
```

BALANCE is a variable that can contain values up to but not including 100000. DISCOUNT is computed in the program to be 10 percent of BALANCE. Write level 77 data description entries which will result in efficient execution of the above 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..

       77  BALANCE PIC S999V99 USAGE IS COMP.
       77  DISCOUNT PIC S99V99 USAGE IS COMP.
```

(You should have specified the same number of decimal places for each variable.)

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..

       77  BALANCE PIC S999V99.
           USAGE IS COMP.
       77  DISCOUNT PIC S99V99.
           USAGE IS COMP.


           SUBTRACT DISCOUNT FROM BALANCE
               GIVING NEW-BALANCE.
```

The value of NEW-BALANCE after execution of the statement above is to be added to a final total. The appropriate picture for NEW-BALANCE for efficient execution of the statement above would be:

a. S999V999

b. S999V9

*   *   *

Neither
(S999V99)

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..

       77  DISCOUNT-RATE PIC SV99.
           USAGE IS COMP.
       77  BALANCE PIC S999V99.
           USAGE IS COMP.


           MULTIPLY DISCOUNT-RATE
               BY BALANCE GIVING DISCOUNT.
```

In order to avoid any truncation or padding with zeros in the value transmitted to DISCOUNT as a result of the above statement, DISCOUNT would be given a picture such as:

a. S999V9999

b. S999V99

c. S999V99999

* * *

a
(b would cause truncation of two digits. This is permissible, of course, but a programmer should be aware of truncation whenever it occurs. In this example, if he needs only two decimal places for DISCOUNT, he may prefer to describe it as S999V99 and specifiy the ROUNDED option in his arithmetic statement.)

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..

       77  DISCOUNT-RATE PIC SV99.
           USAGE IS COMP.
       77  BALANCE PIC S999V99.
           USAGE IS COMP.
```

When a numeric literal is used in an arithmetic statement, it should be expressed in a way that does not require alignment by the compiler. Which statement expresses a numeric literal for efficient execution?

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 20 TO 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..

           ADD 20.00 TO BALANCE.
```

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 20V00 TO BALANCE.
```

*   *   *

b

----

17. Whenever a value (signed or unsigned) is stored in a numeric variable with an unsigned picture, the compiler inserts a symbol meaning "absolute value" into the word that would ordinarily hold the sign representation. Although the value will be treated as positive, execution time for the object program will be increased if the unsigned value is later used in a computation. For this reason an S should be included in the picture of:

a. every numeric variable.

b. any variable that will be used in an arithmetic statement.

c. every variable.

* * *

b

Specifying the correct usage and an S in the picture of any variable that will be used in an arithmetic statement contributes to the efficiency of program execution. Another means of increasing efficiency is to avoid repeating the same conversion or operation on the same variables or literals.

18.

Circumference of cylindrical tank	2PiR	
Area of side of cylindrical tank	2PiRH	R = radius
Area of top or bottom of tank	PiR^2	H = height
Volume of tank	PiR^{2}H	

Figure 117

A programmer for a manufacturing firm might find it necessary to use of the above formulas for computations relating to a single tank radius. Each formula in the table contains the factor R. Instead of including 3.14159 * RADIUS in each COMPUTE statement, the programmer would have a more efficient program if 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..

        COMPUTE PI-R  = 3.14159 * RADIUS
```

were executed, and the variable PI-R were used in all other COMPUTE statements.

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 AREA-OF-SIDE =
            CIRCUMFERENCE * HEIGHT
```

were executed to compute the second formula above after computing the first formula.

*   *   *

Both

----

19. Factors that must be considered in writing an efficient program are:

    a. the type of storage of the nonnumeric variables.

    b. conversion of values of variables used in arithmetic operations.

    c. decimal point alignment of literals or values of variables used in arithmetic operations.

    d. the presence or absence of a sign in values of numeric variables.

    e. repetition of the same operation involving the same literals or variables.

    f. repetition of a conversion involving the same variable.

*   *   *

b,c,d,e,f

----

20. Match the actions a programmer could take with each factor in writing an efficient program.

1) Decimal point alignment of literals or values of variables used in arithmetic operations

2) Conversions of values of variables used in arithmetic operations

3) The presence of a sign in values of numeric variables used in arithmetic operations

a. Specify USAGE clauses for variables to provide for as few conversions as possible in arithmetic operations

b. Specify pictures for variables to provide for decimal point alignement

c. Express numeric literals in a way that does not require alignment by the compiler

d. Specify an S in the pictures of variables

* * *

1) b,c
2) a
3) d

---

21. Match the actions a programmer could take with each factor in writing an efficient program.

1) Repetition of the same operations involving the same variables or literals

2) Repetition of a conversion involving the same variable

a. Provide for a conversion common to several steps in one step and then use the result in subsequent steps

b. Provide for an operation common to several computations in one computation and then use the result in all subsequent computations

c. Specify an S in the picture of variables

* * *

1) b
2) a

The topics presented so far in this lesson have dealt with increasing efficiency in a program by proper use of numeric variables. The next sequence of frames will show several ways of printing numeric variables after any computations are completed.

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..

       77  A PIC S999.
       77  B PIC S999 USAGE IS COMP.
       77  C PIC S99V99.
       77  D PIC S99V99 USAGE IS COMP.
       77  E PIC X(6).
       77  F PIC $99.99.
       77  G PIC XXX.
```

The form of the receiving variable determines the type of elementary move. Figure 118 shows the types of moves. Refer to the entries above and Figure 118 to match the types of moves in the following column with the moves specified in the moves statements.

Specific Conversions in Certain Moves

MOVE sending-variable TO receiving-variable			
	Alphanumeric move	Numeric move	Edit move
Receiving variable Sending variable	Alphanumeric	External Decimal	Edited
Alphanumeric	No conversion	Whole numbers only	Whole numbers only
External Decimal	Whole numbers only, no conversion	No conversion	Value is edited
Edited	No conversion	Invalid move	Invalid move

How to use the chart:

Find the data type of the sending variable in the leftmost column. Then find the data type of the receiving variable in the row across the top of the chart. Extend imaginary lines into the chart from the two data types. These two lines will intersect in a block that tells what conversion takes place.

Figure 118

```
       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..

           MOVE A TO 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..

           MOVE B TO G.

       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 G TO 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 C TO F.
```

a. Alphanumeric move

b. Numeric move

c. Edit move

* * *

1) b
2) a
3) b
4) c

---

23. When a numeric variable is printed or displayed, the rightmost character printed may not be the rightmost digit of the value of the variable. Because the rightmost word contains a sign representation in addition to the digit, the rightmost character printed may be a digit punch with an overpunch. For this reason, a numeric variable is usually moved to an alphanumeric or edited variable before it is printed or displayed. Figure 118 shows effects and limitations of moves involving numeric variables. A numeric variable with picture S999 could be moved to:

    a. an alphanumeric variable

    b. an edited variable

* * *

Either (S999 represents an integer, or whole number.)

---

24. Assume that a value may be moved more than once to achieve a final converted format not achievable in a single move. According to Figure 118, an external decimal value can be moved to an:

   a. alphanumeric variable if it is a whole number.

   b. edited variable, and then to an alphanumeric variable regardless of the location of the decimal point.

* * *

Either

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  PRICE PIC S99V99.
                USAGE IS COMP.
        77  QUANTITY PIC 999.
                USAGE IS COMP.
        77  EXTEND PIC S9(5)V99.


            MULTIPLY PRICE BY QUANTITY
                GIVING EXTEND.
```

If you wanted to print the result of the statement above, you would first:

   a. move EXTEND to an alphanumeric variable.

   b. move EXTFND to an edited variable.

   c. change the PICTURE clause for EXTEND from external decimal to a numeric edited item.

* * *

b,c

---

26. According to Figure 118, a numeric variable can always be moved to an edited variable. The edited variable can always be moved to an alphanumeric variable. One way to print an edited result of an arithmetic operation would be to:

   a. define the result field as an edited variable and then move it to an alphanumeric variable.

   b. define the result field as a decimal variable and then move it to an edited field.

* * *

Either

Receiving Field Source Field	GR	AL	AN	ED	BI	NE	ANE
Group (GR)	Y	Y	Y	Y[1]	Y[1]	Y[1]	Y[1]
Alphabetic	Y	Y	Y	N	N	N	Y
Alphanumeric (AN)	Y	Y	Y	Y[4]	Y[4]	Y[4]	Y
External Decimal (ED)	Y[1]	N	Y[2]	Y	Y	Y	Y[2]
Binary (BI)	Y[1]	N	Y[2]	Y	Y	Y	Y[2]
Numeric Edited (NE)	Y	N	Y	N	N	N	Y
Alphanumeric Edited (ANE)	Y	Y	Y	N	N	N	Y
ZEROS (numeric or alphanumeric)	Y	N	Y	Y[3]	Y[3]	Y[3]	Y
SPACES (AN)	Y	Y	Y	N	N	N	Y
HIGH-VALUE, LOW-VALUE, QUOTES	Y	N	Y	N	N	N	Y
ALL "character"	Y	Y	Y	Y[5]	Y[5]	Y[5]	Y
Numeric Literal	Y[2]	N	Y[2]	Y	Y	Y	Y[2]
Nonnumeric Literal	Y	Y	Y	Y[5]	Y[5]	Y[5]	Y

[1] Move without conversion (like AN to AN)
[2] Only if the decimal point is at the right of the least significant digit
[3] Numeric move
[4] The alphanumeric field is treated as an ED (integer) field
[5] The literal must consist only of numeric characters

Figure 119

27. Figure 119 shows the types of valid moves for various source variables and receiving variables. The source and receiving variables shown are group and elementary variables. according to Figure 119 the type of move from one group variable to another group variable:

a. depends on the form of the receiving variable

b. is always numeric

c. is always an alphanumeric move

* * *

c

28. In a preceding frame, it was stated that the type of elementary move was determined by the form of the receiving variable. You can see from Figure 119 that this rule:

a. applies to all moves.

b. applies only to elementary moves.

c. does not always apply to moves in which a group item is involved.

* * *

b,c

Effects of Types of Moves

Type of move	Receiving item	Compiler action during move	Alignment	Padding if necessary	Truncation if necessary
Alphanumeric	Group	none	at left of value	on right with spaces	on right
	Alphabetic or alphanumeric	any necessary conversion	at left of value	on right with spaces	on right
Numeric	External decimal or packed decimal	any necessary conversion	at decimal point	on left and right with zeros	on left and right
Edit	Edited	editing and any necessary conversion	at decimal point	on left and right with zeros (unless suppressed)	on left and right

Figure 120

29. Figure 120 describes the effects of the different types of moves. For alphanumeric moves it describes the effects for both group and elementary variables. 'Select the statements that are correct based on Figure 120.

a. No conversion takes place when the receiving item is a group variable.

b. Padding with spaces takes place only in an alphanumeric move.

c. If an external decimal value were moved to a longer alphanumeric variable, it would be aligned at the left of the receiving item and padded with spaces on the right.

* * *

All of these

SUMMARY

In the preceding part of this lesson, you have learned some ways to provide for efficiency in a program. The ways of increasing efficiency in arithmetic operations also apply to comparisons of numeric operands, which are compared algebraically and thus require arithmetic operations to be performed. Both numeric operands in a relational condition will be converted to decimal if they are not stored in that form. The decimal points will be aligned and the values will be padded with zeros if the programmer has not provided for this.

END OF LESSON 25

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 26

LESSON 26 - EDIT CHARACTERS

INTRODUCTION

A number of additional picture characters that will be useful to you in your career as a programmer will be presented in this lesson. You wil study these edit characters for use in the PICTURE clause:

Z
*
B
0
-
+
DB
CR

You will also learn the programming techniques that involve taking advantage of valid moves.

This lesson will require approximately three quarters of an hour.

1. Figure 121 shows eight additional picture characters. The picture characters Z and * represent digit positions and specify that .... will be suppressed and replaced by .... or ....

Additional Edit Characters

Picture char.	Data Type	Specification	Additional explanation
Z	numeric edited	A leading zero in the associated position will be suppressed and replaced with a blank	No Z may be to the right of a 9.
*	numeric edited	A leading zero in the associated position will be suppressend and replaced with an asterisk	No * may be to the right of a 9.
B	numeric edited	The associated position in the value will contain a blank, $, or *.	When a B included in a string of $'s *'s, or Z's, a a digit,$, or * could appear in its position.
0	numeric edited	The associated position in the value will contain a zero, blank, $, or *.	When a 0 is included in a string of $'s,*'s or Z's, a digit,$ or * could appear in its position.
CR DB	numeric edited	A negative indicator will be inserted into the value when the value is negative; two blanks will appear otherwise.	These symbols must be at right end of picture.
-	numeric edited	A minus sign is inserted into the value when the value is negative; one blank will appear otherwise	A - may be at either end of picture; may be "floated" with the same rules as $.
+	numeric edited	The appropriate sign (+ or -) will be inserted into the value.	A + may be at either end of the either end of the "floated" with the same rules as $.

Figure 121

* * *

leading zeros
blanks
asterisks

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 AMOUNT PIC ZZ9
```

The Z picture character in the picture of a variable specifies that blanks are to replace leading zeros in positions corresponding to Z's when a value is stored in the variable. If the value 88 is moved to AMOUNT, defined as above, the value would be ƀ88. If the value 8 were moved to AMOUNT, the value would be ƀƀ8. If the value 0 were moved to AMOUNT, the value would be:

a. ƀƀƀ (all blanks)

b. 000

c. ƀƀ0

* * *

c (A 9 always indicates that the digit will appear.)

---

3. When the value 4595 is moved to a variable with the picture ZZZ.99, the value of the variable will be ....

* * *

ƀ45.95

---

4. Asterisks in a picture specification have the effect of replacing suppressed zeros with asterisks. If the value 9000 were moved to a variable defined with the picture ***.99, the value of the variable would be *90.00. If the value 900 were moved to the same variable, the value of the variable would be ....

* * *

**9.00

---

5. The * and Z picture characters may not appear in the same PICTURE clause. In addition, neither an * nor a Z may appear to the right of a 9. Which of the following is a valid picture?

a. ***ZZ

b. ZZ.99

c. **.ZZ

d. Z9.99

e. Z9.ZZ

f. 99.**

g. **.**

* * *

b,d,g

6.

Picture	Source	Result
***.99	000/04	***.04
***.**	000/04	***.04
***.99	000/00	***.00
***.**	000/00	***.**

As shown in the table above, the effect of placing asterisks in all positions in a picture when the value of the variable is not zero is that leading zeros preceding the point are suppressed and replaced by asterisks. When the value of the variable is zero, all positions are replaced by asterisks except the decimal point position. Give the result of moving each of the following values to a variable with picture ****.**.

1) 98700

2) 00005

3) 000000

* * *

1) *987.00
2) ****.05
3) ****.**

7.

Picture	Source	Result
ZZZ.99	000/04	ƀƀƀ.04
ZZZ.ZZ	000/04	ƀƀƀ.04
ZZZ.99	000/00	ƀƀƀ.00
ZZZ.ZZ	000/00	ƀƀƀƀƀƀ

As shown in the table above, the effect of placing Z's in the entire picture is the same as the effect of placing $'s in the entire picture; that is:

a. all leading zeros are suppressed, even if they appear to the right of the decimal point.

b. if the value of the variable is zero, the entire item will be suppressed.

*   *   *

b

---

8. When the value of a variable that has a picture ZZZZ.ZZ is zero, ... will be printed. When the value of a variable that has a picture ****.** is zero, ... will be printed.

*   *   *

blanks (nothing)
asterisks (*'s) and the decimal point

---

9. You know that a comma insertion character within a floating string of dollar signs may be suppressed when a dollar sign is printed to the right of the comma, or printed as a dollar sign if the first printed digit is immediately to the right of the comma position. This rule for comma insertion is true also for strings of Z's or asterisks.

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

           03  AMOUNT PIC ZZZ,ZZZ.
```

Write the value of AMOUNT after the following values are moved to it.

1) 920617

2) 362

*   *   *

1) 920,617
2) ƀƀƀƀ362

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..

          03  AMOUNT PIC ***,***.
          03  BALANCE PIC **,***.**.
```

If the value 20617 were moved to AMOUNT, its value would be 20,617. The value 617 moved to AMOUNT would cause its value to be ****617. If no digit appears to the left of the comma position, an asterisk instead of a comma is printed in the common position. If the value 2198 were moved to BALANCE and printed:

a. the printed value would be **,*21.98.

b. four asterisks would appear at the left of the value.

* * *

b (The printed value would be ****21.98.)

---

11. Write pictures for the variables described below:

1) DEBT is to contain up to $50,000.00; the dollar sign is to be printed immediately preceding the first nonzero digit. If the value is zero, $.00 is to be printed.

2) OWES is to contain amounts up to 50,000.00; leading zeros are to be replaced by blanks. If the value is zero, nothing is to be printed.

3) CHECK is to contain amounts up to 50,000.00; leading zeros are to be replaced by asterisks. If the value is zero, eight asterisks and the decimal point are to be printed.

* * *

1) $$$,$$$.99
2) ZZ,ZZZ.ZZ
3) **,***.** (If the value is zero, the comma is replaced by an asterisk.)

---

12. According to Figure 121, each B or 0 represents one blank or zero to be inserted into a value. The picture 99,000 would cause:

a. three zeros to be inserted into a value.

b. one comma to be inserted into a value.

c. six characters to be printed.

* * *

All of these

---

13. The picture 9,000 would cause the value 5 to be printed as:

a. 0,005

b. 5,000

c. 5000

* * *

b

----

14. The picture 999B9999 would cause the value 2551112 to be printed as ....

* * *

255ƀ1112

----

15. Write a picture that would cause the value 376401495 to be printed as 376ƀ40ƀ1495.

* * *

999B99B9999

----

16. B and 0 picture characters can be included in strings of Z's, *'s, or $'s with the same effect as a comma in such a string. That is, a B or 0 in a picture specification such as ZZZ0ZZ, ***B**, or $$$B$$.99 might be printed as:

a. a digit.

b. an * or $.

* * *

b

----

17. Match the values below with the picture that would cause the value to be printed in that form.

1) ZZZBZZZ	a. ƀ230456
2) ***0***	b. ƀ23ƀ456
	c. *23ƀ456
	d. *230456
	e. 0230456

* * *

1) b
2) d

----

18. It is frequently useful to identify certain printed data items as DB or CR or as + or -. This is done by specifying editing sign control symbols. Refer to Figure 121 and find the Editing Sign Control Symbols. This table gives the result of placing any of the sign control symbols at the right end of the picture of a variable for both positive or zero and negative values of the variable. Refer to the table and give the result of moving each of the following values to a variable with the specified picture.

	Value	Picture of Variable	Printed Result
1)	234567	Z,ZZZ.99CR	
2)	-34567	*,***.99DB	
3)	+34567	Z,ZZZ.99-	
4)	-4280	Z,ZZZ.99+	
5)	+917	*,***.99+	

* * *

Printed Result

1) 2,345.67ƀƀ
2) **345.67DB
3) ƀƀ345.67ƀ
4) ƀƀƀ42.80-
5) ****9.17+

---

19. Write pictures that will produce the following results when the given source values are moved to variables with the pictures.

	Result	Source Value	Picture
1)	ƀ1200-	-01298	
2)	ƀ1200ƀ	+01298	
3)	ƀ1200CR	-01298	

* * *

Picture

1) Z9900- or Z9900+
2) Z9900-
3) Z9900CR
(The 9's in these pictures could be replaced by Z's.)

---

20. The sign characters + and may occur at the left end of a picture specification if desired. These characters may also be used as floating characters just as the dollar sign is used. When a value of 0005 is moved to a variable with the picture +++.++, the printed value will be:

a. +++.05

b. ƀƀ+.05

c. ƀƀƀ.+5

* * *

b

---

21. The maximum value that could be moved to a variable with a picture ---.-- would be:

a. 999.99

b. -999.99

c. 99.99

* * *

c
(Remember that the leftmost character in floating string ($, +, or -) does not represent a digit position. a would result in 99.99 while b would result in -99.99.)

---

22. Write a picture specification for a variable that will have a value of up to seven digits with two digits to the right of the point. Leading zeros in the first four places are to be replaced by asterisks. If the value of the variable is negative, the letters CR are to be printed one space after the rightmost digit.

* * *

****9.99BCR
(The leftmost zero-suppression character (* or Z) does represent a digit position.)

---

23. Match the printed values below with the appropriate pictures.

1) $$,$$9.99-
2) $$,$$$.$$-
3) ZZZ.Z
4) ***.*+
5) *99.9-
6) 9999.9

a. 998.2
b. b̸98.2
c. *72.6-
d. **7.2+
e. $7,654,40+
f. 1969.3
g. $5,678.88b̸
h. b̸-$678.90

* * *

1) g
2) g
3) a,b
4) c,d
5) c
6) f

You have now learned to use all of the edit characters. One additional character, the P character, may be included in the picture of an elementary numeric variable to specify an assumed decimal scaling position outside the number appearing in the data item. When you need further information on this character, you may find it in the Language Specifications Manual under the P picture character.

The next sequence of frames will bring together all of the topics presented in the preceding part of this lesson. Figure 122 will be referred to in most frames. Sequence numbers will be given for your convenience in locating variables in this figure. You may, of course, refer to your Language Specifications Manual at any time.

```
0   0    1    1    2    2    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    DATA DIVISION.
102    FILE SECTION.
103    FD  TRANSACTION-FILE
104        LABEL RECORDS ARE OMITTED.
105    01  ITEM-RECORD.
106        02  CUSTOMER-NUMBER PIC X(5) .
107        02  ITEM-NUMBER PIC X(5).
108        02  UNIT-PRICE PIC 999V99.
109        02  ITEM-DESCRIPTION
110            PIC X(20).
111        02  QUANTITY PIC 999.
112        02  FILLER PIC X(42).
113    01  FD  CUSTOMER-FILE
114        LABEL RECORDS ARE STANDARD
115        BLOCK CONTAINS 6 RECORDS.
116    01  CUSTOMER-RECORD.
117        02  FILLER PIC X.
118        02  CHARGE-ID PIC X(5).
119        02  NAME PIC X(30).
120        02  STREET PIC X(20).
121        02  MAILING.
122            03  CITY PIC X(15).
123            03  STATE PIC X(15).
124            03  ZIP PIC X(5).
201        02  FILLER PIC X(20).
202    FD  BILL-FILE
203        LABEL RECORDS ARE OMITTED.
204    01  BILL-PRINT PIC X(63).
205    WORKING-STORAGE SECTION.
206    77  SAVE-NUMBER PIC X(5).
207    77  SPACING PIC X.
208    77  STATE-CODE PIC X.
209    77  WORK-UNIT-PRICE PIC S999V99.
211    77  WORK-QUANTITY PIC S999.
213    77  SUB-AMOUNT PIC S9999V99.
215    77  TOTAL-AMOUNT PIC S9(5)V99.
217    01  ITEM-LINE.
218        02  FILLER PIC X(3)
219             VALUE IS SPACES.
220        02  ITEM-NUMBER PIC X(5).
221        02  FILLER PIC X(4)
222             VALUE IS SPACES.
223        02  ITEM-DESCRIPTION PIC X(20).
224        02  FILLER PIC X(4)
301             VALUE IS SPACES.
302        02  UNIT-PRICE PIC $$$$.99.
303        02  FILLER PIC X(4)
304             VALUE IS SPACES.
305        02  QUANTITY PIC ZZZ.
306        02  FILLER PIC X(4)
307             VALUE IS SPACES.
308        02  AMOUNT PIC $$,$$$.99.
309    01  PRINT-TOTAL.
310        02  FILLER PIC X(53)
311             VALUE IS SPACES.
312        02  TOTAL PIC $$$,$$$.99.
313    01  DISCOUNT-TOTAL.
314        02  FILLER PIC X(19)
315             VALUE IS SPACES.
316        02  COMMENT-1 PIC X(18)
317             VALUE IS 'MINUS .05 DISCOUNT'.
318        02  FILLER PIC X(19)
319             VALUE IS SPACES.
320        02  DISCOUNT-AMOUNT PIC $$$$.99.
321    01  TAX-TOTAL.
```

```
322        02  FILLER PIC X(19)
323              VALUE IS SPACES.
324        02  COMMENT-2 PIC X(12)
401              VALUE IS 'PLUS .04 TAX'.
402        02  FILLER PIC X(25)
403              VALUE IS SPACES.
404        02  TAX-AMOUNT PIC $$$9.99.
405    01  NAME-LINE.
406        02  FILLER PIC X(10)
407              VALUE IS SPACES.
408        02  NAME-OUT PIC X(30).
409    01  STREET-LINE.
410        02  FILLER PIC X(10)
411              VALUE IS SPACES.
412        02  STREET-OUT PIC X(20).
413    01  ADDRESS-LINE.
414        02  FILLER PIC X(10)
415              VALUE IS SPACES.
416        02  ADDRESS-OUT PIC X(35).
417    01  END-LINE.
418        02  FILLER PIC X(10)
419              VALUE IS SPACES.
420        02  COMMENT-3 PIC X(10)
421              VALUE IS 'PLEASE PAY '.
422        02  TOTAL-SAME PIC *****.**.
423        02  COMMENT-4 PIC X(14)
424              ' WITHIN 30 DAYS'.
```

Figure 122

24. Using Figure 122 match each variable below with the phrases that accurately describe it.

    1) SUB-AMOUNT (213)

    2) ITEM-NUMBER (220)

    3) QUANTITY (305)

    4) UNIT-PRICE (108)

    5) UNIT-PRICE (302)

    6) WORK-UNIT-PRICE (209)

    a. numeric variable

    b. DISPLAY variable

    c. external decimal variable

* * *

1) a
2) b
3) b
4) a,b,c
5) b
6) a

(Remember that a DISPLAY variable is called an external decimal variable.)

25. Refer to Figure 119 to determine which of the following MOVE statements specifies a valid move.

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 ADDRESS-OUT                (416)
               TO BILL-PRINT.              (204)
```

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..

           MOVE TOTAL-AMOUNT               (215)
               TO TOTAL.                   (312)
```

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..

           MOVE UNIT-PRICE OF
               ITEM-RECORD                 (108)
               TO WORK-UNIT-PRICE.         (209)
```

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..

           MOVE SUB-AMOUNT                 (213)
               TO BILL-PRINT.              (204)
```

*   *   *

a,b,c
(d is invalid because the move is from a numeric item that is not an integer to an alphanumeric item.)

---

26. Give the type of move each of the following MOVE statements represents according to Figure 118.

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..

           MOVE ADDRESS-OUT               (416)
               TO BILL-PRINT.             (204)
```

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..

           MOVE TOTAL-AMOUNT              (215)
               TO TOTAL.                  (312)
```

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 UNIT-PRICE OF             (108)
               ITEM-RECORD
               TO WORK-UNIT-PRICE.        (209)
```

* * *

1) alphanumeric
2) edit
3) numeric

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..

           MULTIPLY UNIT-PRICE            (108)
               OF ITEM-RECORD
               BY WORK-QUANTITY           (211)
               GIVING SUB-AMOUNT.         (213)
```

According to the rules of alignment when the statement above is executed:

a. the result will be converted.

b. one computation field will be converted.

c. the result will be 'truncated when placed in the result field.

* * *

b

----

28. Figure 118 shows that in order to place the current value of SUB-AMOUNT (line 213) into the output area of BILL-FILE (line 202), you would execute the statements:

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 SUB-AMOUNT TO BILL-PRINT.
```

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..

        MOVE SUB-AMOUNT TO AMOUNT.      (308)
        MOVE AMOUNT TO BILL-PRINT.
```

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..


        WRITE SUB-AMOUNT.
```

* * *

b

---

29. Use Figures 44 and 121 to write the values that would be printed if the values below were moved to the specified variables.

	Value	Variable	(sequence number)
1)	0000000	TOTAL-SAME	(422)
2)	2	QUANTITY	(305)
3)	51250	TOTAL	(312)
4)	00000	DISCOUNT-AMOUNT	(320)

* * *

Result

1) *****.**

2) ƀƀ2

3) ƀƀƀ$512.50

4) ƀƀƀ$.00

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..

           PERFORM CONVERSION-ROUTINE
               VARYING TEMPERATURE
               FROM -40 BY 5 UNTIL
               TEMPERATURE IS GREATER THAN 120.


       CONVERSION ROUTINE.
           COMPUTE CENTIGRADE = 5 / 9 *
               (TEMPERATURE - 32).
           MOVE CENTIGRADES TO DEGREES-C.
           MOVE TEMPERATURE TO DEGREES-F.
           WRITE OUT-RECORD.
```

In the preceding lesson you wrote the PERFORM statement above to execute the paragraph above. All variables were DISPLAY variables. This execution would be more efficient if:

a. COMP were specified for CENTIGRADE and TEMPERATURE.

b. TEMPERATURE and 32 had the same number of decimal places.

* * *

Both

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..


           PERFORM CONVERSION-ROUTINE
               VARYING TEMPERATURE
               FROM -40 BY 5 UNTIL
               TEMPERATURE IS GREATER THAN 120.

       CONVERSION-ROUTINE.
           COMPUTE CENTIGRADE = 5 / 9 *
               (TEMPERATURE - 32).
           MOVE CENTIGRADE TO DEGREES-C.
           MOVE TEMPERATURE TO DEGREES-F.
           MOVE OUT-RECORD.
```

In CONVERSION-ROUTINE the value 5/9 is computed every time the paragraph is executed. The execution time would be shorter if:

a. the decimal equivalent of 5/9 were used in place of the fraction.

b. the statement COMPUTE FIVE-NINTHS = 5/9 were executed before the PERFORM statement, and FIVE-NINTHS were substituted for 5/9 in CONVERSION-ROUTINE.

* * *

Either of these
(a eliminates the division operation; b eliminates conversion of the literal during execution of the COMPUTE statement.)

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..


       PROCEDURE DIVISION.
       ROUTINE.
           OPEN OUT-FILE.
           COMPUTE FIVE-NINTHS = 5 / 9.
           PERFORM CONVERSION-ROUTINE
               VARYING TEMPERATURE FROM -40
               BY +5 UNTIL TEMPERATURE
               IS GREATER THAN 120.
           CLOSE OUT-FILE
           STOP RUN.
```

Write binary Working-Storage Section entries for:

1) FIVE-NINTHS, which must have a sign and three decimal places.

2) TEMPERATURE, which will contain integer values from -40 to +120.

3) CENTIGRADE, which will have up to four digits and must have two decimal places and a sign.

*   *   *

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  FIVE-NINTHS PIC SV999.
               USAGE IS COMP.
```

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..
```

3)

```
       77  TEMPERATURE PIC S999.
               USAGE IS COMP.


0   0    1    1    2    2    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  CENTIGRADE PIC S99V99.
               USAGE IS COMP.
```

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  OUT-FILE
           LABEL RECORDS ARE OMITTED.
       01  OUT-RECORD.
           02  FILLER PIC X(11) VALUE IS SPACES.
           02  DEGREES-F PIC +999.
           02  FILLER PIC X(10) VALUE IS SPACES
           02  DEGREES-C PIC +99.99.


       WORKING-STORAGE SECTION.
       77  FIVE-NINTHS PIC SV999.
               USAGE IS COMP.
       77  TEMPERATURE PIC S999.
               USAGE IS COMP.
       77  CENTIGRADE PIC S99V99.
               USAGE IS COMP.


       PROCEDURE DIVISION.
       ROUTINE.
           OPEN OUT-FILE.
           COMPUTE FIVE-NINTHS = 5 / 9.
           PERFORM CONVERSION-ROUTINE
               VARYING TEMPERATURE FROM -40
               BY +5 UNTIL TEMPERATURE
               IS GREATER THAN 120.
           CLOSE OUT-FILE.
           STOP RUN.


       CONVERSION-ROUTINE.
           COMPUTE CENTIGRADE = 5 / 9*
               (TEMPERATURE - 32).
           MOVE CENTIGRADE TO DEGREES-C.
           MOVE TEMPERATURE TO DEGREES-F.
           WRITE OUT-RECORD.
```

Rewrite CONVERSION-ROUTINE so that is will be executed more efficiently. Use the variables as 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..


       CONVERSION-ROUTINE.
           COMPUTE CENTIGRADE = FIVE-NINTHS
               * (TEMPERATURE - 32).
           MOVE CENTIGRADE TO DEGREES-C.
           MOVE TEMPERATURE TO DEGREES-F.
           WRITE OUT-RECORD.
```

SUMMARY:

In this lesson you have learned to increase the efficiency of execution of a program by giving the data the appropriate form of storage. You have also learned several additional picture characters that will help you to edit data for production of printed reports.

END OF LESSON 26

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 27

LESSON 27 - TABLE DEFINITION

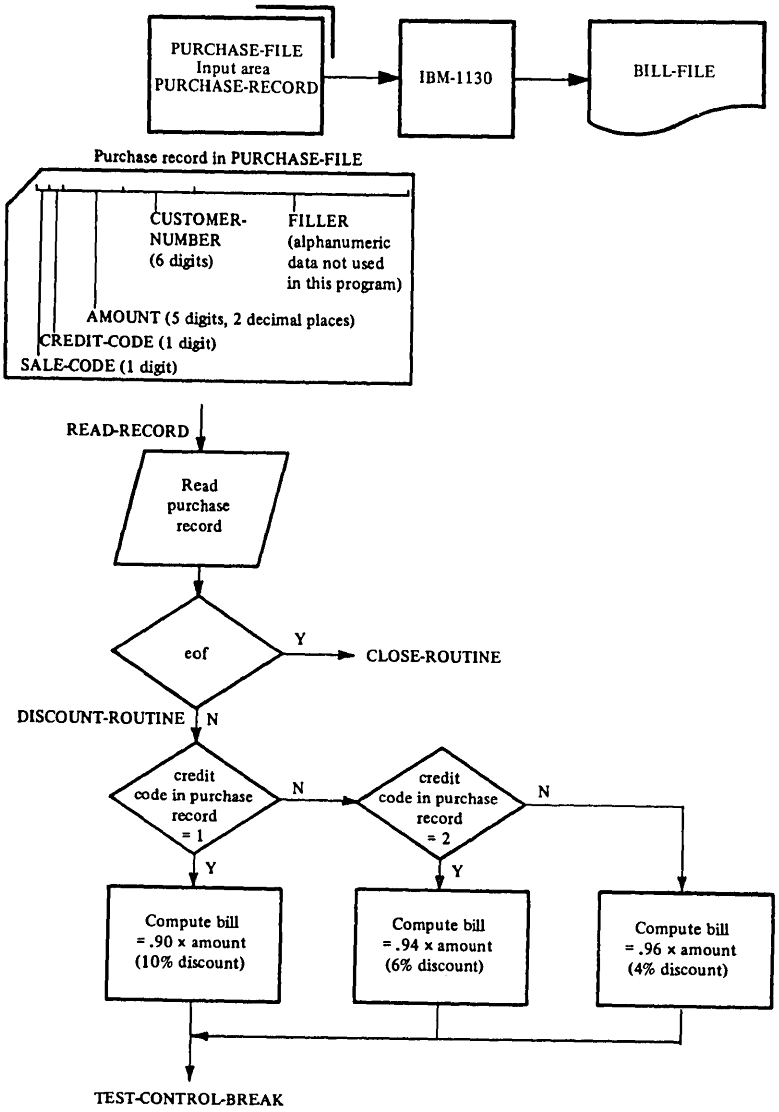
INTRODUCTION

It is often convenient for a programmer to process sets of similar data as tables. In this lesson you will learn to define variables whose values will be tables of data. You will learn different ways to refer to an individual value within a table in order to use it in processing other data. As you are learning to use these table processing techniques you will be coding portions of a billing procedure. Later you will apply what you have learned in coding a program to update an employee master disk-file and a customer master disk-file.

Specific COBOL language features and programming techniques you will learn in this lesson are:

Subscripted names
Qualified subscripted names
OCCUkS clause
Use of tables of data in processing files.

This lesson will require approximately three quarters of an hour.


PURCHASE-FILE
Input area
PURCHASE-RECORD
IBM-1130
BILL-FILE
Purchase record in PURCHASE-FILE
CUSTOMER-NUMBER (6 digits)
FILLER (alphanumeric data not used in this program)
AMOUNT (5 digits, 2 decimal places)
CREDIT-CODE (1 digit)
SALE-CODE (1 digit)
READ-RECORD
Read purchase record
eof
Y
CLOSE-ROUTINE
DISCOUNT-ROUTINE
N
credit code in purchase record = 1
N
credit code in purchase record = 2
N
Y
Y
Compute bill = .90 x amount (10% discount)
Compute bill = .94 x amount (6% discount)
Compute bill = .96 x amount (4% discount)
TEST-CONTROL-BREAK

The system flowchart and the problem flowchart segment for READ-RECORD and DISCOUNT-ROUTINE are for a billing procedure. Discounts are to be given to customers on the basis of their credit ratings. A customer's credit rating is indicated by the credit code in each purchase record. A customer who has good credit rating will have a credit code of 1. A custormer with a medium rating will have a credit code of 2, and one with fair rating will have a credit code of 3. In paragraph DISCOUNT-ROUTINE the credit code is to be tested and the bill for the purchase is to be computed according to the results of the test.

Figure 123

1. Read the problem description in Figure 123. In the record description entry for the input area PURCHASE-RECORD, the variable CREDIT-CODE could be defined as:

   a. a data name, and a test such as
      IF CREDIT-CODE EQUAL TO 1
      could then be used in DISCOUNT-ROUTINE.

   b. a conditional variable with the associated condition names GOOD, MEDIUM, and FAIR and their respective values 1, 2, and 3, and a test such as
      IF GOOD
      could then be used in
      DISCOUNT-ROUTINE.

* * *

Either (If CREDIT-CODE were defined as a conditional variable, either test could be used in DISCOUNT-ROUTINE.)

----

2. In order for the computations in DISCOUNT-ROUTINE in Figure 123 to be done efficiently, a purchase record should be read into an input area and:

   a. elementary variables of the input area should be specified in COMPUTE statements.

   b. elementary values to be used in computations should be moved to elementary COMP variables in working storage.

* * *

b

----

3. Code the Data Division with entries for PURCHASE-FILE, the input area PURCHASE-RECORD defining CREDIT-CODE as a conditional variable, BILL-FILE, BILL-RECORD, the output area for BILL-FILE consisting of 120 alphanumeric characters, and a level 77 working-storage variable for any value of PURCHASE-RECORD that is to be used in the computations in DISCOUNT-ROUTINE described in Figure 123.

* * *

```
0   0    1    1    2    2    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 PURCHASE-FILE
           LABEL RECORDS ARE OMITTED.
       01  PURCHASE-RECORD.
           02 SALE-CODE PIC 9.
           02 CREDIT-CODE PIC 9.
              88 GOOD VALUE 1.
              88 MEDIUM VALUE 2.
              88 FAIR VALUE 3.
           02 AMOUNT PIC 999V99.
           02 CUSTOMER-NUMBER PIC 9(6).
           02 FILLER PIC X(67).
       FD  BILL-FILE
           LABEL RECORDS ARE OMITTED.
       01  BILL-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       77  AMOUNT-WS USAGE IS COMP.
               PIC 999V99.
```

(The optional word IS has been omitted from the VALUE clause.)

---

4. In order for the computations in DISCOUNT-ROUTINE in Figure 119 to be done efficiently, the percentages should be:

   a. set up as elementary COMP variables in working storage.

   b. written as numeric literals in COMPUTE statements.

* * *

a
(COMP may be specified at the group level. It will then apply to each elementary variable in the group.)

5.

PERCENT-RECORD (elementary variable)	(value)
PERCENT-1	.90
PERCENT-2	.94
PERCENT-3	.96

Continue coding the Working-Storage Section by writing the entries for:

1) the level 77 variable resulting from computation in DISCOUNT-ROUTINE. (The result is to represent a dollar amount which will have been rounded to the nearest cent. Remember that any level 77 entry must precede any level 01 entry in working storage.)

2) the record of percentages illustrated above.

*   *   *

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  BILL USAGE IS COMP PIC 999V99.
```

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  PERCENT-RECORD USAGE IS COMP.
           02 PERCENT-1 PIC V99 VALUE .90.
           02 PERCENT-2 PIC V99 VALUE .94.
           02 PERCENT-3 PIC V99 VALUE .96.
```

6. Code the Procedure Division entries for paragraphs READ-RECORD and DISCOUNT-ROUTINE in Figure 123. Include a statement to make the appropriate value in PURCHASE-RECORD available in working storage for computations. Remember that the result of the computation will be a dollar amount and should be rounded to the nearest cent.

*   *   *

```
0   0   1   1   2   2   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-RECORD.
                  READ PURCHASE-FILE
                      AT END GO TO CLOSE-ROUTINE.
                  MOVE AMOUNT TO AMOUNT-WS.
              DISCOUNT-ROUTINE.
                  IF GOOD
                      COMPUTE BILL ROUNDED
                          = AMOUNT-WS * PERCENT-1
                      ELSE
                      IF MEDIUM
                          COMPUTE BILL ROUNDED
                              =AMOUNT-WS * PERCENT-2
                          ELSE
                          COMPUTE BILL ROUNDED
                              = AMOUNT-WS * PERCENT-3.
```

---

7. A set of similar data described with the same picture may be defined and processed as a table. A record that could be defined and processed as a table would be a record of:

a. insurance premiums.

b. student personal data such as name, age, and entrance scores.

*   *   *

a

---

8. Data that could be defined and processed as a table would be contained in a record of:

a. percentages.

b. prices.

*   *   *

Either

9.

PERCENT-RECORD (elementary variable)	(value)
PERCENT-1	.90
PERCENT-2	.94
PERCENT-3	.96

unique name

PERCENT-TABLE (elementary variable)	(value)
PERCENT(1)	.90
PERCENT(2)	.94
PERCENT(3)	.96

common name subscript

The illustration above shows how the record of percentages used in the problem in Figure 123 can be set up as a table. The table illustration shows that each value in a table can be referred to in a COBOL statement by a:

a. unique data name.

b. common data name and a subscript.

* * *

b

PERCENT-TABLE is a one-dimensional table. Each element of the table can be referred to by a data name and one subscript. It is possible to set up two- and three-dimensional tables in a COBOL program and to refer to table elements with two and three subscripts, respectively. Since you will learn to use only one-dimensional tables in this course, however, you will be using a data name with one subscript to refer to a table element.

10.

PERCENT-TABLE	
PERCENT(1)	.90
PERCENT(2)	.94
PERCENT(3)	.96

An element of a one-dimensional table may be referred to by writing the data name followed by a space followed by the subscript enclosed in parentheses. Code the Procedure Division entries for DISCOUNT-ROUTINE in Figure 123 referring to the percentages as elements of the table 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..

       DISCOUNT-ROUTINE.
           IF GCOD
               COMPUTE BILL ROUNDED
                   = AMOUNT-WS * PERCENT (1)
               ELSE
               IF MEDIUM
                   COMPUTE BILL ROUNDED
                       = AMOUNT-WS * PERCENT (2)
                   ELSE
                   COMPUTE BILL ROUNDED
                       = AMOUNT-WS
                       * PERCENT (3).
```

---

11. Table elements may have forms of subscripts other than integers. Any variable that has integer values may be specified as a subscript in the name of a table element in a Procedure Division entry. In Figure 123, a variable to be used as a subscript with PERCENT:

    a. must be defined with a picture of 9's.

    b. could be CREDIT-CODE.

* * *

Both

---

12. In Figure 123 the name PERCENT (CREDIT-CODE) would refer to the value:

    a. .90 when the value of CREDIT-CODE is 1.

    b. .94 when the value of CREDIT-CODE is 2.

    c. .96 when the value of CREDIT-CODE is 3.

* * *

All of these

13.

PERCENT-TABLE	
PERCENT (1)	.90
PERCENT (2)	.94
PERCENT (3)	.96

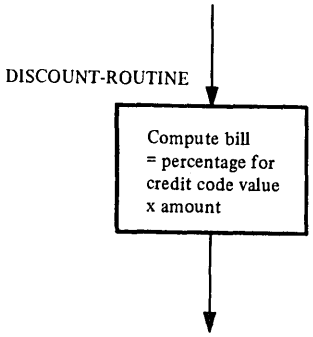


Figure 124

If CREDIT-CODE were to be specified as a subscript in a discount routine for the problem described in Figure 123, the flow chart would be modified to look like the flow chart above. Code the Procedure Division entries for DISCOUNT-ROUTINE following the modified 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..

             DISCOUNT-ROUTINE.
                 COMPUTE BILL ROUNDED
                     = AMOUNT-WS
                     * PERCENT (CREDIT-CODE).
```

(This is an example of the way table processing can increase the efficiency of a program.)

---

14. A subscript may be:

a. any integer.

b. any variable.

c. any variable with integer values.

* * *

a,c

---

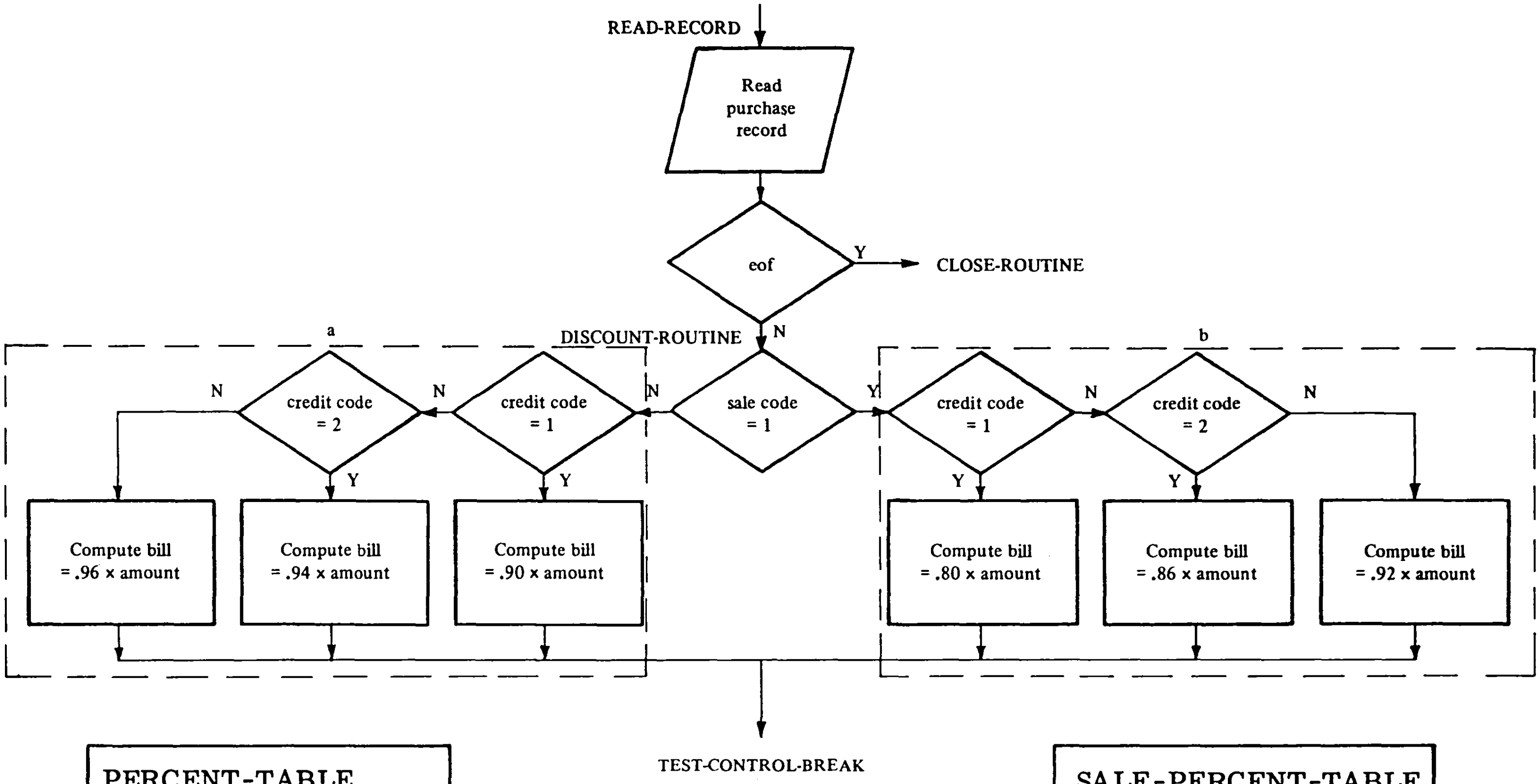
15. A variable to be specified as a subscript may be defined with a picture of:

a. 9's and a V specifying a decimal place.

b. 9's.

* * *

b



PERCENT-TABLE	
PERCENT (1)	.90
PERCENT (2)	.94
PERCENT (3)	.96

SALE-PERCENT-TABLE	
PERCENT (1)	.80
PERCENT (2)	.86
PERCENT (3)	.92

Each month certain products are announced as sale items. Discounts for these products are greater than regular discounts, but, like the regular discounts, are determined by the customer's credit rating. A purchase record for a sale item will contain a sale code of 1. Regular items are indicated by a zero. In paragraph DISCOUNT-ROUTINE the sale code is to be tested and either branch a or b is to be executed. Since there is a set of percentages for each branch a second table should be set up for sale item discounts.

Figure 125

16. Read the problem description in Figure 125. The problem:

a. requires two percentage tables.

b. uses the same percentage table for both regular and sale purchases.

* * *

a

---

17. For the problem in Figure 125 PERCENT (1):

a. is the elementary name of both the values .80 and .90.

b. would have to be qualified to refer to a single value.

* * *

Both

---

18. PERCENT OF SALE-PERCENT-TABLE (1)

In Figure 125 the value .80 can be referred to in a COBOL statement by qualifying the elementary name as shown above. When the name of a table element is qualified, the subscript of the element immediately follows:

a. the elementary name.

b. the table name.

* * *

b

---

19. In Figure 125 the value .92 could be referred to by the name:

a. PERCENT (3) OF SALE-PERCENT-TABLE.

b. PERCENT OF SALE-PERCENT-TABLE (3).

* * *

b

---

20. In Figure 125, if CREDIT-CODE had the value 2, the value .94 could be referred to by the name:

   a. PERCENT OF
      SALE-PERCENT-TABLE (CREDIT-CODE).

   b. PERCENT (CREDIT-CODE)
      OF PERCENT-TABLE.

   c. PERCENT OF
      PERCENT-TABLE (CREDIT-CODE)

* * *

c

---

21. Modify the flow chart in Figure 125 to use CREDIT-CODE as a subscript to refer to the elements of the tables, and code the Procedure Division entries for the modified flow chart.

* * *

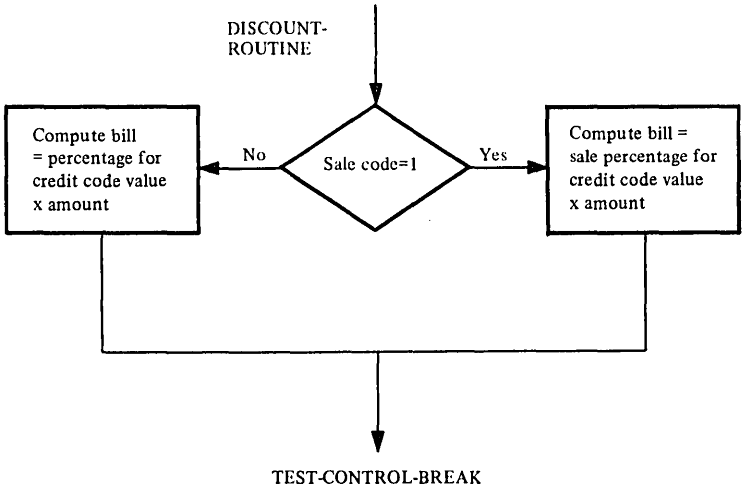

DISCOUNT-ROUTINE
Compute bill = percentage for credit code value x amount
No
Sale code=1
Yes
Compute bill = sale percentage for credit code value x amount
TEST-CONTROL-BREAK

Figure 126

```
0   0    1    1    2    2    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-ROUTINE.
           IF SALE-CODE EQUAL TO 1
               COMPUTE BILL ROUNDED
                   = AMOUNT-WS
                   * PERCENT
                       OF SALE-PERCENT-TABLE
                       (CREDIT-CODE)
               ELSE
               COMPUTE BILL ROUNDED
                   = AMOUNT-WS
                   * PERCENT
                       OF PERCENT-TABLE
                       (CREDIT-CODE).
```

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..

        01  PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES
                PIC V99.
```

When percentages are to be processed as a table, and referred to by subscripted names, the table may be defined by the Data Division entries shown above. The common data name is specified in a single data description entry with the OCCURS clause specifying the number of table elements. Match the correct facts about PERCENT-TABLE with the appropriate portion of the Data Division entries.

1) USAGE COMP

2) OCCURS 3 TIMES

3) PIC V99

a. PERCENT-TABLE will have two elements.

b. PERCENT-TABLE will have three elements.

c. Each element of PERCENT-TABLE will have two digits.

d. Each element of PERCENT-TABLE will have three digits.

e. Each element of PERCENT-TABLE will be stored as binary.

f. Each element of PERCENT-TABLE will be stored in a form for efficient computation.

*   *   *

1) e,f
2) b
3) c

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..

        01  PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES
               PIC V99.
```

In Figure 119 the restrictions for the OCCURS clause indicate that in the record description entry above the OCCURS clause could be specified for:

a. the level 01 variable PERCENT-TABLE instead of the level 02 variable PERCENT.

b. the variable PERCENT if it were defined as level 77 instead of level 02 subordinate to PERCENT-TABLE.

* * *

Neither (The OCCURS clause cannot be specified for a level 01 or a level 77 variable.)

----

24. Code the record description entry for the table of sale percentages shown in Figure 125.

* * *

```
0   0    1    1    2    2    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  SALE-PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES PIC V99.
```

Up to this point you have learned to define variables whose values are to be tables of data and to refer to table elements in COBOL statements specifying integers as subscripts as well as variables whose values are integers.

SUMMARY

You have started to study the valuable programming technique of table construction and usage. Many applications require tables of orderly arranged data, the elements of which may be fixed or variable.

END OF LESSON 27

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 28

LESSON 28 - USE OF TABLES

INTRODUCTION

In this lesson you will continue your study of table construction and usage. You will learn to assign values to tables as well as find table values that satisfy certain conditions in order to use the subscripts of those values in other processing.

A specific COBOL programming technique you will learn in this lesson is:

Assignment of values to table variables

This lesson will require approximately three quarters of an hour.

PERCENT-RECORD (elementary variable)	(value)
PERCENT-1	.90
PERCENT-2	.94
PERCENT-3	.96

unique name

PERCENT-TABLE (elementary variable)	(value)
PERCENT(1)	.90
PERCENT(2)	.94
PERCENT(3)	.96

common name subscript

When variables that are to have similar values such as percentages are defined with unique names as in PERCENT-RECORD, initial values may be assigned in VALUE clauses in the Data Division. When variables that are to have similar values are defined as table elements as in PERCENT-TABLE, they are defined in a way that reduces the amount of coding in the Data Division. Then instead of assigning values in the Data Division, the programmer must provide statements in the Procedure Division to assign values to the table elements. In the next sequence you will learn to assign values to table elements.

---

1. A value may be assigned to a variable in the Procedure Division by:

   a. accepting the value through the console keyboard.

   b. accepting the value through the card reader.

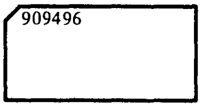
   c. reading the value from a file.

   d. moving a literal to the variable.

* * *

Any of these

2.



PERCENT-TABLE	
PERCENT (1)	.90
PERCENT (2)	.94
PERCENT (3)	.96

Figure 127

```
0   0    1    1    2    2    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  E-VARIABLE PIC V99.
        01  PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES PIC V99.
```

When a table name is specified in an ACCEPT statement the number of characters specified in the picture for the table elements is transmitted to each element beginning with the element whose subscript is 1 and the first card column. If instead an elementary name is specified, the number of characters specified in the picture, beginning with the first card column, is transmitted to the single elementary variable. In each case the unused remainder of the card is disregarded. Refer to the record description entry and the card being accepted above and match the correct effects with the entries 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..

            ACCEPT PERCENT-TABLE.
```

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 E-VARIABLE.
            MOVE E-VARIABLE TO PERCENT (1).
```

a. The values 90, 94 and 96 are transmitted to the elements of PERCENT-TABLE with subscripts 1, 2, and 3, respectively.

b. The only value transmitted is 90, which is transmitted and moved to the element of PERCENT-TABLE with subscript 1.

c. Data in card columns 3 through 80 will be disregarded.

d. Data in card columns 7 through 80 will be disregarded.

* * *

1) a,d
2) b,c
(A subscripted name may not be specified in an ACCEPT statement. To transmit a value to a specific table element, the programmer must define an elementary variable, specify that variable in the ACCEPT statement, and then move the value from the elementary variable to the specific table element.)

3.

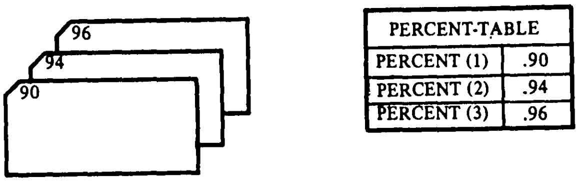


Figure 128

```
0   0    1    1    2    2    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  E-VARIABLE PIC V99 USAGE IS COMP.
       01  PERCENT-TABLE.
           02 PERCENT OCCURS 3 TIMES PIC V99.
```

The values in the cards shown above are to be transmitted to the elements of PERCENT-TABLE. Select the statement(s) that would assign the values 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..

           ACCEPT PERCENT-TABLE.
```

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..

           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE TO PERCENT (1).
           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE TO PERCENT (2).
           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE TO PERCENT (3).
```

* * *

b

(Statement a would access one card and, based on the picture for the table elements, would assign the data in columns 1 and 2 (90) to the element with subscript 1, the data in columns 3 and 4 (blanks) to the element with subscript 2, and the data in columns 5 and 6 (blanks) to the element with subscript 3. The OCCURS clause specifies only three elements, and the remaining 74 columns of the card would be disregarded.)

---

4. If the size of the variable specified in an ACCEPT statement is greater than 80 characters, the 80 characters from the card are stored, left-aligned, in the receiving data item, and the remainder of the data item is filled with spaces. Match the effects of the ACCEPT statement with the record description entries 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..

           ACCEPT AMOUNT-TABLE.
```

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  AMOUNT-TABLE.
           02 AMOUNT OCCURS 20 TIMES
               PIC X(8).
```

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  AMOUNT-TABLE.
           02 AMOUNT OCCURS 10 TIMES
               PIC X(6).
```

a. One card will be accessed and values will be assigned to the table elements with subscripts 1 through 10.

b. Two cards will be accessed and values will be assigned to all table elements.

c. Data in columns 9 through 80 of the second card will be disregarded.

* * *

1) a
2) a

---

5. An elementary item described with the

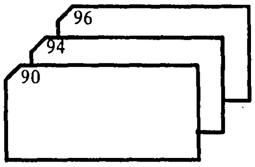
USAGE IS INDEX

clause is called an index data item and contains a value which must correspond to an occurrence number of a table.

```
0   0    1    1    2    2    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  IDX PICTURE 9 VALUE 1 USAGE IS INDEX.
              77  X PICTURE S9V99 VALUE 0.01.
              77  Y PICTURE S9V99 VALUE 0.01.
              01  WORK-REC USAGE IS COMPUTATIONAL.
                  03  E PICTURE S99V99999 OCCURS 5 TIMES.


              PROCEDURE DIVISION.
              BEGIN.
                  PERFORM EXP VARYING IDX FROM 1 BY 1 UNTIL
                  IDX EQUAL TO 5.
                      .
                      .
                      .
              EXP. COMPUTE E(IDX) = 1 + X + X ** 2 / 2 + X ** 3 / 6
                  + X ** 4 / 24 + X ** 5 / 120 + X ** 6 / 720.
                  ADD Y TO X.
```

The rules of the usage clause are described in the Language Specifications Manual.

6.



PERCENT-TABLE	
PERCENT (1)	.90
PERCENT (2)	.94
PERCENT (3)	.96

Figure 129

```
0   0    1    1    2    2    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  E-VARIABLE PIC V99.
       77  CREDIT-CODE PIC 9.
       01  PERCENT-TABLE USAGE IS COMP.
           02 PERCENT OCCURS 3 TIMES PIC V99.
```

If table values are punched one per card, a programmer can transmit each value to a specific table element by coding an ACCEPT statement and a MOVE statement for each table element. He could also code a paragraph with a single ACCEPT statement and a single MOVE statement and code a statement to perform that paragraph for each table element. The latter technique requires only minimal coding for tables of any size. Based on what you have already learned about the PERFORM statement, select the coding example(s) that would assign appropriate values to PERCENT-TABLE if the values have been punched into the cards shown above.

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..

           PERFORM ACCEPT-TABLE-VALUE
               VARYING CREDIT-CODE
               FROM 1 BY 1
               UNTIL CREDIT-CODE GREATER THAN 3.
                   .
                   .
                   .
       ACCEPT-TABLE-VALUE.
           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE
               TO PERCENT (CREDIT-CODE).
```

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..

           MOVE 1 TO CREDIT-CODE.
           PERFORM ACCEPT-TABLE-VALUE
               UNTIL CREDIT-CODE GREATER THAN 3.
                   .
                   .
                   .
       ACCEPT-TABLE-VALUE.
           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE
               TO PERCENT (CREDIT-CODE).
           ADD 1 TO CREDIT-CODE.
```

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..

           MOVE 1 TO CREDIT-CODE.
           PERFORM ACCEPT-TABLE VALUE 3 TIMES.
             .
             .
             .
       ACCEPT-TABLE-VALUE.
           ACCEPT E-VARIABLE.
           MOVE E-VARIABLE
               TO PERCENT (CREDIT-CODE).
           ADD 1 TO CREDIT-CODE.
```

*   *   *

Any of these

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.
       FILE SECTION.
       FD  TABLE-FILE
           LABEL RECORDS ARE OMITTED.
       01  TABLE-RECORD.
           02 TABLE-VALUE PIC V99.
           02 FILLER PIC X(78).
           .
           .
           .
       WORKING-STORAGE SECTION.
       77  CREDIT-CODE PIC 9.
       02  PERCENT-TABLE USAGE IS COMP.
           02 PERCENT OCCURS 3 TIMES PIC V99.
           .
           .
           .
       PROCEDURE DIVISION.
       INITIAL-ROUTINE.
           OPEN TABLE-FILE.
           PERFORM READ-TABLE
               VARYING CREDIT-CODE
               FROM 1 BY 1
               UNTIL CREDIT-CODE GREATER THAN 3.
           .
           .
           .
       READ-TABLE.
           READ TABLE-FILE
               AT END CLOSE TABLE-FILE
                   GO TO NEXT-PARAGRAPH.
           MOVE TABLE-VALUE
               TO PERCENT (CREDIT-CODE).
```

Figure 130

A technique for reading table values from a file is shown in the figure above. This technique:

a. is used for values that are punched one per card.

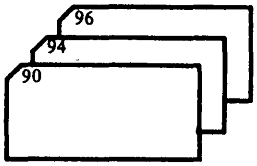
b. is used for values that are punched beginning in card column 1.

c. is done by coding a paragraph containing a single READ statement and a single MOVE statement and coding a statement to perform the paragraph for each element in the table.

* * *

All of these

8.



PERCENT-TABLE	
PERCENT (1)	.90
PERCENT (2)	.94
PERCENT (3)	.96

Figure 131

```
0   0    1    1    2    2    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  CREDIT-CODE PIC 9.
        01  SALE-PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES PIC V99.
```

Code the following:

1) Data Division entries to treat the values in the cards shown above as a file called TABLE-FILE-2.

2) Procedure Division entries to read the values and assign them to the appropriate elements of SALE-PERCENT-TABLE.

*   *   *

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..

        FD  TABLE-FILE-2
            LABEL RECORDS ARE OMITTED.
        01  TABLE-RECORD-2.
            02 TABLE-VALUE-2 PIC V99.
            02 FILLER PIC X(78).
```

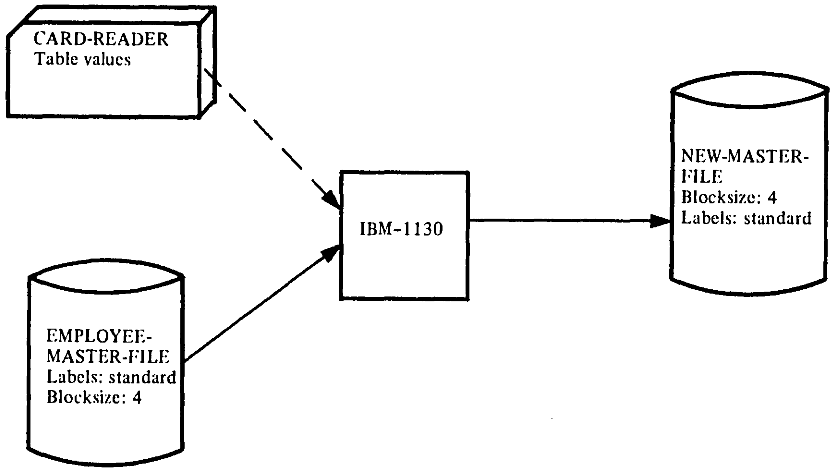
2)

```
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..

            OPEN TABLE-FILE-2
            PERFORM READ-TABLE-2
                VARYING CREDIT-CODE
                FROM 1 BY 1
                UNTIL CREDIT-CODE GREATER THAN 3.
                 .
                 .
                 .
        READ-TABLE-2.
            READ TABLE-FILE-2
                AT END CLOSE TABLE-FILE-2.
                    GO TO NEXT-PARAGRAPH.
            MOVE TABLE-VALUE-2
                TO PERCENT (CREDIT-CODE).
```

(Whenever table values in cards are to be used to process a card file and the table values are to be accepted, they should be accepted before the card file is opened. If the table values are to be treated as a file, the table file should be opened, read, and closed before the second card file is opened.)

In the initial portion of this lesson you learned to specify table elements in statements in the Procedure Division. You learned to use integers as well as variables that have integer values as subscripts to refer to values of specific table elements. Next you learned to use the OCCURS clause to define tables in the Data Division. Finally you learned two ways to assign values to the table elements in the Procedure Division. Now you will have an opportunity to use these techniques and language features in a program of the type you may encounter in a data-processing center.

A manufacturing company has approved a pay rate increase for all employees of the company. As a result, the employee master file used in payroll processing must be updated to reflect the increase. In the next sequence of frames you will be coding a solution for this problem.



Records in EMPLOYEE-MASTER-FILE
and NEW-MASTER-FILE

EMPLOYEE-NUMBER	EMPLOYEE-NAME	SOCIAL-SECURITY	MARITAL-STATUS	DEPENDENTS	GRADE	PAYRATE	MEDICAL	LIFE	RETIREMENT
6 digits	21 letters	9 digits	1 letter	2 digits	2 digits	4 digits 2 decimal places	4 digits 2 decimal places	4 digits 2 decimal places	4 digits 2 decimal places

```
0   0   1   1   2   2   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-PAYRATE-UPDATE.
```

```
0   0   1   1   2   2   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 EMPLOYEE-MASTER-FILE
               ASSIGN TO DF-1-800-X.
           SELECT NEW-MASTER-FILE
               ASSIGN TO DF-2-900-X.
```

Employees of a manufacturing company have been assigned a grade 1 through 10 according to the skills and knowledge required by their jobs. Recently the company has approved pay rate increases for all employees. The increases vary with employee grade as shown in the table below. A program is to be written to update the pay rate portion of each employee master record. The system flowchart, a diagram of the employee master record, and the first two divisions of the program are shown above.

GRADE	INCREASE
1	3%
2	3%
3	3%
4	5%
5	5%
6	8%
7	8%
8	8%
9	10%
10	10%

Figure 132

9. Read the problem description in Figure 132. A programmer could define an input area and an output area with data description entries for each of the values shown in the employee master record diagram in Figure 132. Then he could code the program to read a record, move the pay rate value to variable and each of the other values to the output area. The grade, which would still be available in the input area, could be specified as a subscript to refer to the correct percent increase for computing the new pay rate. After moving the new pay rate to the output area, the program could write the new record. The problem could be solved with less coding, however. Since only the grade and pay rate must be referred to directly, it is unnecessary to code data description entries for each of the other values in the records. The first 39 characters and the last 12 characters in the record may be defined as FILLER items, provided the record is read into the output area or read and moved as a group. Code the File Section of the Data Division to provide for the files EMPLOYEE-MASTER-FILE and NEW-MASTER-FILE with the latter type of processing.

* * *

```
0   0   1   1   2   2   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-MASTER-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  EMPLOYEE-MASTER-RECORD.
           02 FILLER PIC X(39).
           02 GRADE PIC 99.
           02 PAYRATE PIC 99V99.
           02 FILLER PIC X(12).
       FD  NEW-MASTER-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  NEW-MASTER-RECORD.
           02 FILLER PIC X(39).
           02 NEW-GRADE PIC 99.
           02 NEW-PAYRATE PIC 99V99.
           02 FILLER PIC X(12).
```

10. If an employee is currently performing a grade 1 job, he will receive an increase of 3 percent according to the table in Figure 132. His new pay rate will be 1.03 times his current pay rate. For the most efficient computation a programmer should code a COMPUTE statement in which:

a. the value 1.03 is written as a numeric literal.

b. he specifies a variable and assigns the value 1.03.

* * *

b

---

11. Code the Working-Storage Section of the Data Division defining:

1) a variable for the values used in computation of the new pay rate, except the percent increases.

2) FACTOR-TABLE to allow the percent factors of 1 plus the percent increase in Figure 132 to be processed as a table.

3) a variable to be used as a subscript.

4) a variable to which a percent factor of 1 plus the percent increase may be transmitted from the card reader.

* * *

```
0   0    1    1    2    2    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  PAYRATE-WS USAGE IS COMP PIC 99V99.
       77  E-VARIABLE PIC 9V99.
       77  GRADE PIC 99.
       01  FACTOR-TABLE USAGE IS COMP.
           02 FACTOR OCCURS 10 TIMES PIC 9V99.
```

12.

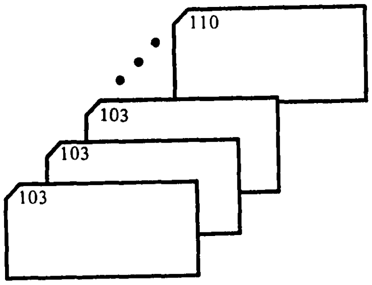


Figure 133

Code the Procedure Division for the problem in Figure 132 to accept the table values that have been punched into cards as shown above and then to update the employee master records.

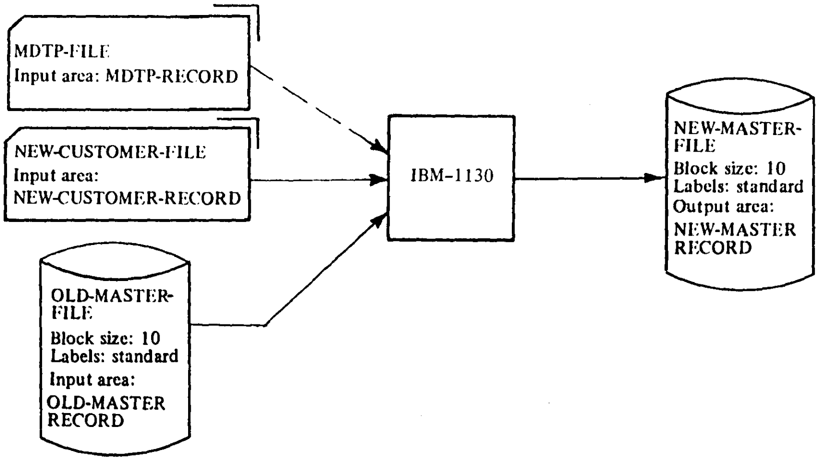
* * *

```
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..

       PROCEDURE DIVISION.
       INITIAL-ROUTINE.
           PERFORM ACCEPT-TABLE-VALUE
               VARYING GRADE
               FROM 1 BY 1
               UNTIL GRADE GREATER THAN 10.
           OPEN INPUT EMPLOYEE-MASTER-FILE
               OUTPUT NEW-MASTER-FILE.
       UPDATE-RECORD-ROUTINE.
           READ EMPLOYEE-MASTER-FILE
               INTO NEW-MASTER-FILE
               AT END GO TO CLOSE-ROUTINE.
           MOVE PAYRATE TO PAYRATE-WS.
           COMPUTE PAYRATE-WS ROUNDED
               = FACTOR (GRADE) * PAYRATE-WS.
           MOVE PAYRATE-WS TO NEW-PAYRATE.
           WRITE NEW-MASTER-RECORD.
           GO TO UPDATE-RECORD-ROUTINE.
       ACCEPT-TABLE-VALUE.
           ACCEPT E-VARIABLE.
            MOVE E-VARIABLE TO FACTOR (GRADE).
       CLOSE-ROUTINE.
           CLOSE EMPLOYEE-MASTER-FILE
               NEW-MASTER-FILE.
           STOP RUN.
```

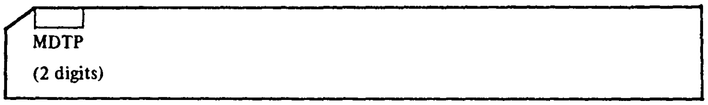
Thus far you have learned that you may specify an integer or a variable that has integer values as a subscript to refer to a specific element of a table so that you can use the value of that element in some way. For example, in preceding frames you have specified CREDIT-CODE as a subscript to refer to a specific element to compute the bill for a customer. In other data-processing problems you may wish to find a table element whose value satisfies a certain condition so that you can use the subscript of that element in some way. You will be coding a program to use a table in this way in the next sequence of frames.

---

13. Read the problem description in Figure 134. Code the Identification and Environment Divisions for the program called MASTER-FILE-UPDATE.

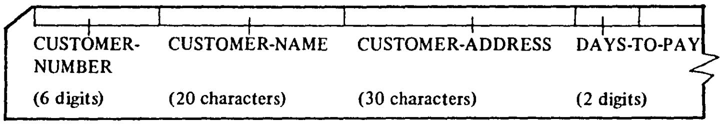


Maximum days to pay record

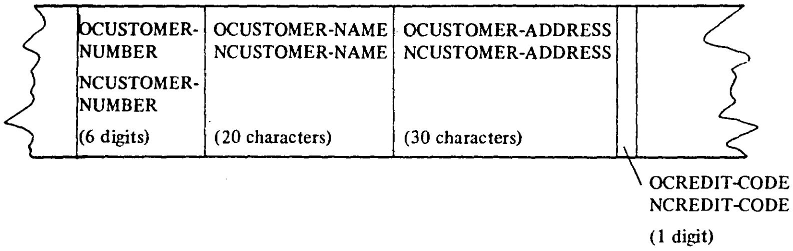


MAX-DAYS-TO-PAY-TABLE	
MAX-DAYS-TO-PAY (1)	10
MAX-DAYS-TO-PAY (2)	20
MAX-DAYS-TO-PAY (3)	30

New customer record



Old and new master records



A customer master file is to be updated by adding records for new customers after assigning a credit code based on how promptly the customer paid for purchases during the preceding month. NEW-CUSTOMER-FILE contains a card record for each new customer with the new customer number, customer name, customer address, and the number of days from the day the bill was issued until the day the payment was received. The records have been arranged in order by customer number. The last record in each file is a dummy record with a customer number of 999999. A test routine is to determine whether a record for an old or new customer is to be written into the new master file. The new master file is to be created with records for both old and new customers in order by customer number. When a record for an old customer is to be written, it will be transferred without changes to the new master file. (It can therefore be written directly from the input area.) When a record for a new customer is to be written, it is to be built from the customer number, name and address in the record from NEW-CUSTOMER-FILE and the credit code is to be determined in the following way. The value of DAYS-TO-PAY in the record from NEW-CUSTOMER-FILE is to be compared with elements of MAX-DAYS-TO-PAY-TABLE shown above. When the value of DAYS-TO-PAY is less than or equal to an element, the subscript of that element is to be the customer's credit code.

A system flowchart and diagrams of the input and output records are shown above. A program flow chart segment for the routine to build the new master record is shown in Figure 135.

Figure 134

```
                         *        *        *

0   0    1    1    2    2    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-FILE-UPDATE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT NEW-CUSTOMER-FILE
               ASSIGN TO RD-1442.
           SELECT OLD-MASTER-FILE
               ASSIGN TO DF-1-600.
           SELECT NEW-MASTER-FILE
               ASSIGN TO DF-2-800.
           SELECT MDTP-FILE
               ASSIGN TO RD-1442.

------------------------------------------------------------------------
```

14. Code the File Section of the Data Division to provide for the files described and illustrated in Figure 134.

* * *

```
0   0    1    1    2    2    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  NEW-CUSTOMER-FILE
           LABEL RECORDS ARE OMITTED.
       01  NEW-CUSTOMER-RECORD.
           02 CUSTOMER-NUMBER PIC 9(6).
           02 CUSTOMER-NAME PIC X(20).
           02 CUSTOMER-ADDRESS PIC X(30).
           02 DAYS-TO-PAY PIC 99.
           02 FILLER PIC X(22).
       FD  OLD-MASTER-FILE
           LABEL RECORDS ARE STANDARD
       01  OLD-MASTER-RECORD.
           02 OCUSTOMER-NUMBER PIC 9(6).
           02 OCUSTOMER-NAME PIC X(20).
           02 OCUSTOMER-ADDRESS PIC X(30).
           02 OCREDIT-CODE PIC 9.
       FD  NEW-MASTER-FILE
           LABEL RECORDS ARE STANDARD
       01  NEW-MASTER-RECORD.
           02 OCUSTOMER-NUMBER PIC 9(6).
           02 OCUSTOMER-NAME PIC X(20).
           02 OCUSTOMER-ADDRESS PIC X(30).
           02 OCREDIT-CODE PIC 9.
       FD  MDTP-FILE
           LABEL RECORDS ARE OMITTED.
       01  MDTP-RECORD.
           02 MDTP PIC 99.
           02 FILLER PIC X(78).
```

---

15. Code the Working-Storage Section of the Data Division to provide for the table described and illustrated in Figure 134 and the elementary variable CREDIT-CODE to be used as a subscript.

* * *

```
0   0    1    1    2    2    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  CREDIT-CODE PIC 9.
       01  MAX-DAYS-TO-PAY-TABLE.
           02 MAX-DAYS-TO-PAY
               OCCURS 3 TIMES                     (22)
               PIC 99.
```

---

16. A record would be written into NEW-MASTER-FILE 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 NEW-MASTER-RECORD
            FROM OLD-MASTER-RECORD.
```

if the record is of an old customer.

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 NEW-MASTER-RECORD
            FROM NEW-CUSTOMER-RECORD.
```

if the record is of a new customer.

* * *

a (For a new customer the appropriate variables would be moved to the output area and then NCREDIT code would be determined. The statement used in this case would be WRITE NEW-MASTER-RECORD.)

---

17. By comparing the days to pay for the new customer with the table elements, the program is to determine the subscript to be used as the credit code for the new customer. This could be done by specifying:

    a. CREDIT-CODE (which is defined as an elementary variable in working storage) as the subscript in the statement to test the days to pay and then moving the value of CREDIT-CODE to NCREDIT-CODE.

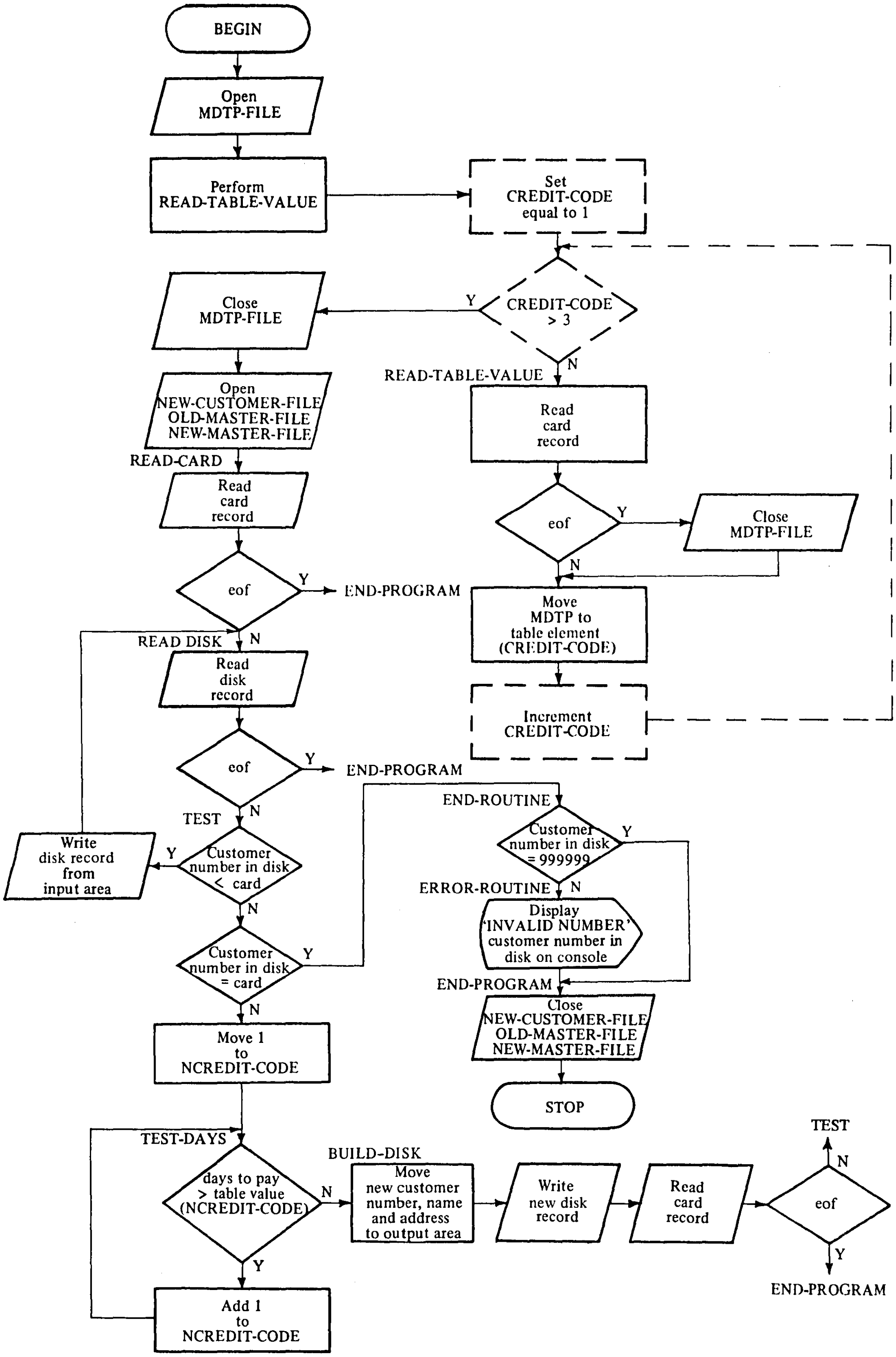
    b. NCREDIT-CODE (which is contained in the output area) as the subscript in the statement to test the days to pay.

* * *

Either (Less coding is required for b.)

---

18. Follow the program flow chart in Figure 135 and code the Procedure Division for MASTER-FILE-UPDATE.



BEGIN
Open MDTP-FILE
Perform READ-TABLE-VALUE
Set CREDIT-CODE equal to 1
CREDIT-CODE > 3
Y
N
Close MDTP-FILE
Open NEW-CUSTOMER-FILE OLD-MASTER-FILE NEW-MASTER-FILE
READ-TABLE-VALUE
Read card record
eof
Y
Close MDTP-FILE
N
Move MDTP to table element (CREDIT-CODE)
Increment CREDIT-CODE
READ-CARD
Read card record
eof
Y
END-PROGRAM
READ DISK
N
Read disk record
eof
Y
END-PROGRAM
TEST
N
Customer number in disk < card
Y
Write disk record from input area
N
Customer number in disk = card
Y
N
Move 1 to NCREDIT-CODE
END-ROUTINE
Customer number in disk = 999999
Y
ERROR-ROUTINE
N
Display 'INVALID NUMBER' customer number in disk on console
END-PROGRAM
Close NEW-CUSTOMER-FILE OLD-MASTER-FILE NEW-MASTER-FILE
STOP
TEST-DAYS
days to pay > table value (NCREDIT-CODE)
N
Y
Add 1 to NCREDIT-CODE
BUILD-DISK
Move new customer number, name and address to output area
Write new disk record
Read card record
eof
TEST
N
Y
END-PROGRAM

Figure 135

```
                                         *         *         *

0   0    1    1    2    2    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.
       INITIAL-ROUTINE.
           OPEN INPUT MDTP-FILE.
           PERFORM READ-TABLE-VALUE               (32)
               VARYING CREDIT-CODE
               FROM 1 BY 1
               UNTIL CREDIT-CODE GREATER THAN 3.
           CLOSE MDTP-FILE.
           OPEN INPUT NEW-CUSTOMER-FILE
               OLD-MASTER-FILE
               OUTPUT NEW-MASTER-FILE.
       READ-CARD.
           READ NEW-CUSTOMER-FILE
               AT END GO TO END-PROGRAM.
       READ-DISK.
           READ OLD-MASTER-FILE
               AT END GO TO END-PROGRAM.
       TEST.
           IF OCUSTOMER-NUMBER
               LESS THAN CUSTOMER-NUMBER
               WRITE NEW-MASTER-RECORD
                   FROM OLD-MASTER-RECORD
               GO TO READ-DISK.
           IF OCUSTOMER-NUMBER
               EQUAL TO CUSTOMER-NUMBER
               GO TO END-ROUTINE.
           MOVE 1 TO NCREDIT-CODE.
       TEST-DAYS.
           IF DAYS-TO-PAY GREATER THAN
               MAX-DAYS-TO-PAY (NCREDIT-CODE)      (13)
               ADD 1 TO NCREDIT-CODE
               GO TO TEST-DAYS.
       BUILD-DISK.
           MOVE CUSTOMER-NUMBER
               TO NCUSTOMER-NUMBER.
           MOVE CUSTOMER-NAME
               TO NCUSTOMER-NAME.
           MOVE CUSTOMER-ADDRESS
               TO NCUSTOMER-ADDRESS.
           WRITE NEW-MASTER-RECORD.
           READ NEW-CUSTOMER-FILE
               AT END GO TO END-PROGRAM.
           GO TO TEST.
       READ-TABLE-VALUE.                          (32)
           READ MDTP-FILE
               AT END CLOSE MDTP-FILE.
           MOVE MDTP                               (13)
               TO MAX-DAYS-TO-PAY (CREDIT-CODE).
       END-ROUTINE.
           IF OCUSTOMER-NUMBER
               EQUAL TO 999999
               GO TO END-PROGRAM.
               ELSE GO TO ERROR-ROUTINE.
       ERROR-ROUTINE.
           DISPLAY 'INVALID NUMBER'
               OCUSTOMER-NUMBER UPON CONSOLE.
       END-PROGRAM.
           CLOSE NEW-CUSTOMER-FILE
               OLD-MASTER-FILE NEW-MASTER-FILE.
           STOP RUN.

------------------------------------------------------------------------
```

SUMMARY:

You have now completed Lesson 28 in which you have learned to define variables whose values will be tables of data, to assign values to table elements, and to use tables in two ways. In coding a program to update an employee master file you used subscripts to specify table values to be used in processing other data. Then in a program to update a customer master file you tested table values to determine the subscript of the value that satisfied the specified condition.

END OF LESSON 28

LESSON 29

LESSON 29 - PROCESSING WITH INDEXES • CALLING SUBPROGRAMS

INTRODUCTION

In this lesson you will learn to use a variable to determine the number of table elements for a table whose size will vary from one execution to the next. You will modify the billing procedure that you coded to include additional processing using tables.

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

INDEXED BY option of the OCCURS clause
SET statement
CALL statement
READY statement
SET statement
ENTER statement.

This lesson will require approximately one hour.

1.

SALE-TABLE	
SALE-ITEM(1)	A20933
SALE-ITEM(2)	A21445
SALE-ITEM(3)	A22222
SALE-ITEM(4)	A23762
SALE-ITEM(5)	A25333
SALE-ITEM(6)	A26443
SALE-ITEM(7)	A29547
SALE-ITEM(8)	J33166
SALE-ITEM(9)	J24788
SALE-ITEM(10)	J67011

The subscript of each table element is called an occurrence number of a table. In SALE-TABLE:

a. the highest occurrence number is 10 .

b. occurrence number 8 corresponds to the value A29547.

* * *

a

----

2. The number that is specified in the OCCURS clause is the same as the highest:

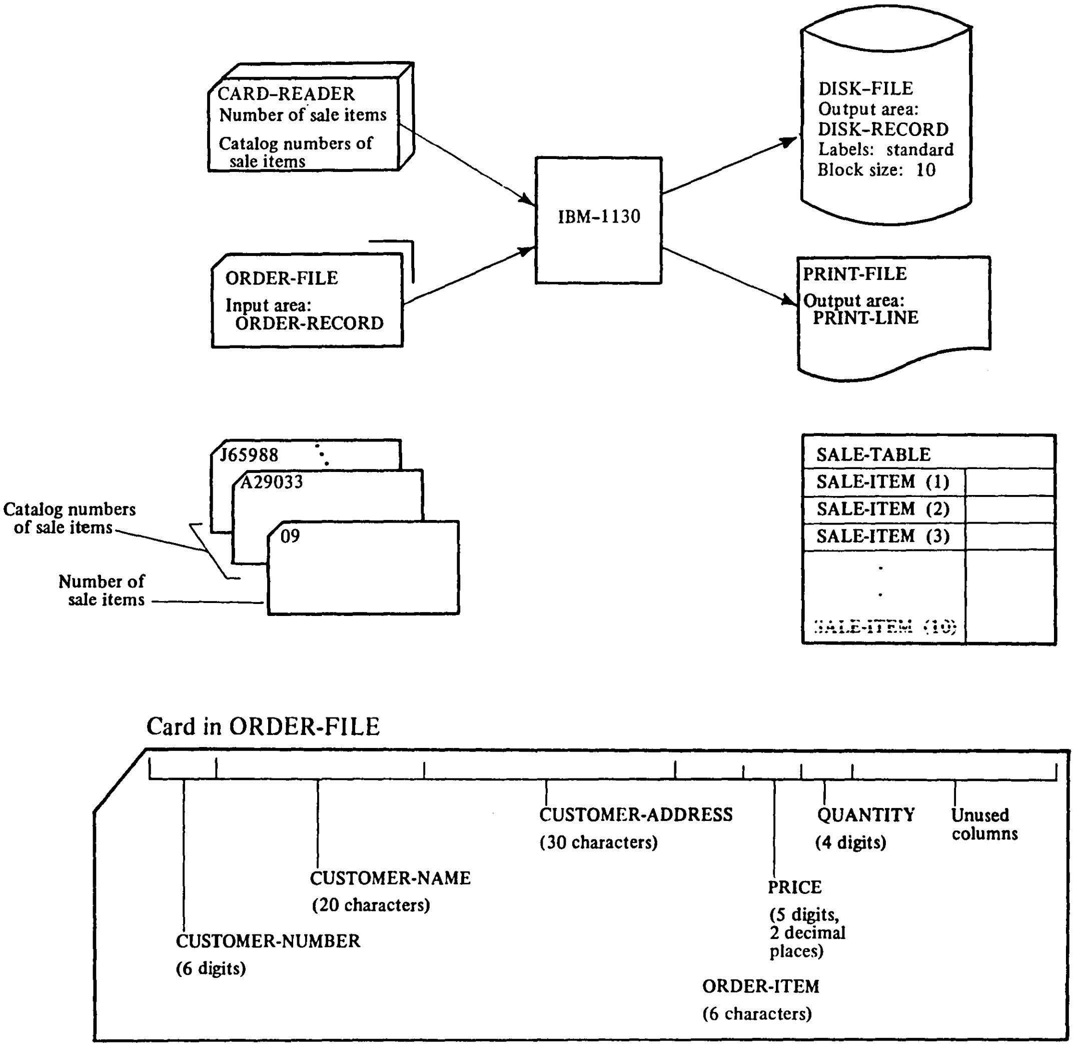
a. subscript for the table.

b. occurrence number for the table.

* * *

Both

```
0   0    1    1    2    2    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  TABLE-SIZE PIC 999.
        01  TERRITORY-TABLE.
            02 TERRITORY-SALESMEN
               OCCURS 4 TIMES PIC 999.
        01  SALES-TABLE.
            02 SALES
               OCCURS 200 TIMES
               PIC 9(5)V99.
```



Orders received by the mail-order wholesale house are punched into cards in ORDER-FILE as shown above. As an order is processed, the program is to determine whether the item ordered is a sale item. If so the price is to be computed as 90 percent of the regular price before the amount of the order is computed. The catalog numbers of sale items which have been punched one per card as shown above are to be treated as values of SALE-TABLE. The number of sale items for any one month will also be punched into a card to determine the desired size of SALE-TABLE for that month. The maximum number of sale items for any one month is 10.

Figure 136

3. The problem described in Figure 136 is to use a table whose size may vary from one execution to the next. Read the problem description in Figure 136. Code the following entries for the billing program.

   1) Working-Storage Section entries to define the variable.

      TABLE-SIZE which is to contain the number of sale items,

      TABLE-VALUE to which a table value will be transmitted prior to being moved to a specific table element,

      SALE-TABLE whose size is to be determined by TABLE-SIZE, and SUBSCRIPT to be used as a subscript in referring to table elements.

   2) Procedure Division entries to accept the table size and the table values.

* * *

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..

       WORKING-STORAGE SECTION.
       77  TABLE-SIZE PIC 99.
       77  TABLE-VALUE PIC X(6).
       77  SUBSCRIPT PIC 99.
       01  SALE-TABLE.
           02 SALE-ITEM
               OCCURS 10 TIMES
               PIC X(6).
```

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.
       INITIAL-ROUTINE.
           ACCEPT TABLE-SIZE.
           PERFORM ACCEPT-TABLE-VALUE
               VARYING SUBSCRIPT
               FROM 1 BY 1
               UNTIL SUBSCRIPT GREATER THAN
                   TABLE-SIZE
                    .
                    .
                    .
       ACCEPT-TABLE-VALUE.
           ACCEPT TABLE-VALUE.
           MOVE TABLE-VALUE
               TO SALE-ITEM (SUBSCRIPT).
```

In the preceding lesson you coded a program in which you found a table value that satisfied a certain condition and then used the subscript of that element for subsequent processing. You used the IF sentence.

---

4. In the problem described in Figure 136 the test to determine whether the item ordered is a sale item can be coded using the IF sentence. The flow chart segment a in Figure 139 shows the logic for this test using the IF sentence. Refer to the first three divisions of the billing program in Figure 137 if necessary and code the Procedure Division entries for segment a. (Assume that a record has been read from ORDER-FILE and that the appropriate variables have been moved to COMP variables.)

```
0   0    1    1    2    2    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.
       FILE-CONTROL.
           SELECT ORDER-FILE
               ASSIGN TO RD-1442.
           SELECT DISK-FILE
               ASSIGN TO DF-1-600-X.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE-SECTION.
       FD  ORDER-FILE
           LABEL RECORDS ARE OMITTED.
       01  ORDER-RECORD.
           02 CUSTOMER-NUMBER PIC 9(6).
           02 CUSTOMER-NAME PIC X(20).
           02 CUSTOMER-ADDRESS PIC X(30).
           02 ORDER-ITEM PIC X(6).
           02 PRICE PIC 999V99.
           02 QUANTITY PIC 9(4).
           02 FILLER PIC X(9).
       FD  DISK-FILE
           LABEL RECORDS ARE STANDARD,
       01  DISK-RECORD.
           02 FILLER PIC X(56).
           02 ORDER-ITEM-T PIC X(6).
           02 PRICE-T PIC 999V99.
           02 QUANTITY-T PIC 9(4).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-LINE PIC X(121).
       WORKING-STORAGE SECTION.
       77  TABLE-SIZE PIC 99.
       77  SUBSCRIPT PIC 99.
       01  SALE-TABLE.
           02  SALE-ITEM
               OCCURS 10 TIMES
               PIC X(6).
       01  COMPUTATION-RECORD.
           02 PRICE-WS PIC 999V99.
           02 QUANTITY-WS PIC 9(4).
           02 ONE PIC 9 VALUE 1.
           02 NINETY PIC V99 VALUE .90.
           02 AMOUNT PIC 9(7)V99.
```

Figure 137

*   *   *

```
0    0    1    1    2    2    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 1 TO SUBSCRIPT.
    TEST-SALE-ITEMS.
        IF ORDER-ITEM EQUAL TO
            SALE-ITEM (SUBSCRIPT)
            COMPUTE PRICE-WS ROUNDED
                = PRICE-WS * NINETY
            ELSE
            IF SUBSCRIPT LESS THAN
                TABLE-SIZE
                ADD ONE TO SUBSCRIPT
                GO TO TEST-SALE-ITEMS.
        COMPUTE AMOUNT
            = PRICE-WS * QUANTITY-WS.
```

Rules for User-Supplied Portions of the OCCURS Clause

Format

```
OCCURS integer TIMES [DEPENDING ON identifier]

    [INDEXED BY index-name]
```

Rules

1. Integer must specify the maximum table size for any execution of the program.

2. Index-name is defined by its appearance in INDEXED BY option.

3. Values of index-name are relative addresses that correspond to occurrence numbers of the table.

Figure 138

5. When you wish to look for a table element whose value satisfies a certain condition and take certain action when the value is found, you can use the INDEXED BY option of the OCCURS clause. According to Figure 138 index-name:

   a. must be defined with a picture of 9's.

   b. is defined by its appearance in the INDEXED BY option.

*   *   *

b

---

6. Figure 134 states that index-name has:

   a. values that correspond to occurrence numbers of the table.

   b. data values.

   c. values that are relative addresses.

* * *

a,c

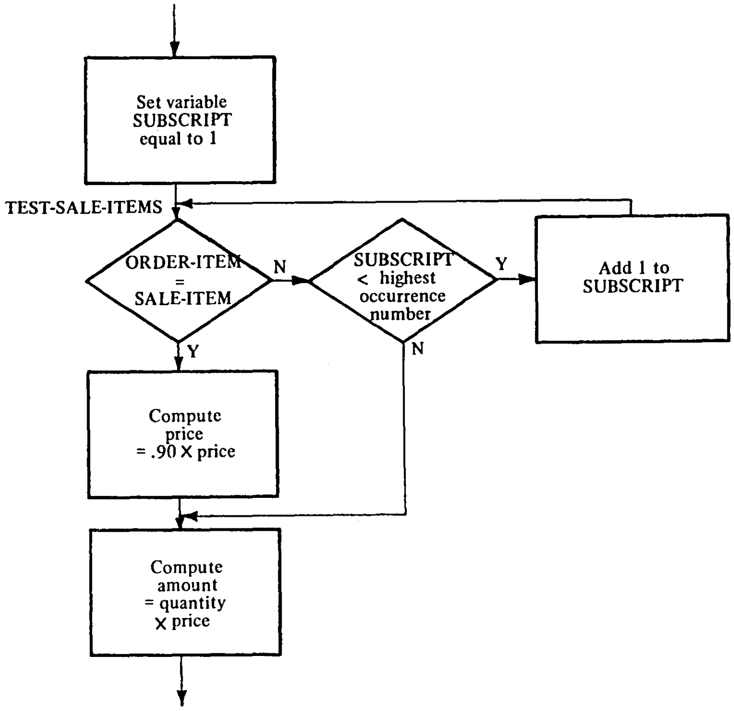


Figure 139

7.

```
                     ⎧ index-name-2 ⎫
SET index-name-1 TO  ⎨ identifier   ⎬
                     ⎩ literal      ⎭
```

You coded a MOVE statement for the first step in segment a of Figure 139. A MOVE statement is used to assign a value to a data variable. It cannot be used, however, to assign a value to an index variable. The SET statement must be used instead. According to the format shown above, the first step in segment a of Figure 139 could be coded as:

a. SET S TO 1.

b. SET S TO ONE.

* * *

Either
(In Figure 137 ONE was assigned a value of 1 in the Data Division.)

8.

```
                                      ┌UP BY   ┐ ┌identifier-4┐
SET index-name-4 [index-name-5]...   <         > <            >
                                      └DOWN BY ┘ └literal-2   ┘
```

When the preceeding format is used the contents of index-name-4 (and index-name-5 if present, etc.) are incremented (UP BY) or decremented (DOWN BY) by that which corresponds to the number represented by identifier-4 or literal-2.

SET I UP BY 1.

The above statement will increase index I by one.

SET I DOWN BY 1.

The above statement will decrease index I by one.

---

9. Code the record description entry for SALE-TABLE in Figure 136 so that a PERFORM statement can be used for the test of sale items. (Specify an index for the table.)

* * *

```
0   0    1    1    2    2    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  SALE-TABLE.
               02 SALE-ITEM
                   OCCURS 10 TIMES
                   INDEXED BY S PIC X(6).


               SET S TO 1.
               PERFORM SEARCH-SALES-ITEM
                       VARYING S FROM 1 BY 1
                       UNTIL S GREATER THAN 10.


           SEARCH-SALES-ITEM.
               IF ORDER-ITEM EQUAL TO SALES-ITEM(S)
               COMPUTE PRICE-WS ROUNDED
                   = PRICE-WS * NINETY.
               COMPUTE AMOUNT
                   = PRICE-WS * QUANTITY-WS.
```

---

10. SET identifier TO index-name

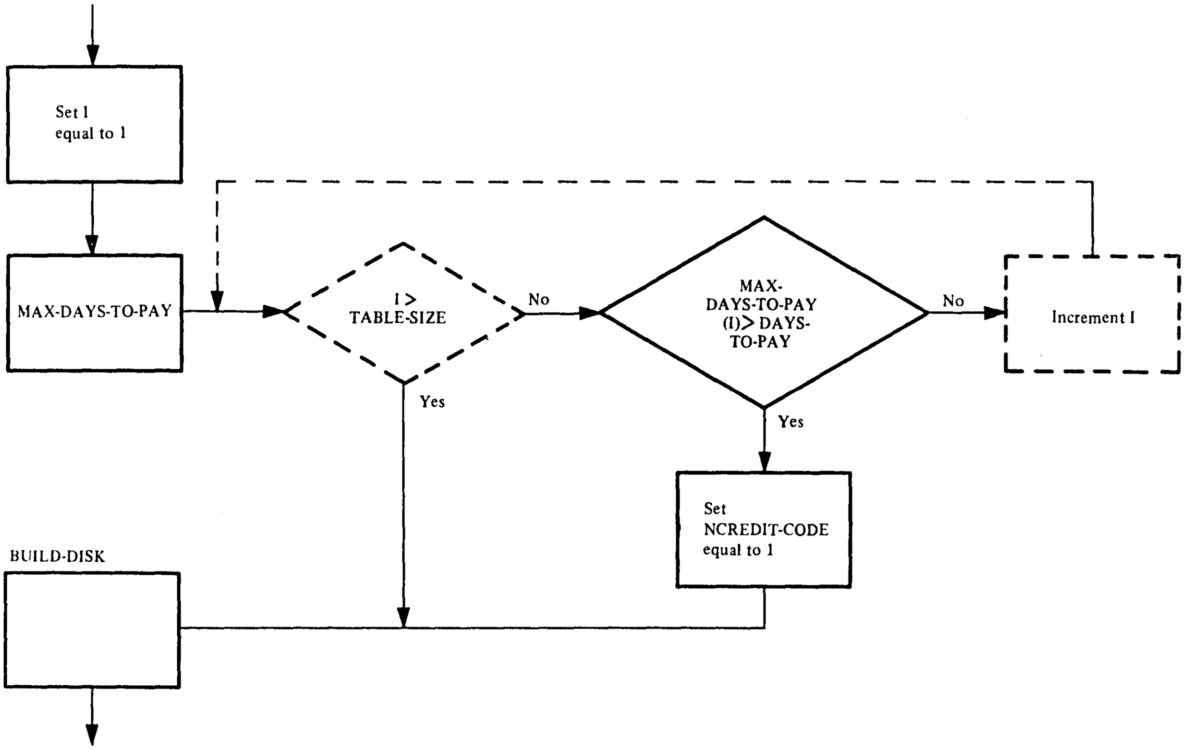


Figure 140

In the preceding lesson you coded a program in which a new customer credit code was determined by the number of days elapsed before the customer paid. A portion of the flow chart for that program has been modified to show how an IF sentence could have been used in the program. This logic would require the SET statement format shown above. Assume that the index I has been specified in the INDEXED BY option of the OCCURS clause for MAX-DAYS-TO-PAY and code the Procedure Division entries for the segment shown above using the IF sentence.

* * *

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

       SEARCH-TABLE.
           SET I TO 1.
           IF MAX-DAYS-TO-PAY (I)
           NOT LESS THAN DAYS-TO-PAY
           SET NCREDIT-CODE TO I
           ELSE GO TO INCR-TABLE.
           GO TO BUILD-TABLE.
       INCR-TABLE.
           IF I NOT EQUAL TO TABLE-SIZE
               SET I UP BY 1
               ELSE GO TO FULL-TABLE.
           GO TO SEARCH-TABLE.
                .
                .
                .
       BUILD-TABLE.
                .
                .
                .
       FULL-TABLE.
           DISPLAY 'DAYS-TO-PAY NOT IN TABLE'.
                .
                .
                .
```

11.

```
     ┌index-name-1 [index-name-2]...┐       ┌index-name-3┐
SET  ┤                              ├  TO  ┤identifier-3├
     └identifier-1 [identifier-2]...┘       └literal-1   ┘
```

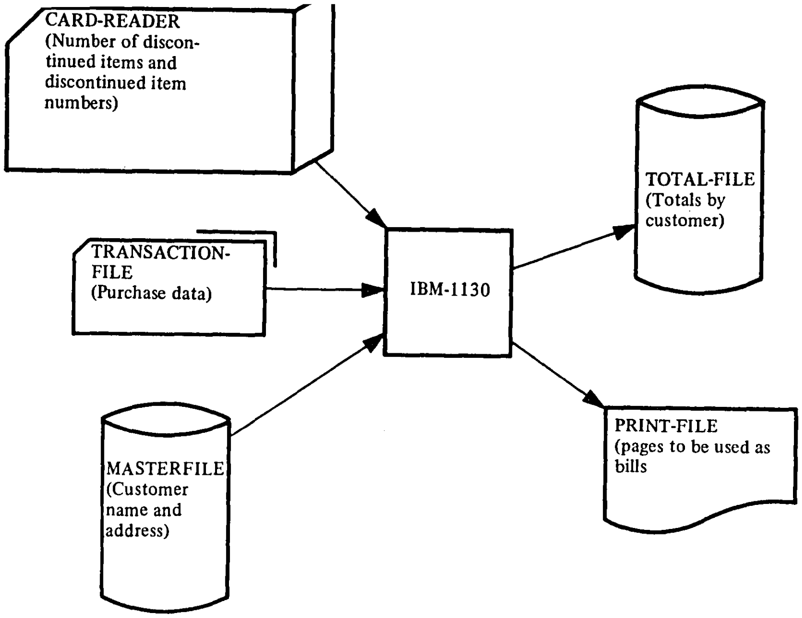
The SET statement format above shows that:

a. separate SET statements would be required to assign the value 1 to both I and NCREDIT-CODE.

b. the value 1 could be assigned to I and NCREDIT-CODE in a single statement.

* * *

b


CARD-READER
(Number of discontinued items and discontinued item numbers)
TRANSACTION-FILE
(Purchase data)
MASTERFILE
(Customer name and address)
IBM-1130
TOTAL-FILE
(Totals by customer)
PRINT-FILE
(pages to be used as bills

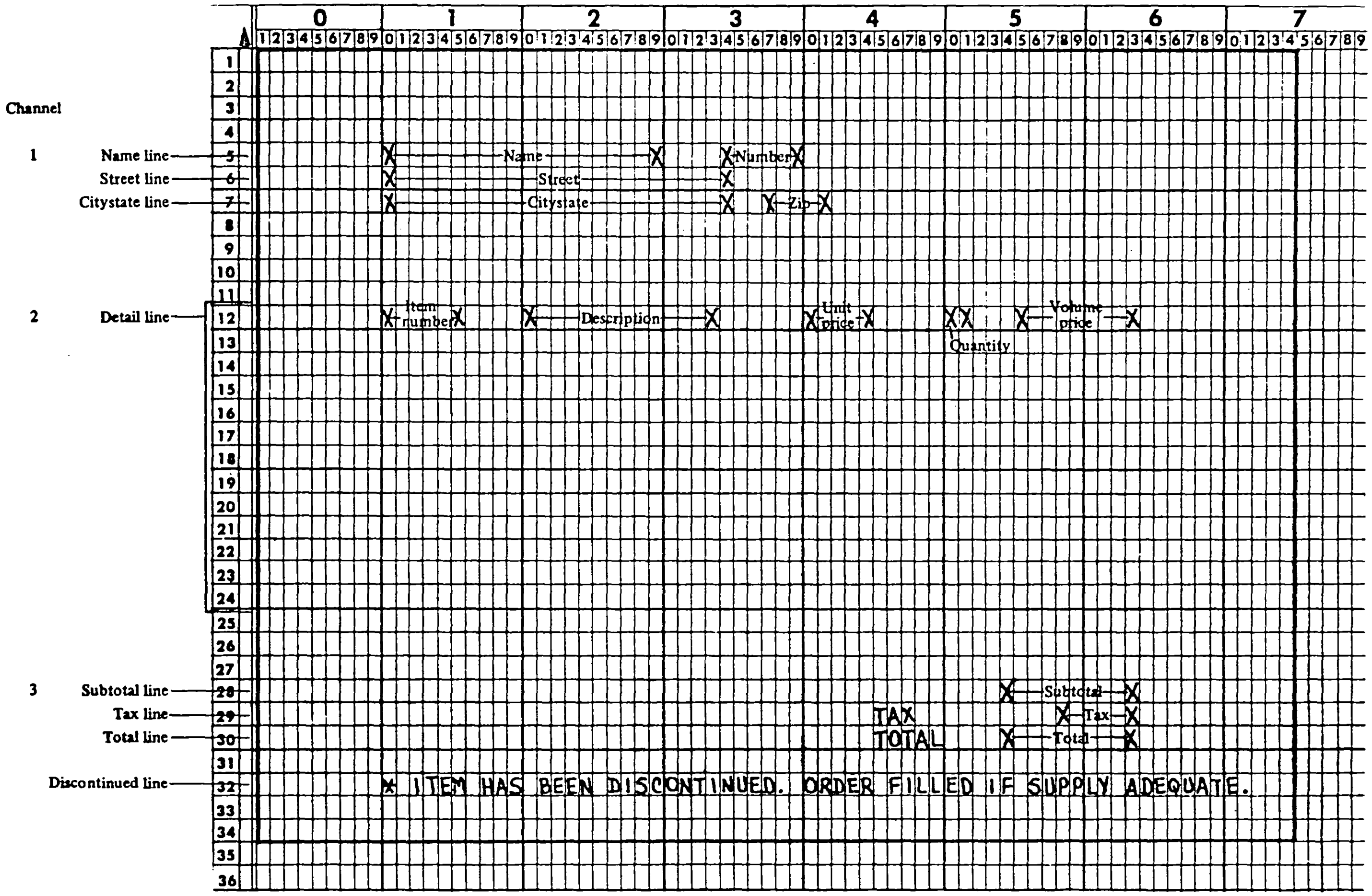
The billing procedure previously done is to be modified to include a text for discontinued items. The item number for each discontinued item has been punched into the first six columns of a card and is to be assigned to an element of the table shown below:

(Index D)

DISCONTINUED-TABLE	
DISCONTINUED-ITEM-NUMBER (1)	A17080
DISCONTINUED-ITEM-NUMBER (2)	B92641
. . . . .	

If the item number in a transaction card matches a number in the discontinued table, an asterisk is to be printed in place of QUANTITY and blanks are to appear in VOLUME-PRICE. In addition, when all transactions for customer have been processed, a note is to be printed below the total line. The manufacturer has produced numerous designs and over a period of years discontinued items may number several hundred. The Printer Spacing Chart in Figure 142 shows the desired format, and the program flowchart in Figure 143 shows the necessary modifications in logic. The coding for the first three divisions is given in Figure 144.

Figure 141



Channel
1 Name line
Street line
Citystate line
2 Detail line
3 Subtotal line
Tax line
Total line
Discontinued line
Name
Number
Street
Citystate
Zip
Item number
Description
Unit price
Quantity
Volume price
TAX
TOTAL
Subtotal
Tax
Total
* ITEM HAS BEEN DISCONTINUED. ORDER FILLED IF SUPPLY ADEQUATE.

Figure 142

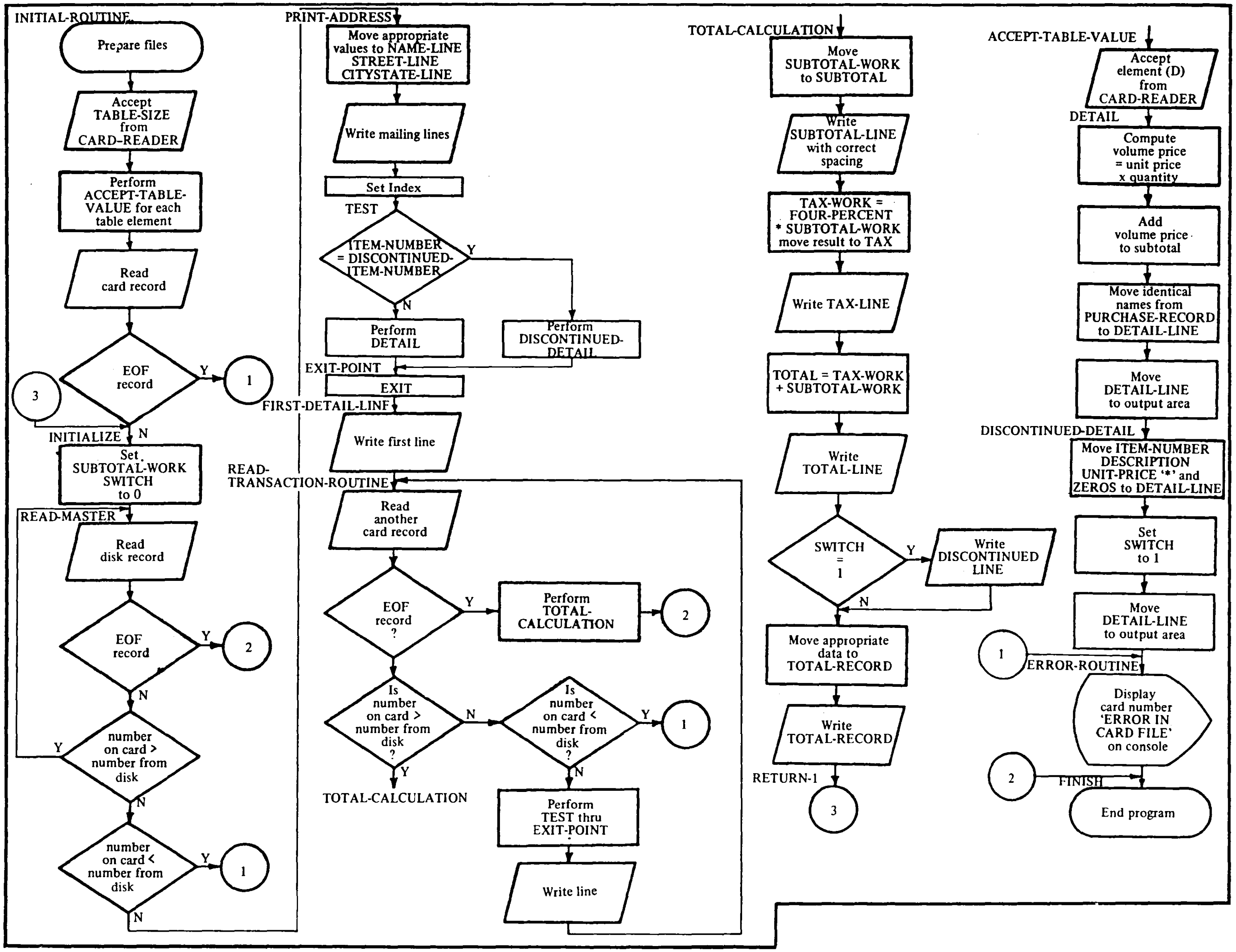

INITIAL-ROUTINE
Prepare files
Accept TABLE-SIZE from CARD-READER
Perform ACCEPT-TABLE-VALUE for each table element
Read card record
EOF record
Y
1
3
INITIALIZE
N
Set SUBTOTAL-WORK SWITCH to 0
READ-MASTER
Read disk record
EOF record
Y
2
N
number on card > number from disk
Y
N
number on card < number from disk
Y
1
N
PRINT-ADDRESS
Move appropriate values to NAME-LINE STREET-LINE CITYSTATE-LINE
Write mailing lines
Set Index
TEST
ITEM-NUMBER = DISCONTINUED-ITEM-NUMBER
Y
N
Perform DETAIL
Perform DISCONTINUED-DETAIL
EXIT-POINT
EXIT
FIRST-DETAIL-LINE
Write first line
READ-TRANSACTION-ROUTINE
Read another card record
EOF record ?
Y
Perform TOTAL-CALCULATION
2
Is number on card > number from disk ?
N
Y
TOTAL-CALCULATION
Is number on card < number from disk ?
Y
1
N
Perform TEST thru EXIT-POINT
Write line
TOTAL-CALCULATION
Move SUBTOTAL-WORK to SUBTOTAL
Write SUBTOTAL-LINE with correct spacing
TAX-WORK = FOUR-PERCENT * SUBTOTAL-WORK move result to TAX
Write TAX-LINE
TOTAL = TAX-WORK + SUBTOTAL-WORK
Write TOTAL-LINE
SWITCH = 1
Y
Write DISCONTINUED LINE
N
Move appropriate data to TOTAL-RECORD
Write TOTAL-RECORD
RETURN-1
3
ACCEPT-TABLE-VALUE
Accept element (D) from CARD-READER
DETAIL
Compute volume price = unit price x quantity
Add volume price to subtotal
Move identical names from PURCHASE-RECORD to DETAIL-LINE
Move DETAIL-LINE to output area
DISCONTINUED-DETAIL
Move ITEM-NUMBER DESCRIPTION UNIT-PRICE '*' and ZEROS to DETAIL-LINE
Set SWITCH to 1
Move DETAIL-LINE to output area
1
ERROR-ROUTINE
Display card number 'ERROR IN CARD FILE' on console
2
FINISH
End program

Figure 143

12. Read the problem description in Figure 141. Figure 144 shows the Data Division which has been modified to provide COMP variables to increase program efficiency. The picture character Z has been specified for VOLUME-PRICE so that blanks will appear on the invoice if the order cannot be filled. The variable DISCONTINUED-LINE has also been defined and assigned a value to be printed on some of the bills. Complete the Data Division by coding a data description entry for TABLE-SIZE which will determine the size of the table, TABLE-VALUE to which a table value will be transmitted prior to being moved to a specific table element, and a record description entry for the table illustrated in Figure 141.

* * *

```
0   0    1    1    2    2    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  TABLE-SIZE PIC 9999.
       77  TABLE-VALUE PIC X(6).
       01  DISCONTINUED-TABLE.
           12 DISCONTINUED-ITEM-NUMBER
               OCCURS 1000 TIMES
               INDEXED BY D                        (4)
               PIC X(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..

       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-1-800-X.
           SELECT TRANSACTION-FILE
               ASSIGN TO RD-1442.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
           SELECT TOTAL-FILE
               ASSIGN TO DF-2-900-X.
       DATA DIVISION.
       FILE SECTION.
       FD  MASTER-FILE
           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).
               03  CITY-STATE PIC X(15).
           02  FILLER PIC X(24).
       FD  TRANSACTION-FILE
           LABEL RECORDS ARE OMITTED.
       01  PURCHASE-RECORD.
           02  CUSTOMERNUMBER 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 X(46).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  BILL PIC X(75).
       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  SUBTOTAL-WORK-SWITCH PIC X.
       01  COMPUTATIONAL-RECORD USAGE COMP.
           02  UNIT-PRICE-WORK PIC 999V99.
           02  QUANTITY-WORK PIC 99.
           02  VOLUME-PRICE-WORK PIC 9999V99.
           02  SUBTOTAL-WORK PIC 99999V99.
           02  FOUR-PERCENT PIC V99 VALUE .04.
           02  TAX-WORK PIC 999V99.
           02  TOTAL-WORK PIC 99999V99.
```

```
01  NAME-LINE.
    02  FILLER PIC X(10) VALUE SPACES.
    02  NAME PIC X(20).
    02  FILLER PIC X(4) VALUE SPACES.
    02  CUSTOMER-NUMBER PIC X(6).
    02  FILLER PIC X(35) VALUE SPACES.


01  STREET-LINE.
    02  FILLER PIC X(10) VALUE SPACES.
    02  STREET PIC X(25).
    02  FILLER PIC X(40) VALUE SPACES.


01  CITYSTATE-LINE.
    02  FILLER PIC X(10) VALUE SPACES.
    02  CITYSTATE PIC X(25).
    02  FILLER PIC XX VALUE SPACES.
    02  ZIP PIC X(5).
    02  FILLER PIC X(33) VALUE SPACES.


01  DETAIL-LINE.
    02  FILLER PIC X(10) VALUE SPACES.
    02  ITEM-NUMBER PIC X(6).
    02  FILLER PIC X(4) VALUE SPACES.
    02  DESCRIPTION PIC X(15).
    02  FILLER PIC X(5) VALUE SPACES.
    02  UNIT-PRICE PIC 999.99.
    02  FILLER PIC XXXX VALUE SPACES.
    02  QUANTITY PIC XX.
    02  FILLER PIC X(4) VALUE SPACES.
    02  VOLUME-PRICE PIC Z,ZZZ.ZZ.
    02  FILLER PIC X(11) VALUE SPACES.


01  SUBTOTAL-LINE.
    02  FILLER PIC X(54) VALUE SPACES.
    02  SUBTOTAL PIC $$$,$99.99.
    02  FILLER PIC X(11) VALUE SPACES.


01  TAX-LINE.
    02  FILLER PIC X(45) VALUE SPACES.
    02  CONSTANT1 PIC XXX VALUE 'TAX'.
    02  FILLER PIC X(8) VALUE SPACES.
    02  TAX PIC 999.99.
    02  FILLER PIC X(11) VALUE SPACES.


01  TOTAL-LINE.
    02  FILLER PIC X(45) VALUE SPACES.
    02  CONSTANT2 PIC X(5) VALUE 'TOTAL'.
    02  FILLER PIC X(4) VALUE SPACES.
    02  TOTAL PIC $$$,$99.99.
    02  FILLER PIC X(11) VALUE SPACES.


01  DISCONTINUED-LINE.
    02  FILLER PIC X(10) VALUE SPACES.
    02  CONSTANT3 PIC X(31) VALUE
        '* ITEM HAS BEEN DISCONTINUED.  '.
    02  CONSTANT4 PIC X(32) VALUE
        'ORDER FILLED IF SUPPLY ADEQUATE.'.
    02  FILLER PIC X(2) VALUE SPACES.
```

Figure 144

```
0   0    1    1    2    2    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 CUSTOMERNUMBER OF PURCHASE RECORD TO
               CUSTOMERNUMBER 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-PRICES OF PURCHASE-RECORD
               TO UNIT-PRICE OF DETAIL-LINE.
           MOVE QUANTITY OF PURCHASE-RECORD TO
               QUANTITY OF DETAIL-LINE.
           MOVE STREET TO STREET-O.
           MOVE CITY-STATE TO CITY-STATE-O.
           WRITE BILL FROM NAME-LINE
               AFTER ADVANCING TO-NAME-LINE.
           WRITE BILL FROM STREET-LINE
               AFTER ADVANCING 1 LINE.
           WRITE BILL FROM CITYSTATE-LINE
               AFTER ADVANCING 1 LINE.
       PRINT-DETAIL.
           COMPUTE VOLUME-PRICE-WORK =
               UNIT-PRICE OF PURCHASE-RECORD
               * QUANTITY OF PURCHASE-RECORD.
           ADD VOLUME-PRICE-WORK
               TO SUBTOTAL-WORK.
               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 DETAIL-LINE TO BILL.
       FIRST-DETAIL-LINE.
           WRITE BILL
               AFTER ADVANCING TO-DETAIL-LINE.
       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
        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 CUSTOMERNUMBER
        'ERROR IN CARD FILE'.
FINISH.
    CLOSE MASTER-FILE TRANSACTION-FILE
        PRINT-FILE TOTAL-FILE.
    STOP RUN.
```

Figure 145

13. The Procedure Division for the problem is shown in Figure 145. Follow the program flow chart in Figure 143 and code the missing portions of the Procedure Division below to incorporate table handling, PERFORM statements, and other entries as indicated.

```
0   0    1    1    2    2    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.

       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 TRANSACTION-FILE
               AT END GO TO ERROR-ROUTINE.
       INITIALIZE.
           MOVE ZEROS TO SUBTOTAL-WORK.

       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-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 TO STREET-O.
           MOVE CITY-STATE TO CITY-STATE-O.
           WRITE BILL FROM NAME-LINE
               AFTER ADVANCING TO-NAME-LINE.
           WRITE BILL FROM STREET-LINE
               AFTER ADVANCING 1 LINE.
           WRITE BILL FROM CITYSTATE-LINE
               AFTER ADVANCING 1 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..

       FIRST-DETAIL-LINE.
           WRITE BILL
               AFTER ADVANCING TO-DETAIL-LINE.
       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.
```

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 BILL
               AFTER ADVANCING 1 LINE.
           GO TO READ-TRANSACTION-ROUTINE.
       TOTAL-CALCULATION.
           MOVE SUBTOTAL-WORK TO SUBTOTAL.
           WRITE BILL FROM SUBTOTAL-LINE
               AFTER ADVANCING TO-SUBTOTAL-LINE.
           COMPUTE TAX-WORK = SUBTOTAL-WORK.
           MOVE TAX-WORK TO TAX.
           WRITE BILL FROM TAX-LINE
               AFTER ADVANCING 1 LINE.
           COMPUTE TOTAL-WORK = TAX-WORK
               + SUBTOTAL-WORK.
           WRITE BILL FROM TOTAL-LINE
               AFTER ADVANCING 1 LINE.
```

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..

           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.
```

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..

       DETAIL.
           COMPUTE VOLUME-PRICE-WORK =
               UNIT-PRICE-WORK * QUANTITY-WORK.
           ADD VOLUME-PRICE-WORK
               TO SUBTOTAL-WORK.
           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 VOLUME-PRICE OF PURCHASE-RECORD
               TO VOLUME-PRICE OF DETAIL-LINE.
           MOVE DETAIL-LINE TO BILL.
```

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..

       ERROR-ROUTINE.
           DISPLAY CUSTOMERNUMBER
               'ERROR IN CARD FILE'
               UPON CONSOLE.
       FINISH.
           CLOSE MASTER-FILE
               TRANSACTION-FILE
               PRINT-FILE TOTAL-FILE.
           STOP RUN.
```

* * *

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..

           ACCEPT TABLE-SIZE.
           PERFORM ACCEPT-TABLE-VALUE
               VARYING D FROM 1 BY 1
               UNTIL D GREATER THAN TABLE-SIZE.
```

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..

           MOVE ZERO TO SWITCH.
```

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..

           SET D TO 1.
       TEST.
           IF ITEM-NUMBER EQUAL TO
               DISCONTINUED-ITEM-NUMBER (D)
               GO TO TEST1.
           IF D IS LESS THAN 1000
               ADD 1 TO D
               GO TO TEST
           ELSE PERFORM DETAIL.
           GO TO EXIT POINT
       TEST1.
           PERFORM DISCONTINUED-DETAIL.

       EXIT-POINT.
           EXIT.
```

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..

           PERFORM TEST THRU EXIT-POINT.
```

5)

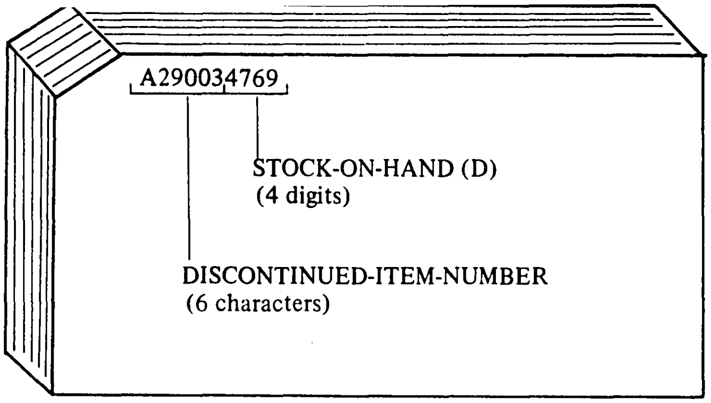
```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           IF SWITCH EQUAL TO 1
               WRITE BILL
                   FROM DISCONTINUED-LINE
                   AFTER ADVANCING 2 LINES.
```

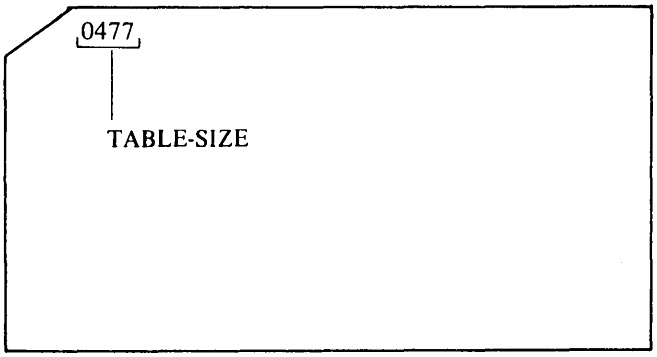
6)

```
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..

       ACCEPT-TABLE-VALUE.
           ACCEPT TABLE-VALUE.
           MOVE TABLE-VALUE
               TO DISCONTINUED-ITEM-NUMBER (D).
```

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..

       DISCONTINUED-DETAIL.
           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 '*' TO QUANTITY OF DETAIL-LINE.
           MOVE ZEROS TO VOLUME-PRICE
               OF DETAIL-LINE.
           MOVE 1 TO SWITCH.
           MOVE DETAIL-LINE TO BILL.
```





DISCONTINUED-TABLE		
TABLE-VALUE (1)	A29003 DISCONTINUED-ITEM-NUMBER (1)	4769 STOCK-ON-HAND (1)
TABLE-VALUE (2)	DISCONTINUED-ITEM-NUMBER (2)	STOCK-ON-HAND (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  DISCONTINUED-TABLE.
           02 TABLE-VALUE
               OCCURS 1000 TIMES
               INDEXED BY D.
              03 DISCONTINUED-ITEM-NUMBER PIC X(6).
              03 STOCK-ON-HAND PIC 9(4).

           ACCEPT TABLE-SIZE.
           PERFORM ACCEPT-TABLE-VALUE
               VARYING D FROM 1 BY 1
               UNTIL D GREATER THAN TABLE-SIZE.

       ACCEPT-TABLE-VALUE.
           ACCEPT TABLE-VALUE (D).
```

Figure 146

14. Although production of some items has been discontinued, orders for these items are to be filled until the stock on hand is depleted. The data processing center has been asked to incorporate a test of stock on hand into its billing program. Figure 146 shows the way data is to be supplied to the programmer as well as the table that the programmer might set up for his program. Each table element can consist of two (or more) elementary variables. For example, the element TABLE-VALUE (1) consists of the two elementary variables DISCONTINUED-ITEM-NUMBER (1) and STOCK-ON-HAND (1). The record description entry for the table is included in the figure along with the entries to assign values to the table elements. The paragraphs TEST through EXIT-POINT that you coded in the preceding frame are to be modified to provide the logic shown in Figure 147. Code the necessary entries.

* * *

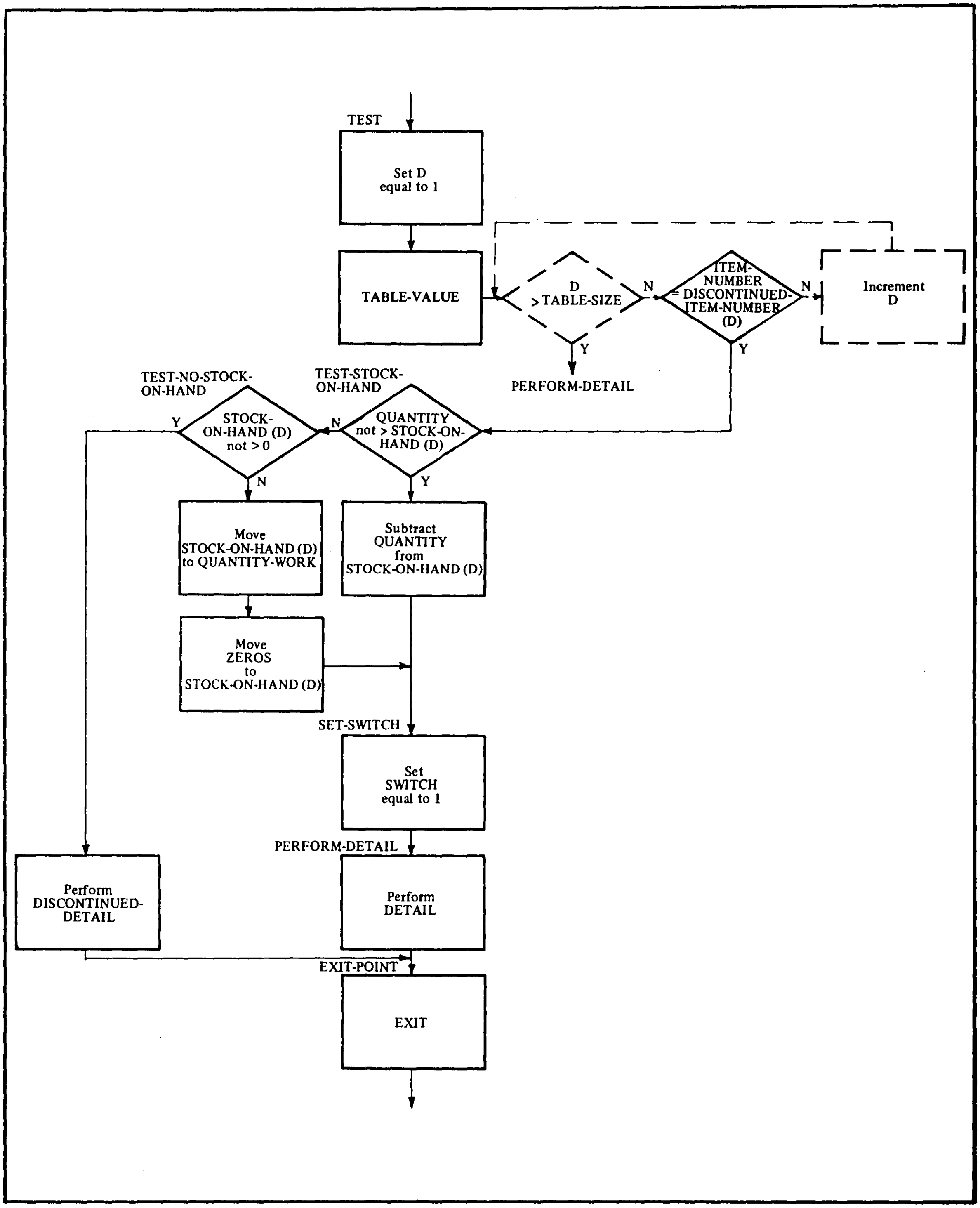

TEST
Set D equal to 1
TABLE-VALUE
D > TABLE-SIZE
N
Y
PERFORM-DETAIL
ITEM-NUMBER = DISCONTINUED-ITEM-NUMBER (D)
N
Y
Increment D
TEST-NO-STOCK-ON-HAND
TEST-STOCK-ON-HAND
STOCK-ON-HAND (D) not > 0
Y
N
QUANTITY not > STOCK-ON-HAND (D)
N
Y
Move STOCK-ON-HAND (D) to QUANTITY-WORK
Subtract QUANTITY from STOCK-ON-HAND (D)
Move ZEROS to STOCK-ON-HAND (D)
SET-SWITCH
Set SWITCH equal to 1
PERFORM-DETAIL
Perform DISCONTINUED-DETAIL
Perform DETAIL
EXIT-POINT
EXIT

Figure 147

```
        TEST.
           IF ITEM-NUMBER EQUAL TO
           DISCONTINUED-ITEM-NUMBER (D)
           GO TO TEST-STOCK-ON-HAND.
           IF D IS LESS THAN 1000
           ADD 1 TO D.
           GO TO TEST.
           PERFORM DETAIL
       TEST-STOCK-ON-HAND.
           IF QUANTITY NOT GREATER THAN
               STOCK-ON-HAND (D)
               COMPUTE STOCK-ON-HAND (D)
                   = STOCK-ON-HAND (D)
                   - QUANTITY-WORK
               GO TO SET-SWITCH.
               ELSE GO TO TEST-NO-STOCK-ON-HAND.
       TEST-NO-STOCK-ON-HAND.
           IF STOCK-ON-HAND (D)
               NOT GREATER THAN ZERO
               PERFORM DISCONTINUED-DETAIL
               GO TO EXIT-POINT
               ELSE MOVE STOCK-ON-HAND (D)
                   TO QUANTITY-WORK
                   MOVE ZEROS
                       TO STOCK-ON-HAND (D).
       SET-SWITCH.
           MOVE 1 TO SWITCH.
       PERFORM-DETAIL.
           PERFORM DETAIL.
       EXIT-POINT.
           EXIT.
```

15.

The $\left\{\begin{array}{l}\text{READY}\\ \text{RESET}\end{array}\right\}$ TRACE statement is provided for program debugging.

It may appear anywhere in an IBM ANSI COBOL program.

After the READY TRACE statement is executed, each time execution of a paragraph or section begins, its external statement number is displayed on the printer.

The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

---

16. The ENTER statement serves only as documentation and is intended to provide a means of allowing the use of more than one language in the same program.

ENTER language-name [routine-name]

The ENTER statement is accepted as comments.

ENTER ASSEMBLER. CALL...

ENTER COBOL.

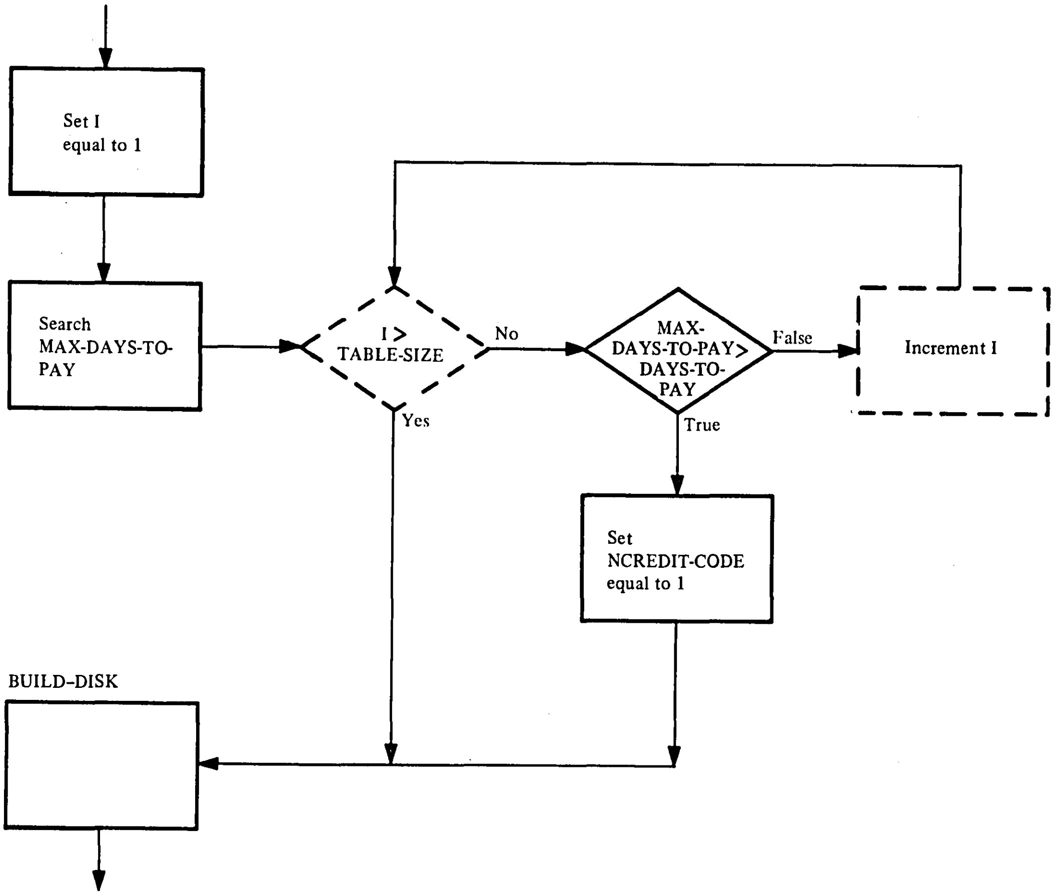


Figure 148

17. The CALL statement permits communications between a COBOL object program and one or more COBOL subprograms or other language subprograms.

Each of the operands in the USING option of the Procedure Division header must have been defined as a data item in the Linkage Section of the program in which this header occurs, and must have a level number of 01 or 77.

When the USING option is present, the object program operates as though each occurrence of identifier-1, identifier-2, etc., in the Procedure Division had been replaced by the corresponding identifier from the USING option in the CALL statement of the calling program. That is, corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional and not by name.

The following is an example of a calling program with the USING 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..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. CALLPROG.
           .
           .
           .
       DATA DIVISION.
           .
           .
           .
       WORKING-STORAGE SECTION.
       01  RECORD-1.
           03  SALARY          PICTURE S9(5)V99.
           03  RATE            PICTURE S9V99.
           03  HOURS           PICTURE S99V9.
           .
           .
           .
       PROCEDURE DIVISION.
           .
           .
           .
           CALL "SUBPROG" USING RECORD-1.
           .
           .
           .
```

The following is an example of a called subprogram associated with the preceding calling 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. SUBPROG.
           .
           .
           .
       DATA DIVISION.
           .
           .
           .
       LINKAGE SECTION.
       01  PAYREC.
           02  PAY                    PICTURE S9(5)V99.
           02  HOURLY-RATE            PICTURE S9V99.
           02  HOURS                  PICTURE S99V9.
           .
           .
           .
       PROCEDURE DIVISION USING PAYREC.
           .
           .
           .
           EXIT PROGRAM.
```

Processing begins in CALLPROG, which is the calling program. When the statement

```
CALL "SUBPROG" USING RECORD-1.
```

is executed, control is transferred to the first statement of the Procedure Division in SUBPROG, which is the called program. In the calling program, the operand of the USING option is identified as RECORD-1.

When SUBPROG receives control, the values within RECORD-1 are made available to SUBPROG; in SUBPROG, however, they are referred to as PAYREC. Note that the PICTURE clauses for the subfields of PAYREC (described in the Linkage Section of SUBPROG) are the same as those for RECORD-1.

When processing within SUBPROG reaches the EXIT PROGRAM statement, control is returned to CALLPROG at the statement immediately following the original CALL statement. Processing then continues in CALLPROG.

In any given execution of this program, if the values within RECORD-1 are changed between the time of the CALL and the return, the values stored at the time of the return will be the changed, not the original, values. If the programmer wishes to use the original values, then he must ensure that they have been saved.

SUMMARY

You have now completed Lesson 29 in which you have learned several additional COBOL language features that can be used in table processing. You have learned to use the DEPENDING ON option of the OCCURS clause to specify a variable whose value is to determine the size of a table for the current execution of the program. You have used the INDEXED BY option of the OCCURS clause to define a table index. You have used the SET statement to assign values to index variables.

END OF LESSON 29

LESSON 30

LESSON 30 - PAYROLL PROGRAM PROCESSING

INTRODUCTION

In this lesson you will combine a number of the language features and programming techniques that you have learned as you code a payroll program using card, disk, and printer files.

A COBOL language feature that you will learn to use in this lesson is:

REDEFINES clause

This lesson will require approximately three quarters of an hour.

1. The REDEFINES clause allows the programmer to use alternate descriptions of data for the same computer storage area. Figure 149 shows that the REDEFINES clause can be used to redefine:

   a. pictures.

   b. subdivision.

* * *

Either

---

2. Read the description of the effect of redefinition of a picture in the example in Figure 149. According to Figure 149, an effect of redefining a picture is that a value originally defined as one type of data can be treated as another:

   a. by moving the value to another variable.

   b. without moving the value.

* * *

b

Examples and Effects of the REDEFINES Clause

Used to redefine	Example	Effect
Picture	Ø1 A PIC 99. Ø1 B REDEFINES A PIC XX.	The value of A originally defined as numeric data is treated as alphanumeric data if referred to by B, without moving the value to B.
Subdivision	Ø1 SALARY-RECORD. Ø2 SOCIAL-SECURITY PIC 9(9). Ø2 SALARY PIC 999V99. Ø2 MONTH PIC 99. Ø1 WAGE-RECORD REDEFINES SALARY-RECORD. Ø2 SOCIAL-SECURITY PIC 9(9). Ø2 WAGE PIC 9V99, Ø2 HOURS PIC 99. Ø2 WEEK PIC 99.	Two kinds of records with different subdivision can be processed differently without moving one record or a portion of the record to another variable. Five digits for monthly salary in a salaried employee record can be referred to by SALARY or three digits for hourly pay and two for hours in an hourly paid employee record can be referred to by WAGE and HOURS, respectively, without moving the values to WAGE and HOURS.

Figure 149

3. Read the description of the effect of redefinition of subdivision in Figure 149. According to this explanation, an effect of redefining subdivision is that five digits could be treated as a single data item in one type of record and as two data items in a second type:

   a. without moving the values to another variable.

   b. by moving the values to another variable.

* * *

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..
       01  TABLE-AEDIT REDEFINES TABLE-ADISP...
```

According to Figure 149, in the entry above:

a. TABLE-AEDIT is the table originally defined.

b. the REDEFINES clause is specified for TABLE-ADISP.

* * *

Neither (The opposite is true.)

Summary of Data Description Clauses

level number $\left\{\begin{array}{l}\text{data-name-1}\\ \text{FILLER}\end{array}\right\}$ appropriate clauses

Clause Format	Purpose	Restrictions
REDEFINES data-name-2	Allows alternate descriptions of data to apply to the same storage area.	1. Must immediately follow data-name-1. 2. Level number of data-name-1 must be the same as that of data-name-2. 3. Level number must not be 88. 4. Data-name-1 must be in working storage. 5. Data-name-2 cannot contain an OCCURS clause and cannot be subordinate to a name for which an OCCURS clause is specified.
OCCURS integer TIMES [INDEXED BY index-name	Specifies table size.	May not be specified for a level 01 or level 77 name.
PICTURE is character-string	Gives form of numeric variable or editing requirements of elementary data items shown in figures 44 and 121.	Only characters in Figures 40 and 117 may be used in the character string.
USAGE IS $\left\{\begin{array}{l}\underline{\text{DISPLAY}}\\ \underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMP}}\end{array}\right\}$	Specifies the manner in which data is to be stored.	May be written at the group or elementary level.
VALUE IS literal	1. Defines the initial value of an item in working storage. 2. Gives the value associated with condition name.	1. In the Working-Storage Section may assign a value or a condition name. 2. In the File Section may be used for condition name only.

Figure 150

5. The restrictions in Figure 150 state that in a data description entry the REDEFINES clause:

   a. must immediately follow the data name.

   b. may follow any other clause.

*   *   *

a

----

6. According to Figure 150 the REDEFINES clause may be specified for:

   a. any level 01 variable.

   b. level 01 variables in working storage.

*   *   *

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..
              01  TABLE-AEDIT REDEFINES TABLE-ADISP...
```

Refer to Figure 150 and match the names from the entry above with the section(s) of the Data Division in which each may be defined.

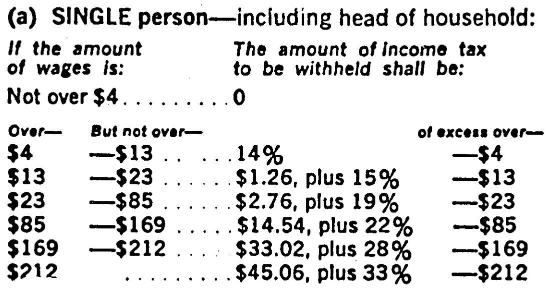
1) File Section	a. TABLE-AEDIT
2) Working-Storage Section	b. TABLE-ADISP

*   *   *

1) b
2) a, b

Percentage Method of Withholding

WEEKLY Payroll Period



(a) SINGLE person—including head of household:

If the amount of wages is:		*The amount of income tax to be withheld shall be:*	
Not over $4		0	
Over—	**But not over—**		**of excess over—**
$4	—$13	14%	—$4
$13	—$23	$1.26, plus 15%	—$13
$23	—$85	$2.76, plus 19%	—$23
$85	—$169	$14.54, plus 22%	—$85
$169	—$212	$33.02, plus 28%	—$169
$212		$45.06, plus 33%	—$212

SINGLE-DOLLAR-RECORD	
SINGLE-DOLLAR-1	000
SINGLE-DOLLAR-2	004
SINGLE-DOLLAR-3	013
SINGLE-DOLLAR-4	023
SINGLE-DOLLAR-5	085
SINGLE-DOLLAR-6	169
SINGLE-DOLLAR-7	212
SINGLE-DOLLAR-8	999

SINGLE-PERCENT-RECORD	
SINGLE-PERCENT-1	‸00
SINGLE-PERCENT-2	‸14
SINGLE-PERCENT-3	‸15
SINGLE-PERCENT-4	‸19
SINGLE-PERCENT-5	‸22
SINGLE-PERCENT-6	‸28
SINGLE-PERCENT-7	‸33
SINGLE-PERCENT-8	‸33

SINGLE-CONSTANT-RECORD	
SINGLE-CONSTANT-1	00‸00
SINGLE-CONSTANT-2	00‸00
SINGLE-CONSTANT-3	01‸26
SINGLE-CONSTANT-4	02‸76
SINGLE-CONSTANT-5	14‸54
SINGLE-CONSTANT-6	33‸02
SINGLE-CONSTANT-7	45‸06
SINGLE-CONSTANT-8	45‸06

SINGLE-DOLLAR-OVERLAY	
SINGLE-DOLLAR (1)	
SINGLE-DOLLAR (2)	
SINGLE-DOLLAR (3)	
SINGLE-DOLLAR (4)	
SINGLE-DOLLAR (5)	
SINGLE-DOLLAR (6)	
SINGLE-DOLLAR (7)	
SINGLE-DOLLAR (8)	

SINGLE-PERCENT-OVERLAY	
SINGLE-PERCENT (1)	
SINGLE-PERCENT (2)	
SINGLE-PERCENT (3)	
SINGLE-PERCENT (4)	
SINGLE-PERCENT (5)	
SINGLE-PERCENT (6)	
SINGLE-PERCENT (7)	
SINGLE-PERCENT (8)	

SINGLE-CONSTANT-OVERLAY	
SINGLE-CONSTANT (1)	
SINGLE-CONSTANT (2)	
SINGLE-CONSTANT (3)	
SINGLE-CONSTANT (4)	
SINGLE-CONSTANT (5)	
SINGLE-CONSTANT (6)	
SINGLE-CONSTANT (7)	
SINGLE-CONSTANT (8)	

The dollar amounts, percentages, and constants for computing federal income tax for a single person paid weekly can be set up as tables in a COBOL program. When taxable income is greater than dollar amount (n) but not greater than dollar amount (n+1) federal tax can be computed using the following formula.

[taxable - dollar amount(n)] X [percentage(n)]
+ excess constant(n)

A search statement can specify this test and computation provided an index is defined for each table. Since the table values are fairly stable and are used each week, specifying the values in the program itself will ensure correct table values for each execution. The values can be specified in VALUE clauses for unique variables within records of dollar amounts, percentages, and constants. Then the REDEFINES clause can be used to allow the overlay tables to apply to the values.

Figure 151

8. The REDEFINES clause can be used to allow a table defined with an index to apply to values that were assigned to variables within another record variable with value clauses. The REDEFINES clause can be used in this way for a portion of the payroll problem which is described in Figure 151. Read the problem description in this figure. Match the variables with the clauses that would be specified for them.

   1) VALUE clause

   2) OCCURS clause with the INDEXED BY option

   3) REDEFINES clause

   a. Elementary variables within
      SINGLE-DOLLAR-RECORD,
      SINGLE-PERCENT-RECORD,
      and
      SINGLE-CONSTANT-RECORD

   b. SINGLE-DOLLAR-RECORD,
      SINGLE-PERCENT-RECORD,
      and
      SINGLE-CONSTANT-RECORD

   c. Elementary variables within
      SINGLE-DOLLAR-OVERLAY,
      SINGLE-PERCENT-OVERLAY,
      and
      SINGLE-CONSTANT-OVERLAY

   d. SINGLE-DOLLAR-OVERLAY,
      SINGLE-PERCENT-OVERLAY,
      and
      SINGLE-CONSTANT-OVERLAY

* * *

1) a
2) c
3) d

---

9. Code record description entries for the record of dollar amounts and the dollar amount overlay table for use with Table Search statements. Specify the index DS.

* * *

```
0   0   1   1   2   2   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  SINGLE-DOLLAR-RECORD USAGE COMP.
                  02  SINGLE-DOLLAR-1 PIC 999
                      VALUE ZEROS.
                  02  SINGLE-DOLLAR-2 PIC 999
                      VALUE 004.
                  02  SINGLE-DOLLAR-3 PIC 999
                      VALUE 013.
                  02  SINGLE-DOLLAR-4 PIC 999
                      VALUE 023.
                  02  SINGLE-DOLLAR-5 PIC 999
                      VALUE 085.
                  02  SINGLE-DOLLAR-6 PIC 999
                      VALUE 169.
                  02  SINGLE-DOLLAR-7 PIC 999
                      VALUE 212.
                  02  SINGLE-DOLLAR-8 PIC 999
                      VALUE 999.
              01  SINGLE-DOLLAR-OVERLAY
                      REDEFINES SINGLE-DOLLAR-RECORD
                      USAGE COMP.
                  02  SINGLE-DOLLAR OCCURS 8 TIMES
                      INDEXED BY DS PIC 999.
```

SUMMARY

You have now completed Lesson 30 in which you learned the use of the REDEFINES clause.

END OF LESSON 30

LESSON 31

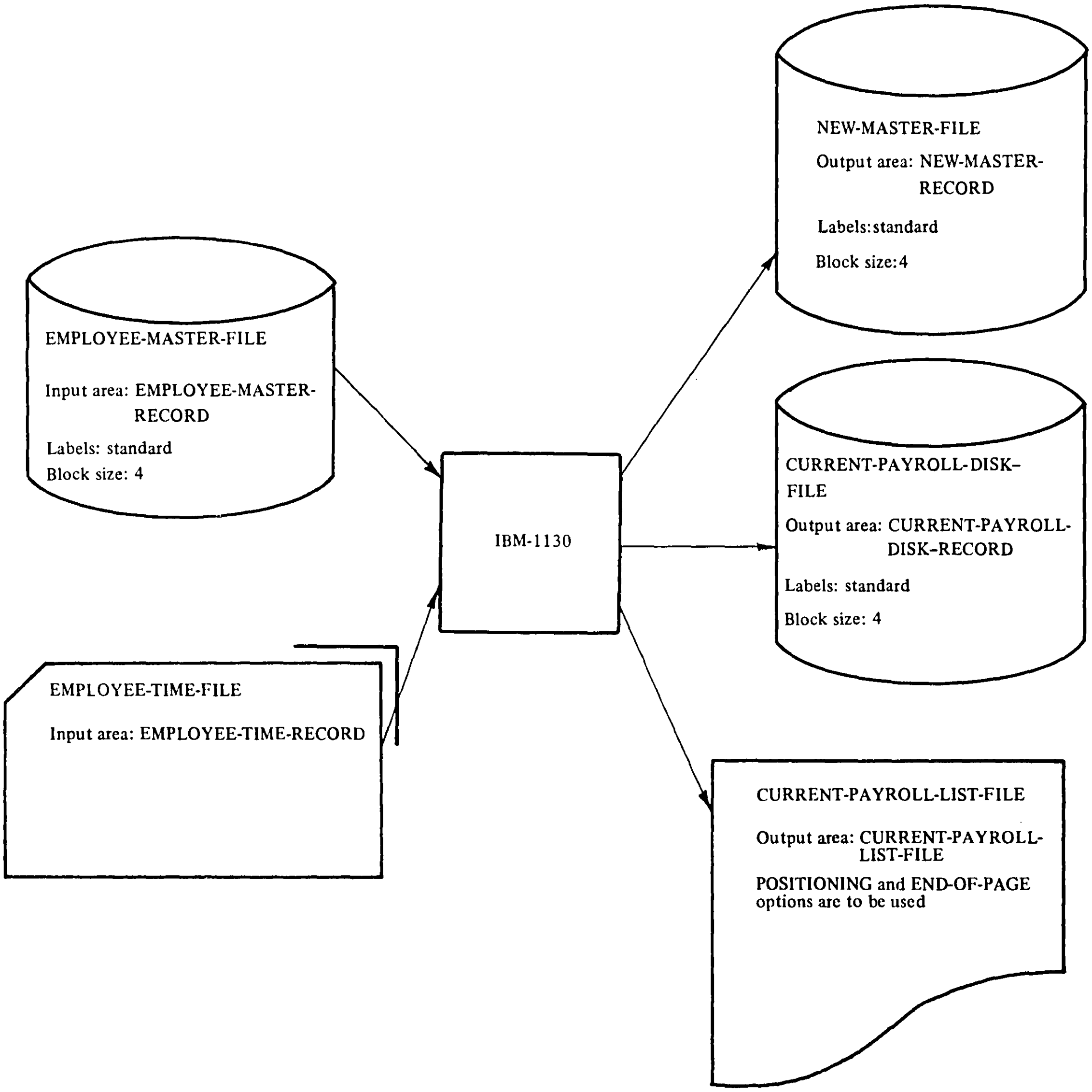
LESSON 31 - PAYROLL PROGRAM PROCESSING (2)

INTRODUCTION

In this lesson you wil build on your previous payroll processing programming efforts. The current payroll disk file created earlier will now be used for printing checks and earning statements. This lesson is optional.

This lesson will require approximately thirty minutes.

1. Read the problem description in Figure 152. Code the Identification and Environment Divisions for the program WEEKLY-PAYROLL.


NEW-MASTER-FILE
Output area: NEW-MASTER-RECORD
Labels:standard
Block size:4
EMPLOYEE-MASTER-FILE
Input area: EMPLOYEE-MASTER-RECORD
Labels: standard
Block size: 4
IBM-1130
CURRENT-PAYROLL-DISK-FILE
Output area: CURRENT-PAYROLL-DISK-RECORD
Labels: standard
Block size: 4
EMPLOYEE-TIME-FILE
Input area: EMPLOYEE-TIME-RECORD
CURRENT-PAYROLL-LIST-FILE
Output area: CURRENT-PAYROLL-LIST-FILE
POSITIONING and END-OF-PAGE options are to be used

During payroll processing for each pay period employee master records and employee time cards are to be processed to produce checks and earnings statements. As the data for the checks and earnings statements is selected and computed it is to be written into a current payroll disk file and printed in a current payroll report. The current payroll disk file, which provides a permanent record of the payroll data, can then be used to print checks and earnings statements with the desired editing and format. Since employee master records contain year-to-date data, a new master file is to be created as this data is updated for each employee.

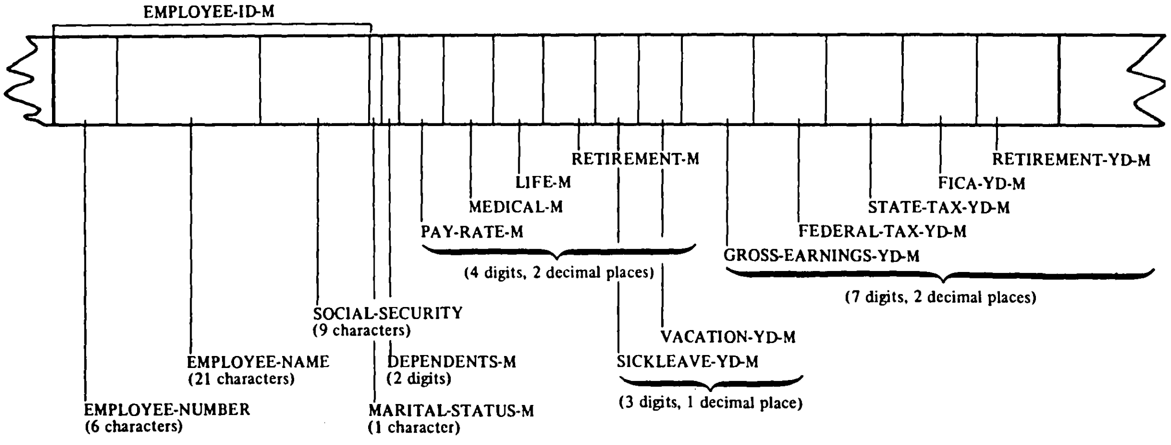
Figure 152

* * *

```
0   0    1    1    2    2    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. WEEKLY-PAYROLL.
       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-X.
           SELECT EMPLOYEE-TIME-FILE
               ASSIGN TO RD-1442.
           SELECT NEW-MASTER-FILE
               ASSIGN TO DF-2-700-X.
           SELECT CURRENT-PAYROLL-DISK-FILE
               ASSIGN TO DF-3-800-X.
           SELECT CURRENT-PAYROLL-LIST-FILE
               ASSIGN TO PR-1132-C.
               RESERVE NO ALTERNATE AREA.

-----------------------------------------------------------------------
```

2. Records from each of the files used in WEEKLY-PAYROLL are illustrated in Figures 153, 154 and 155. Use these figures along with Figure 152 to code the File Section of the Data Division.

Record in EMPLOYEE-MASTER-FILE



A record description entry for the records in EMPLOYEE-MASTER-FILE exists as library text with the name EMPLOYEE-MASTER.

Records in NEW-MASTER-FILE are to have the format of records in EMPLOYEE-MASTER-FILE. The letters NM are to replace the letter M in the data names. The condition names are not to be described in NEW-MASTER-RECORD.

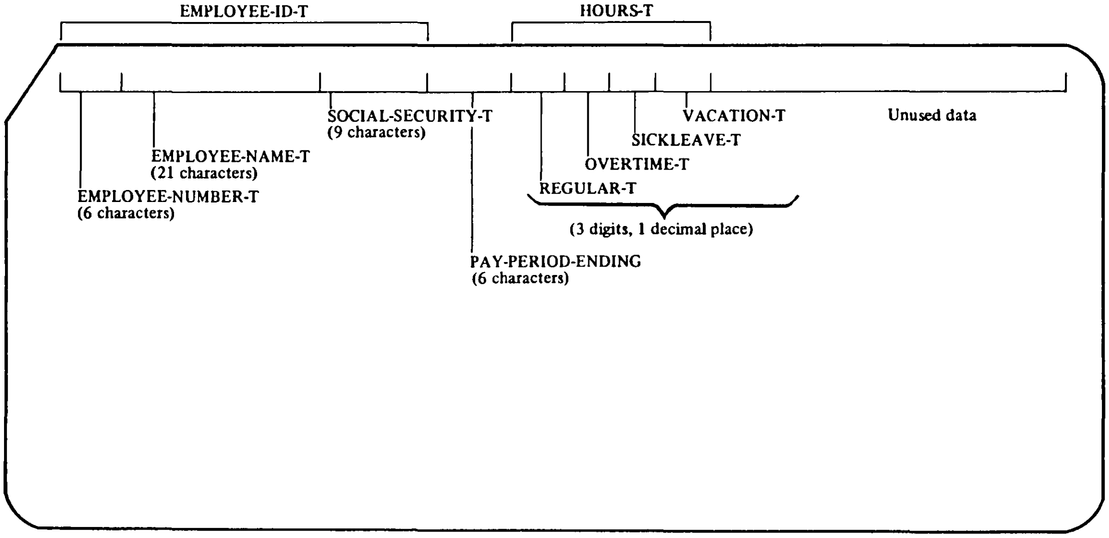
Library name: EMPLOYEE-MASTER

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  EMPLOYEE-ID-M.
               03  EMPLOYEE-NUMBER PIC X(6).
               03  EMPLOYEE-NAME PIC X(19).
               03  SOCIAL-SECURITY PIC X(9).
           02  MARITAL-STATUS-M PIC X.
               88  SINGLE VALUE 'S'.
               88  MARRIED VALUE 'M'.
           02  DEPENDENTS-M PIC 99.
           02  PAY-RATE-M PIC 99V99.
           02  MEDICAL-M PIC 99V99.
           02  RETIREMENT-M PIC 99V99.
           02  SICKLEAVE-YD-M PIC 99V9.
           02  VACATION-YD-M PIC 99V9.
           02  GROSS-EARNINGS-YD-M PIC 9(5)V99.
           02  FEDERAL-TAX-YD-M PIc 9(5)V99.
           02  STATE-TAX-YD-M PIC 9(5)V99.
           02  FICA-YD-M PIC 9(5)V99.
           02  RETIREMENT-YD-M PIC 9(5)V99.
```

Figure 153

Record in EMPLOYEE-TIME-FILE



Library name: EMPLOYEE-TIME-CARD

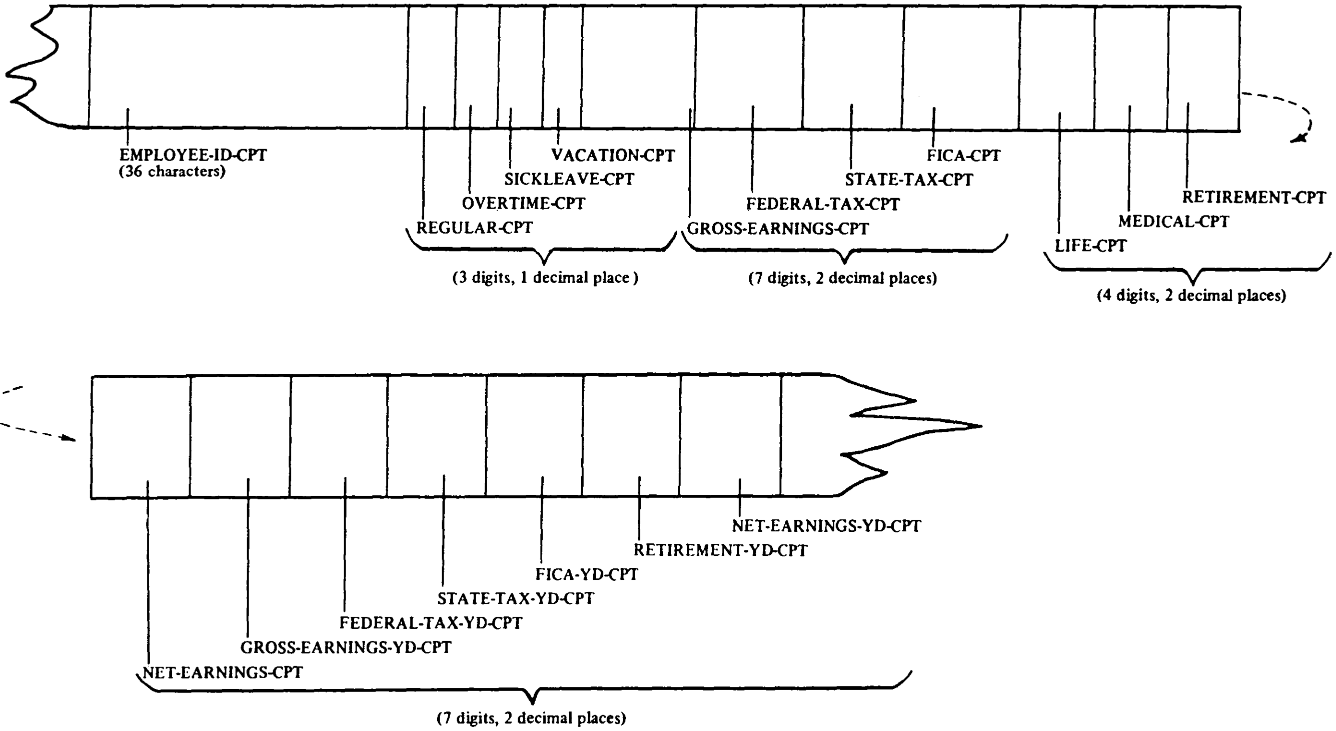
Text:

```
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  EMPLOYEE-ID-T.
              03  EMPLOYEE-NUMBER-T PIC X(6).
              03  EMPLOYEE-NAME-T PIC X(19).
              03  SOCIAL-SECURITY-T PIC X(9).
          02  PAY-PERIOD-ENDING PIC X(6).
          02  HOURS-T.
              03  REGULAR-T PIC 99V9.
              03  OVERTIME-T PIC 99V9.
              03  SICKLEAVE-T PIC 99V9.
              03  VACATION-T PIC 99V9.
          02  FILLER PIC X(26).
```

Figure 154

Figure 155

Record in CURRENT-PAYROLL-TAPE-FILE



Record in CURRENT-PAYROLL-LIST-FILE

CURRENT-PAYROLL-LIST-LINE (121 characters)

Several working storage record variables will be used along with CURRENT-PAYROLL-LIST-LINE to provide a page title, headings, and detail lines with specified editing. These variables are illustrated in Figure 16-6.

```
                          *        *        *

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..

       FILE SECTION.
       FD  EMPLOYEE-MASTER-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  EMPLOYEE-MASTER-RECORD
           COPY EMPLOYEE-MASTER.
       FD  EMPLOYEE-TIME-FILE
           LABEL RECORDS ARE OMITTED.
       01  EMPLOYEE-TIME-RECORD
           COPY EMPLOYEE-TIME-CARD.
       FD  NEW-MASTER-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  NEW-MASTER-RECORD.
           02  EMPLOYEE-ID-NM.
               03  EMPLOYEE-NUMBER PIC X(6).
               03  EMPLOYEE-NAME PIC X(19).
               03  SOCIAL-SECURITY PIC X(9).
           02  MARITAL-STATUS-NM PIC X.
           02  DEPENDENTS-NM PIC 99.
           02  PAY-RATE-NM PIC 99V99.
           02  MEDICAL-NM PIC 99V99.
           02  RETIREMENT-NM PIC 99V99.
           02  SICKLEAVE-YD-NM PIC 99V9.
           02  VACATION-YD-NM PIC 99V9.
           02  GROSS-EARNINGS-YD-NM PIC 9(5)V99.
           02  FEDERAL-TAX-YD-NM PIC 9(5)V99.
           02  STATE-TAX-YD-NM PIC 9(5)V99.
           02  FICA-YD-NM PIC 9(5)V99.
           02  RETIREMENT-YD-NM PIC 9(5)V99.
       FD  CURRENT-PAYROLL-DISK-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  CURRENT-PAYROLL-DISK-RECORD.
           02  EMPLOYEE-ID-CPT PIC X(34).
           02  REGULAR-CPT PIC 99V9.
           02  OVERTIME-CPT PIC 99V9.
           02  SICKLEAVE-CPT PIC 99V9.
           02  VACATION-CPT PIC 99V9.
           02  GROSS-EARNINGS-CPT PIC 9(5)V99.
           02  FEDERAL-TAX-CPT PIC 9(5)V99.
           02  STATE-TAX-CPT PIC 9(5)V99.
           02  FICA-CPT PIC 9(5)V99.
           02  MEDICAL-CPT PIC 99V99.
           02  RETIREMENT-CPT PIC 99V99.
           02  NET-EARNINGS-CPT PIC 9(5)V99.
           02  GROSS-EARNINGS-YD-CPT PIC 9(5)V99.
           02  FEDERAL-TAX-YD-CPT PIC 9(5)V99.
           02  STATE-TAX-YD-CPT PIC 9(5)V99.
           02  FICA-YD-CPT PIC 9(5)V99.
           02  RETIREMENT-YD-CPT PIC 9(5)V99.
           02  NET-EARNINGS-YD-CPT PIC 9(5)V99.
       FD  CURRENT-PAYROLL-LIST-FILE
           LABEL RECORDS ARE OMITTED.
       01  CURRENT-PAYROLL-LIST-LINE PIC X(120).

----------------------------------------------------------------------
```

3.

```
                           X----X
                         (END-DATE)
```

An independent variable to be used in WEEKLY-PAYROLL is shown above. Its value is to be keyed into storage through the console typewriter at the beginning of the program. The value is to be moved to DATE in TITLE-RECORD to be printed at the top of each page. Code the working storage entry for this variable.

*   *   *

```
0   0    1    1    2    2    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  END-DATE PIC X(6).
```

----

4. Record variables used along with the output area CURRENT-PAYROLL-LIST-LINE are illustrated on the Printer Spacing Charts in Figure 156. Code the working storage entries for the title and heading record variables.

Working storage record variables used with CURRENT-PAYROLL-LIST-LINE

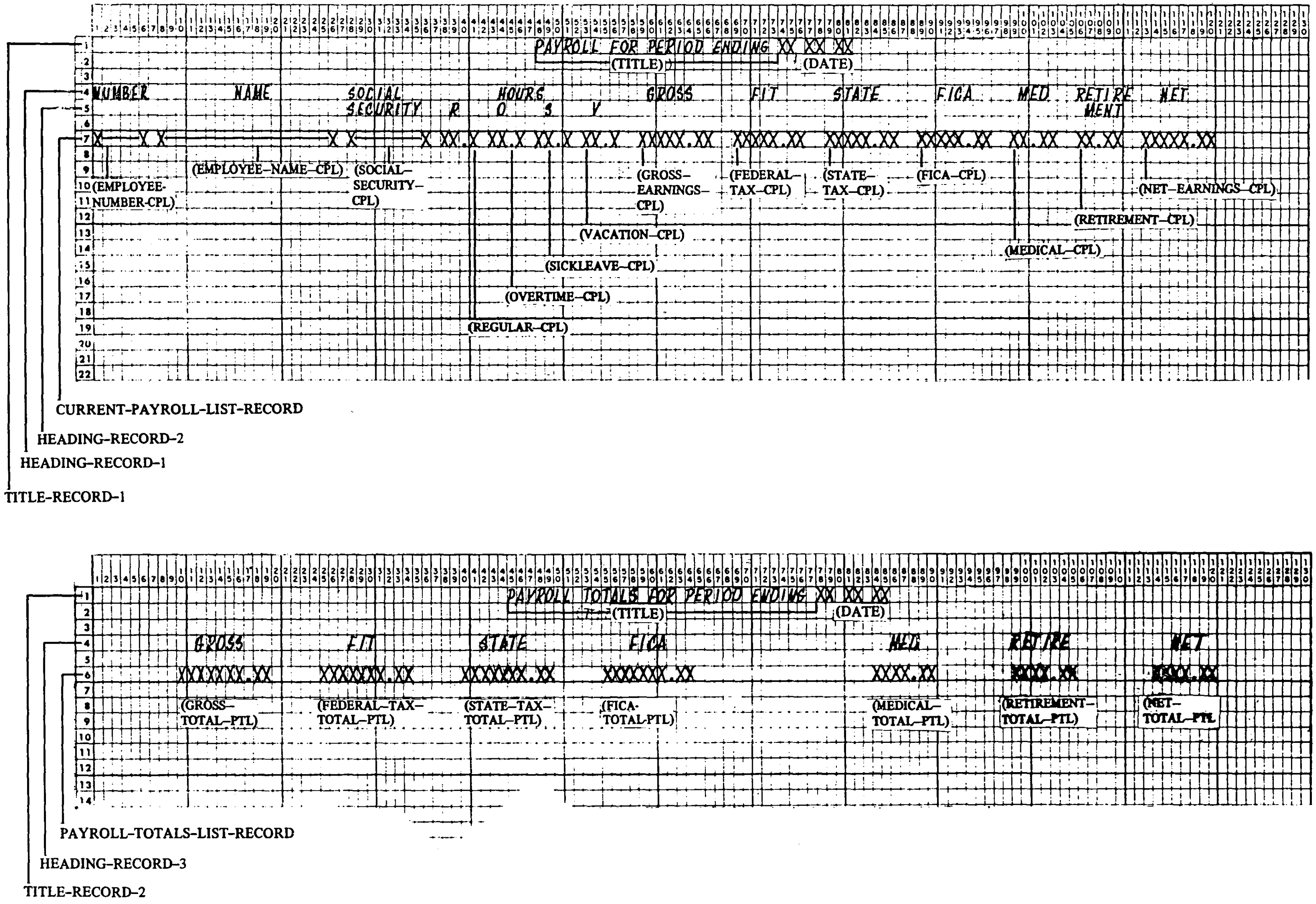


Figure 156

```
                          *        *        *

0   0    1    1    2    2    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  TITLE-RECORD-1.
           02  FILLER PIC X(50) VALUE SPACES.
           02  TITLE PIC X(26) VALUE
               'PAYROLL FOR PERIOD ENDING'.
           02  DATE PIC 99B99B99.
           02  FILLER PIC X(36) VALUE SPACES.
       01  HEADING-RECORD-1.
           02  FILLER PIC X VALUE SPACE.
           02  HEADING-1 PIC X(15) VALUE
               'NUMBER         '.
           02  HEADING-2 PIC X(14) VALUE
               'NAME          '.
           02  HEADING-3 PIC X(16) VALUE
               'SOCIAL          '.
           02  HEADING-4 PIC X(16) VALUE
               'HOURS           '.
           02  HEADING-5 PIC X(11) VALUE
               'GROSS      '.
           02  HEADING-6 PIC X(9) VALUE
               'FIT      '.
           02  HEADING-7 PIC X(11) VALUE
               'STATE      '.
           02  HEADING-8 PIC X(8) VALUE
               'FICA    '.
           02  HEADING-10 PIC X(6) VALUE
               'MED   '.
           02  HEADING-11 PIC X(9) VALUE
               'RETIRE   '.
           02  HEADING-12 PIC X(9) VALUE
               'NET      '.
       01  HEADING-RECORD-2.
           02  FILLER PIC X(17) VALUE SPACES.
           02  HEADING-1 PIC X(11) VALUE
               'SECURITY   '.
           02  HEADING-2 PIC X(16) VALUE
               'R    O    S    V'.
           02  FILLER PIC X(59) VALUE SPACES.
           02  HEADING-3 PIC X(17) VALUE
               'MENT             '.
       01  TITLE-RECORD-2.
           02  FILLER PIC X(45) VALUE SPACES.
           02  TITLE PIC X(33) VALUE
           'PAYROLL TOTALS FOR PERIOD ENDING'.
           02  DATE PIC 99B99B99.
           02  FILLER PIC X(34) VALUE SPACES.
       01  HEADING-RECORD-3.
           02  FILLER PIC X(12) VALUE SPACES.
           02  HEADING-1 PIC X(16) VALUE
               'GROSS           '.
           02  HEADING-2 PIC X(14) VALUE
               'FIT           '.
           02  HEADING-3 PIC X(16) VALUE
               'STATE           '.
           02  HEADING-4 PIC X(15) VALUE
               'FICA           '.
           02  HEADING-6 PIC X(13) VALUE
               'MED          '.
           02  HEADING-7 PIC X(17) VALUE
               'RETIRE           '.
           02  HEADING-8 PIC X(14) VALUE
               'NET           '.

------------------------------------------------------------------------
```

5. Code the working storage entries for CURRENT-PAYROLL-LIST-RECORD and PAYROLL-TOTALS-LIST-RECORD, the record variables illustrated in Figure 156. Provide for suppression of leading zeros and decimal point insertion.

* * *

```
0   0    1    1    2    2    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  CURRENT-PAYROLL-LIST-RECORD.
           02  CARRIAGE-CONTROL PIC X
               VALUE SPACE.
           02  EMPLOYEE-NUMBER-CPL PIC X(6).
           02  FILLER PIC X VALUE SPACE.
           02  EMPLOYEE-NAME-CPL PIC X(19).
           02  FILLER PIC X VALUE SPACE.
           02  SOCIAL-SECURITY-CPL PIC X(9).
           02  FILLER PIC X VALUE SPACE.
           02  REGULAR-CPL PIC ZZ,9.
           02  FILLER PIC X VALUE SPACE.
           02  OVERTIME-CPL PIC ZZ.9.
           02  FILLER PIC X VALUE SPACE.
           02  SICKLEAVE-CPL PIC ZZ.9.
           02  FILLER PIC X VALUE SPACE.
           02  VACATION-CPL PIC ZZ.9.
           02  FILLER PIC XX VALUE SPACES.
           02  GROSS-EARNINGS-CPL PIC Z(5).99.
           02  FILLER PIC XX VALUE SPACES.
           02  FEDERAL-TAX-CPL PIC Z(5).99.
           02  FILLER PIC XX VALUE SPACES.
           02  STATE-TAX-CPL PIC Z(5).99.
           02  FILLER PIC XX VALUE SPACES.
           02  FICA-CPL PIC Z(5).99.
           02  FILLER PIC XX VALUE SPACES.
           02  MEDICAL-CPL PIC ZZ.99.
           02  FILLER PIC XX VALUE SPACES.
           02  RETIREMENT-CPL PIC ZZ.99.
           02  FILLER PIC XX VALUE SPACES.
           02  NET-EARNINGS-CPL PIC Z(5).99.
           02  FILLER PIC X(3) VALUE SPACES.
       01  PAYROLL-TOTALS-LIST-RECORD.
           02  FILLER PIC X(10) VALUE SPACES.
           02  GROSS-TOTAL-PTL PIC Z(7).99.
           02  FILLER PIC X(5) VALUE SPACES.
           02  FEDERAL-TAX-TOTAL-PTL
               PIC Z(7).99.
           02  FILLER PIC X(5) VALUE SPACES.
           02  STATE-TAX-TOTAL-PTL PIC Z(7).99.
           02  FILLER PIC X(5) VALUE SPACES.
           02  FICA-TOTAL-PTL PIC Z(7).99.
           02  FILLER PIC X(7) VALUE SPACES.
           02  FILLER PIC X(8) VALUE SPACES.
           02  MEDICAL-TOTAL-PTL PIC Z(4).99.
           02  FILLER PIC X(8) VALUE SPACES.
           02  RETIREMENT-TOTAL-PTL PIC Z(4).99.
           02  FILLER PIC X(8) VALUE SPACES.
           02  NET-TOTAL-PTL PIC Z(4).99.
           02  FILLER PIC X(3) VALUE SPACES.
```

----

6. A record variable including test and constant values and intermediate variables to be used in comparisons and computations in WEEKLY-PAYROLL is shown in Figure 157. Code a record description entry for TEST-CONSTANT-COMP-RECORD.

TEST-CONSTANT-COMP-RECORD

Variable	Value	Purpose
ZERO-TEST	00.0	Used to test whether hours were charged to sickleave or vacation
MAXIMUM-HOURS	80.0	Used to test whether hours charged to sickleave or vacation have exceeded the maximum
TIME-AND-A-HALF	1.5	Multiplied by overtime hours in computation of current earnings
EXEMPTION	00013.50	Multiplied by the number of dependents and subtracted from gross earnings to give taxable earnings
MAXIMUM-FICA-EARNINGS	07800.00	Used to test whether year-to-date gross earnings has exceeded the maximum for which FICA tax is to be deducted
FICA-PERCENT	.048	Used to compute FICA tax
STATE-TAX-PERCENT	.02	Used to compute state tax
UNUSED-HOURS EXCESS-HOURS	3 digits, 1 decimal place	Used to compute sickleave and vacation pay
GROSS-EARNINGS-NYDWR TAXABLE STANDARD-DEDUCTIONS DEDUCTIONS	7 digits, 2 decimal places	Used to compute FICA and net earnings

Figure 157

*   *   *

```
0   0    1    1    2    2    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  TEST-CONSTANT-COMP-RECORD.
               USAGE IS COMP.
           02  ZERO-TEST PIC 99V9 VALUE ZEROS.
           02  MAXIMUM-HOURS PIC 99V9
               VALUE 80.0.
           02  TIME-AND-A-HALF PIC 9V9
               VALUE 1.5.
           02  EXEMPTION PIC 9(5)V99
               VALUE 00013.50.
           02  MAXIMUM-FICA-EARNINGS
               PIC 9(5)V99 VALUE 07800.00.
           02  FICA-PERCENT PIC V999
               VALUE .048.
           02  STATE-TAX-PERCENT PIC V99
               VALUE .02.
           02  UNUSED-HOURS PIC 99V9.
           02  EXCESS-HOURS PIC 99V9.
           02  GROSS-EARNINGS-NYDWR
               PIC 9(5)V99.
           02  TAXABLE PIC 9(5)V99.
           02  STANDARD-DEDUCTIONS PIC 9(5)V99.
           02  DEDUCTIONS PIC 9(5)V99.
```

(An S may be included in the PICTURE clause of any COMP-3 variable.)

---

7. Three record variables that will be used for computation in WEEKLY-PAYROLL are shown in Figure 157. Code a record description entry for each of these variables.

CURRENT-PAYROLL-WORK-RECORD		Used to prepare values for CURRENT-PAYROLL-LIST-RECORD
REGULAR OVERTIME SICKLEAVE VACATION	3 digits, 1 decimal place	Values from EMPLOYEE-TIME-CARD
DEPENDENTS	2 digits	Values from EMPLOYEE-MASTER-RECORD
PAY-RATE	4 digits, 2 decimal places	
GROSS-EARNINGS FEDERAL-TAX STATE-TAX FICA	7 digits, 2 decimal places	Values are to be computed.
MEDICAL RETIREMENT		Values from EMPLOYEE-MASTER-RECORD
NET-EARNINGS		Value is to be computed.

YEAR-TO-DATE-WORK-RECORD		Used to prepare values for NEW-MASTER-RECORD
SICKLEAVE-YDWR VACATION-YDWR	3 digits, 1 decimal place	Values from EMPLOYEE-MASTER-RECORD to be updated from CURRENT-PAYROLL-WORK-RECORD
GROSS-EARNINGS-YDWR FEDERAL-TAX-YDWR STATE-TAX-YDWR RETIREMENT-YDWR	7 digits, 2 decimal places	

PAYROLL-TOTALS-WORK-RECORD		used to prepare values for PAYROLL-TOTALS-LIST-RECORD
GROSS-TOTAL FEDERAL-TAX-TOTAL STATE-TAX-TOTAL FICA-TOTAL MEDICAL-TOTAL RETIREMENT-TOTAL NET-TOTAL	9 digits, 2 decimal places	Values are to be accumulated from values in CURRENT-PAYROLL-WORK-RECORD for all employees. Variables should be assigned an initial value of zero.

Figure 158

*   *   *

```
0   0    1    1    2    2    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  CURRENT-PAYROLL-WORK-RECORD.
               USAGE IS COMP.
           02  REGULAR PIC 99V9.
           02  OVERTIME PIC 99V9.
           02  SICKLEAVE PIC 99V9.
           02  VACATION PIC 99V9.
           02  DEPENDENTS PIC 99.
           02  PAY-RATE PIC 99V99.
           02  GROSS-EARNINGS PIC 9(5)V99.
           02  FEDERAL-TAX PIC 9(5)V99.
           02  STATE-TAX PIC 9(5)V99.
           02  FICA PIC 9(5)V99.
           02  MEDICAL PIC 9(5)V99.
           02  RESTIREMENT PIC 9(5)V99.
           02  NET-EARNINGS PIC 9(5)V99.
       01  YEAR-TO-DATE-WORK-RECORD
               USAGE IS COMP.
           02  SICKLEAVE-YDWR PIC 99V9.
           02  VACATION-YDWR PIC 99V9.
           02  GROSS-EARNINGS-YDWR PIC 9(5)V99.
           02  FEDERAL-TAX-YDWR PIC 9(5)V99.
           02  STATE-TAX-YDWR PIC 9(5)V99.
           02  FICA-YDWR PIC 9(5)V99.
           02  RETIREMENT-YDWR PIC 9(5)V99.
       01  PAYROLL-TOTALS-WORK-RECORD.
               USAGE IS COMP.
           02  GROSS-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  FEDERAL-TAX-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  STATE-TAX-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  FICA-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  LIFE-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  MEDICAL-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  RETIREMENT-TOTAL PIC 9(7)V99
               VALUE ZEROS.
           02  NET-TOTAL PIC 9(7)V99
               VALUE ZEROS.

-----------------------------------------------------------------------
```

Percentage Method of Withholding

WEEKLY Payroll Period

(a) SINGLE person—including head of household:

If the amount of wages is: — *The amount of income tax to be withheld shall be:*

Not over $4.........0

Over—	But not over—		of excess over—
$4	—$13	......14%	—$4
$13	—$23	......$1.26, plus 15%	—$13
$23	—$85	......$2.76, plus 19%	—$23
$85	—$169	.....$14.54, plus 22%	—$85
$169	—$212	.... $33.02, plus 28%	—$169
$212		..........$45.06, plus 33%	—$212

(b) Married person—

If the amount of wages is: — *The amount of income tax to be withheld shall be:*

Not over $4.........0

Over—	But not over—		of excess over—
$4	—$23	......14%	—$4
$23	—$58	......$2.66, plus 15%	—$23
$58	—$169	.....$7.91, plus 19%	—$58
$169	—$340	.....$29.00, plus 22%	—$169
$340	—$423	.....$66.62, plus 28%	—$340
$423		..........$89.86, plus 33%	—$423

SINGLE-DOLLAR-TABLE	
SINGLE-DOLLAR-1	000
SINGLE-DOLLAR-2	004
SINGLE-DOLLAR-3	013
SINGLE-DOLLAR-4	023
SINGLE-DOLLAR-5	085
SINGLE-DOLLAR-6	169
SINGLE-DOLLAR-7	212
SINGLE-DOLLAR-8	999

SINGLE-PERCENT-TABLE	
SINGLE-PERCENT-1	‸00
SINGLE-PERCENT-2	‸14
SINGLE-PERCENT-3	‸15
SINGLE-PERCENT-4	‸19
SINGLE-PERCENT-5	‸22
SINGLE-PERCENT-6	‸28
SINGLE-PERCENT-7	‸33
SINGLE-PERCENT-8	‸33

SINGLE-CONSTANT-TABLE	
SINGLE-CONSTANT-1	00‸00
SINGLE-CONSTANT-2	00‸00
SINGLE-CONSTANT-3	01‸26
SINGLE-CONSTANT-4	02‸76
SINGLE-CONSTANT-5	14‸54
SINGLE-CONSTANT-6	33‸02
SINGLE-CONSTANT-7	45‸06
SINGLE-CONSTANT-8	45‸06

(Index DS)

(Index PS)

(Index CS)

SINGLE-DOLLAR-OVERLAY	
SINGLE-DOLLAR (1)	
SINGLE-DOLLAR (2)	
SINGLE-DOLLAR (3)	
SINGLE-DOLLAR (4)	
SINGLE-DOLLAR (5)	
SINGLE-DOLLAR (6)	
SINGLE-DOLLAR (7)	
SINGLE-DOLLAR (8)	

SINGLE-PERCENT-OVERLAY	
SINGLE-PERCENT (1)	
SINGLE-PERCENT (2)	
SINGLE-PERCENT (3)	
SINGLE-PERCENT (4)	
SINGLE-PERCENT (5)	
SINGLE-PERCENT (6)	
SINGLE-PERCENT (7)	
SINGLE-PERCENT (8)	

SINGLE-CONSTANT-OVERLAY	
SINGLE-CONSTANT (1)	
SINGLE-CONSTANT (2)	
SINGLE-CONSTANT (3)	
SINGLE-CONSTANT (4)	
SINGLE-CONSTANT (5)	
SINGLE-CONSTANT (6)	
SINGLE-CONSTANT (7)	
SINGLE-CONSTANT (8)	

MARRIED-DOLLAR-TABLE	
MARRIED-DOLLAR-1	000
MARRIED-DOLLAR-2	004
MARRIED-DOLLAR-3	023
MARRIED-DOLLAR-4	058
MARRIED-DOLLAR-5	169
MARRIED-DOLLAR-6	340
MARRIED-DOLLAR-7	423
MARRIED-DOLLAR-8	999

MARRIED-PERCENT-TABLE	
MARRIED-PERCENT-1	‸00
MARRIED-PERCENT-2	‸14
MARRIED-PERCENT-3	‸15
MARRIED-PERCENT-4	‸19
MARRIED-PERCENT-5	‸22
MARRIED-PERCENT-6	‸28
MARRIED-PERCENT-7	‸33
MARRIED-PERCENT-8	‸33

MARRIED-CONSTANT-TABLE	
MARRIED-CONSTANT-1	00‸00
MARRIED-CONSTANT-2	00‸00
MARRIED-CONSTANT-3	02‸66
MARRIED-CONSTANT-4	07‸91
MARRIED-CONSTANT-5	29‸00
MARRIED-CONSTANT-6	66‸62
MARRIED-CONSTANT-7	89‸86
MARRIED-CONSTANT-8	89‸86

(Index DM)

(Index PM)

(Index CM)

MARRIED-DOLLAR-OVERLAY	
MARRIED-DOLLAR (1)	
MARRIED-DOLLAR (2)	
MARRIED-DOLLAR (3)	
MARRIED-DOLLAR (4)	
MARRIED-DOLLAR (5)	
MARRIED-DOLLAR (6)	
MARRIED-DOLLAR (7)	
MARRIED-DOLLAR (8)	

MARRIED-PERCENT-OVERLAY	
MARRIED-PERCENT (1)	
MARRIED-PERCENT (2)	
MARRIED-PERCENT (3)	
MARRIED-PERCENT (4)	
MARRIED-PERCENT (5)	
MARRIED-PERCENT (6)	
MARRIED-PERCENT (7)	
MARRIED-PERCENT (8)	

MARRIED-CONSTANT-OVERLAY	
MARRIED-CONSTANT (1)	
MARRIED-CONSTANT (2)	
MARRIED-CONSTANT (3)	
MARRIED-CONSTANT (4)	
MARRIED-CONSTANT (5)	
MARRIED-CONSTANT (6)	
MARRIED-CONSTANT (7)	
MARRIED-CONSTANT (8)	

Figure 159

```
0   0    1    1    2    2    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  SINGLE-DOLLAR-TABLE USAGE COMP.
           02  SINGLE-DOLLAR-1 PIC 9(5)V99 VALUE ZEROS.
           02  SINGLE-DOLLAR-2 PIC 9(5)V99 VALUE 00004.00.
           02  SINGLE-DOLLAR-3 PIC 9(5)V99 VALUE 00013.00.
           02  SINGLE-DOLLAR-4 PIC 9(5)V99 VALUE 00023.00.
           02  SINGLE-DOLLAR-5 PIC 9(5)V99 VALUE 00085.00.
           02  SINGLE-DOLLAR-6 PIC 9(5)V99 VALUE 00169.00.
           02  SINGLE-DOLLAR-7 PIC 9(5)V99 VALUE 00212.00.
           02  SINGLE-DOLLAR-8 PIC 9(5)V99 VALUE 00999.00.
       01  SINGLE-DOLLAR-OVERLAY
               REDEFINES SINGLE-DOLLAR-TABLE.
           02  SINGLE-DOLLAR
               OCCURS 8 TIMES INDEXED BY DS
               USAGE COMP PIC 9(5)V99.
       01  SINGLE-PERCENT-TABLE USAGE COMP.
           02  SINGLE-PERCENT-1 PIC V99 VALUE ZEROS.
           02  SINGLE-PERCENT-2 PIC V99 VALUE .14.
           02  SINGLE-PERCENT-3 PIC V99 VALUE .15.
           02  SINGLE-PERCENT-4 PIC V99 VALUE .19.
           02  SINGLE-PERCENT-5 PIC V99 VALUE .22.
           02  SINGLE-PERCENT-6 PIC V99 VALUE .28.
           02  SINGLE-PERCENT-7 PIC V99 VALUE .33.
           02  SINGLE-PERCENT-8 PIC V99 VALUE .33.
       01  SINGLE-PERCENT-OVERLAY
               REDEFINES SINGLE-PERCENT-TABLE.
           02  SINGLE-PERCENT
               OCCURS 8 TIMES INDEXED BY PS
               USAGE COMP PIC V99.
       01  SINGLE-CONSTANT-TABLE USAGE COMP.
           02  SINGLE-CONSTANT-1 PIC 9(5)V99 VALUE ZEROS.
           02  SINGLE-CONSTANT-2 PIC 9(5)V99 VALUE ZEROS.
           02  SINGLE-CONSTANT-3 PIC 9(5)V99 VALUE 00001.26.
           02  SINGLE-CONSTANT-4 PIC 9(5)V99 VALUE 00002.76.
           02  SINGLE-CONSTANT-5 PIC 9(5)V99 VALUE 00014.54.
           02  SINGLE-CONSTANT-6 PIC 9(5)V99 VALUE 00033.02.
           02  SINGLE-CONSTANT-7 PIC 9(5)V99 VALUE 00045.06.
           02  SINGLE-CONSTANT-8 PIC 9(5)V99 VALUE 00045.06.
       01  SINGLE-CONSTANT-OVERLAY
           REDEFINES SINGLE-CONSTANT-TABLE.
           02  SINGLE-CONSTANT
               OCCURS 8 TIMES INDEXED BY CS
               USAGE COMP PIC 9(5)V99.
```

Figure 160

8. The values to be used in computation of federal income tax (percentage method) for the weekly payroll period are shown in Figure 159. The record variables and tables to be used in WEEKLY-PAYROLL are also illustrated in this figure. The record description entries for values for single employees are given in Figure 160. Code the entries for the values for married employees. Since the federal income tax will be used in the computation of net earnings and year-to-date values, the dollar amounts and constants should be described with the picture 9(5)V99 as in NEW-MASTER-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..

       01  MARRIED-DOLLAR-TABLE USAGE IS COMP.
           02  MARRIED-DOLLAR-1 PIC 9(5)V99
               VALUE ZEROS.
           02  MARRIED-DOLLAR-2 PIC 9(5)V99
               VALUE 00004.00.
           02  MARRIED-DOLLAR-3 PIC 9(5)V99
               VALUE 00023.00.
           02  MARRIED-DOLLAR-4 PIC 9(5)V99
               VALUE 00058.00.
           02  MARRIED-DOLLAR-5 PIC 9(5)V99
               VALUE 00169.00.
           02  MARRIED-DOLLAR-6 PIC 9(5)V99
               VALUE 00340.00.
           02  MARRIED-DOLLAR-7 PIC 9(5)V99
               VALUE 00423.00.
           02  MARRIED-DOLLAR-8 PIC 9(5)V99
               VALUE 00999.00.
       01  MARRIED-DOLLAR-OVERLAY
               REDEFINES MARRIED-DOLLAR-TABLE.      (1)
           02  MARRIED-DOLLAR
               OCCURS 8 TIMES INDEXED BY DM
               USAGE IS COMP PIC 9(5)V99.
       01  MARRIED-PERCENT-TABLE USAGE IS COMP.
           02  MARRIED-PERCENT-1 PIC V99
               VALUE .00.
           02  MARRIED-PERCENT-2 PIC V99
               VALUE .14.
           02  MARRIED-PERCENT-3 PIC V99
               VALUE .15.
           02  MARRIED-PERCENT-4 PIC V99
               VALUE .19.
           02  MARRIED-PERCENT-5 PIC V99
               VALUE .22.
           02  MARRIED-PERCENT-6 PIC V99
               VALUE .28.
           02  MARRIED-PERCENT-7 PIC V99
               VALUE .33.
           02  MARRIED-PERCENT-8 PIC V99
               VALUE .33.
```

```
01  MARRIED-PERCENT-OVERLAY
        REDEFINES MARRIED-PERCENT-TABLE.     (1)
    02  MARRIED-PERCENT
        OCCURS 8 TIMES INDEXED BY PM
        USAGE IS COMP PIC V99.
01  MARRIED-CONSTANT-TABLE USAGE IS COMP.
    02  MARRIED-CONSTANT-1 PIC 9(5)V99
        VALUE ZEROS.
    02  MARRIED-CONSTANT-2 PIC 9(5)V99
        VALUE ZEROS.
    02  MARRIED-CONSTANT-3 PIC 9(5)V99
        VALUE 00002.66.
    02  MARRIED-CONSTANT-4 PIC 9(5)V99
        VALUE 00007.91.
    02  MARRIED-CONSTANT-5 PIC 9(5)V99
        VALUE 00029.00.
    02  MARRIED-CONSTANT-6 PIC 9(5)V99
        VALUE 00066.62.
    02  MARRIED-CONSTANT-7 PIC 9(5)V99
        VALUE 00089.86.
    02  MARRIED-CONSTANT-8 PIC 9(5)V99
        VALUE 00089.86.
01  MARRIED-CONSTANT-OVERLAY                 (1)
        REDEFINES MARRIED-CONSTANT-TABLE.
    02  MARRIED-CONSTANT
        OCCURS 8 TIMES INDEXED BY CM
        USAGE IS COMP PIC 9(5)V99.
```

You have now completed the first three divisions of WEEKLY-PAYROLL.

The program flow chart for WEEKLY-PAYROLL is presented in eight segments in Figures 161 through 168. Although payroll programs vary with the individual needs of employers, processing steps that are likely to be included in any payroll program have been selected for WEEKLY-PAYROLL. In the next sequence of frames you will be coding the Procedure Division for the program.

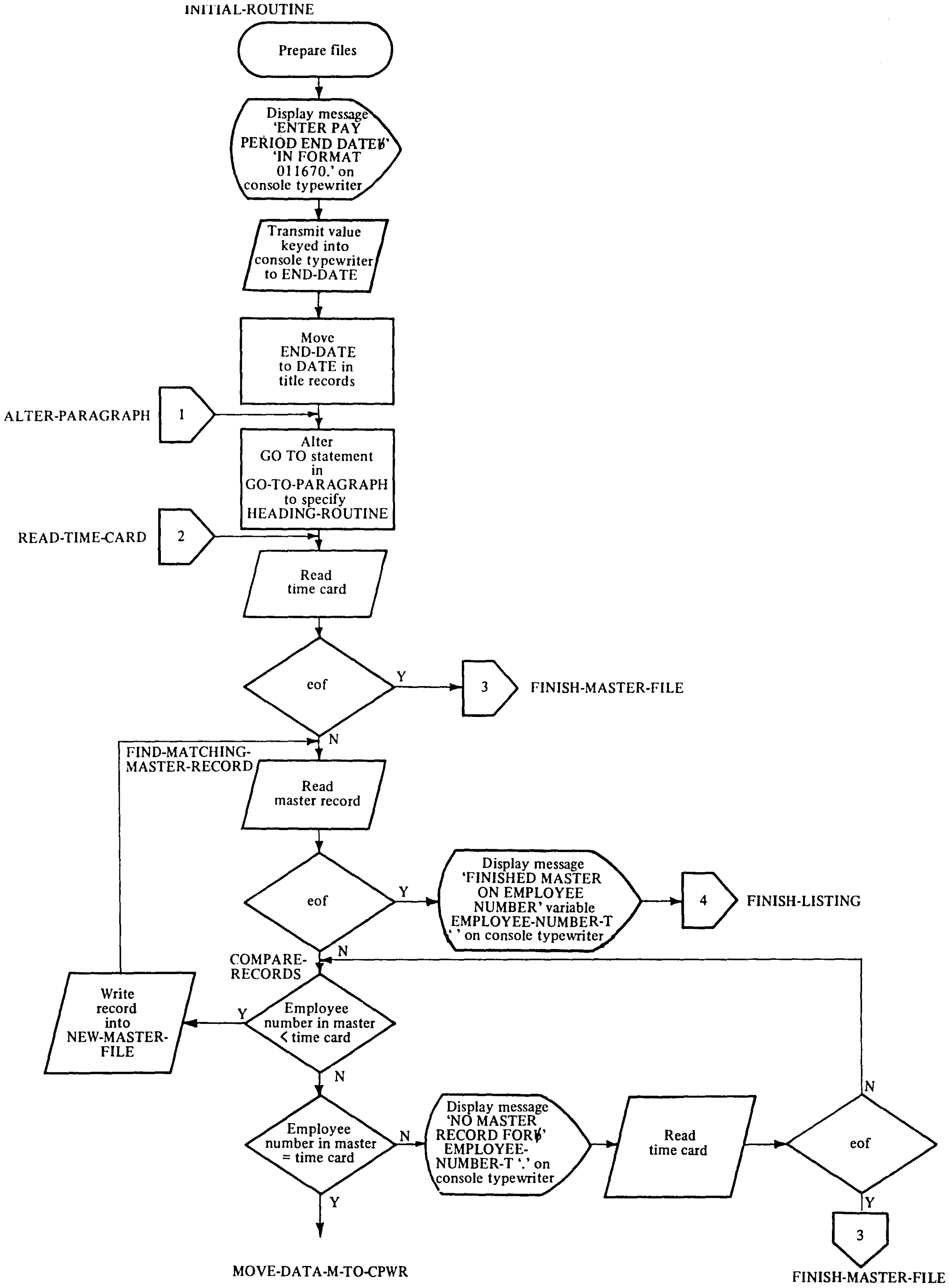

INITIAL-ROUTINE
Prepare files
Display message 'ENTER PAY PERIOD END DATEb̸' 'IN FORMAT 011670.' on console typewriter
Transmit value keyed into console typewriter to END-DATE
Move END-DATE to DATE in title records
ALTER-PARAGRAPH
1
Alter GO TO statement in GO-TO-PARAGRAPH to specify HEADING-ROUTINE
READ-TIME-CARD
2
Read time card
eof
Y
3
FINISH-MASTER-FILE
N
FIND-MATCHING-MASTER-RECORD
Read master record
eof
Y
Display message 'FINISHED MASTER ON EMPLOYEE NUMBER' variable EMPLOYEE-NUMBER-T '.' on console typewriter
4
FINISH-LISTING
N
COMPARE-RECORDS
Write record into NEW-MASTER-FILE
Y
Employee number in master < time card
N
Employee number in master = time card
N
Display message 'NO MASTER RECORD FORb̸' EMPLOYEE-NUMBER-T '.' on console typewriter
Read time card
eof
N
Y
3
FINISH-MASTER-FILE
Y
MOVE-DATA-M-TO-CPWR

Figure 161

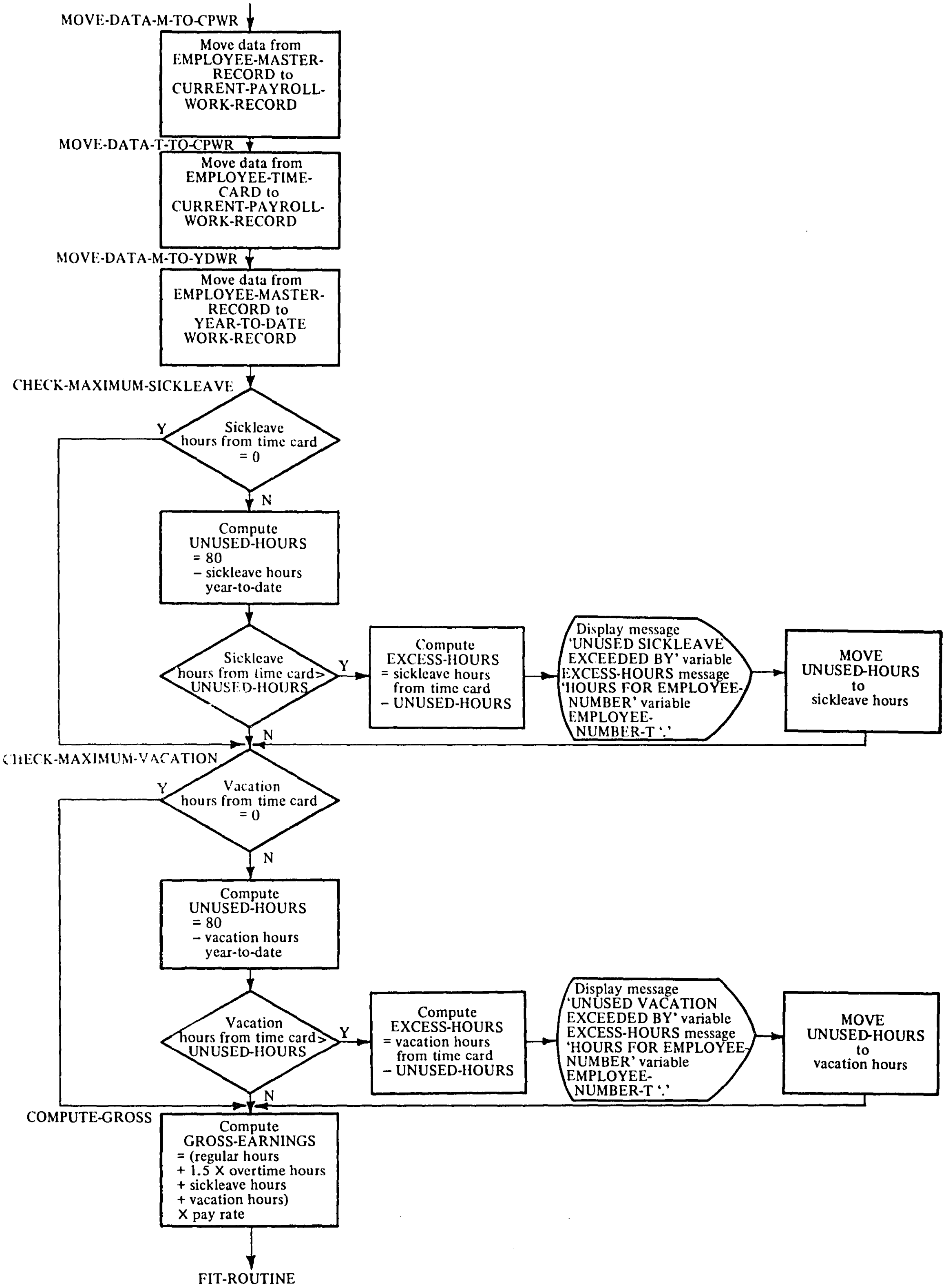

MOVE-DATA-M-TO-CPWR
Move data from EMPLOYEE-MASTER-RECORD to CURRENT-PAYROLL-WORK-RECORD
MOVE-DATA-T-TO-CPWR
Move data from EMPLOYEE-TIME-CARD to CURRENT-PAYROLL-WORK-RECORD
MOVE-DATA-M-TO-YDWR
Move data from EMPLOYEE-MASTER-RECORD to YEAR-TO-DATE WORK-RECORD
CHECK-MAXIMUM-SICKLEAVE
Sickleave hours from time card = 0
Y
N
Compute UNUSED-HOURS = 80 – sickleave hours year-to-date
Sickleave hours from time card> UNUSED-HOURS
Y
N
Compute EXCESS-HOURS = sickleave hours from time card – UNUSED-HOURS
Display message 'UNUSED SICKLEAVE EXCEEDED BY' variable EXCESS-HOURS message 'HOURS FOR EMPLOYEE-NUMBER' variable EMPLOYEE-NUMBER-T '.'
MOVE UNUSED-HOURS to sickleave hours
CHECK-MAXIMUM-VACATION
Vacation hours from time card = 0
Y
N
Compute UNUSED-HOURS = 80 – vacation hours year-to-date
Vacation hours from time card> UNUSED-HOURS
Y
N
Compute EXCESS-HOURS = vacation hours from time card – UNUSED-HOURS
Display message 'UNUSED VACATION EXCEEDED BY' variable EXCESS-HOURS message 'HOURS FOR EMPLOYEE-NUMBER' variable EMPLOYEE-NUMBER-T '.'
MOVE UNUSED-HOURS to vacation hours
COMPUTE-GROSS
Compute GROSS-EARNINGS = (regular hours + 1.5 X overtime hours + sickleave hours + vacation hours) X pay rate
FIT-ROUTINE

Figure 162

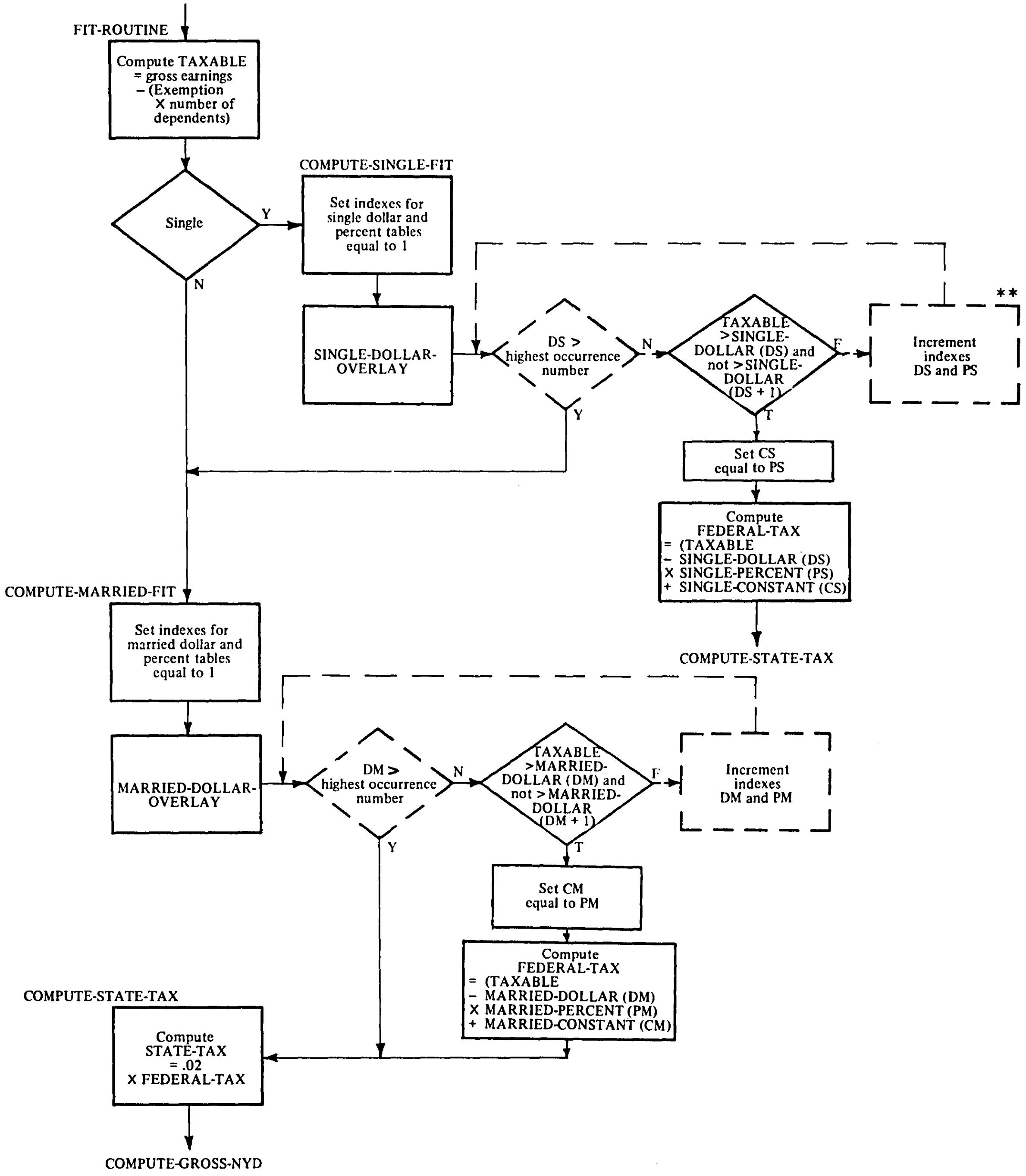
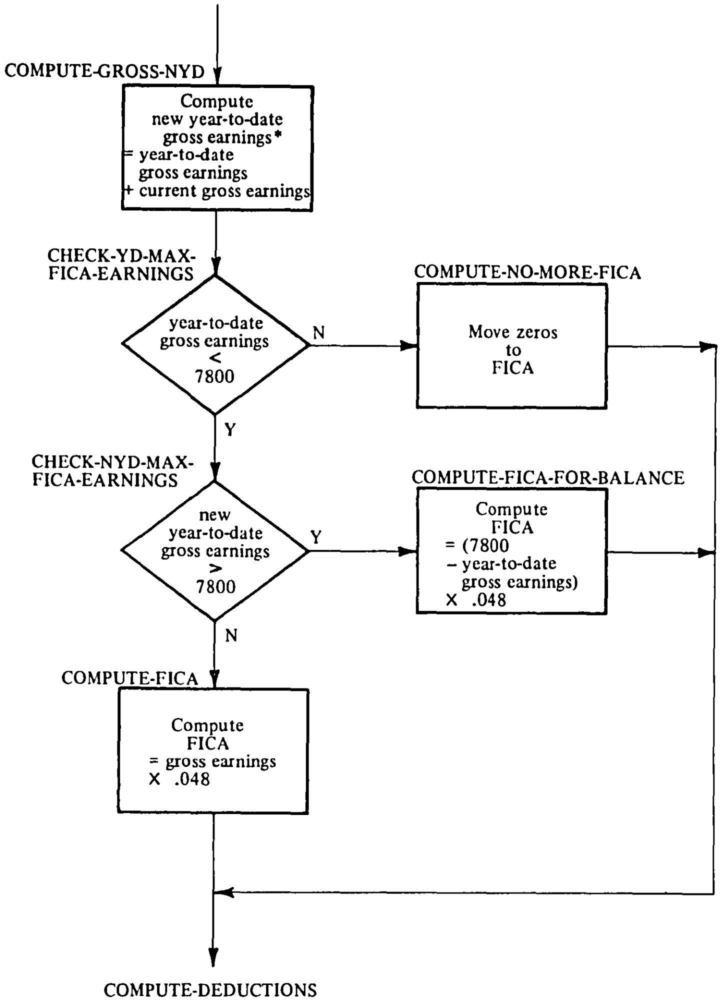

FIT-ROUTINE
Compute TAXABLE = gross earnings – (Exemption X number of dependents)
Single
Y
N
COMPUTE-SINGLE-FIT
Set indexes for single dollar and percent tables equal to 1
SINGLE-DOLLAR-OVERLAY
DS > highest occurrence number
N
Y
TAXABLE >SINGLE-DOLLAR (DS) and not >SINGLE-DOLLAR (DS + 1)
F
T
**
Increment indexes DS and PS
Set CS equal to PS
Compute FEDERAL-TAX = (TAXABLE – SINGLE-DOLLAR (DS) X SINGLE-PERCENT (PS) + SINGLE-CONSTANT (CS)
COMPUTE-STATE-TAX
COMPUTE-MARRIED-FIT
Set indexes for married dollar and percent tables equal to 1
MARRIED-DOLLAR-OVERLAY
DM > highest occurrence number
N
Y
TAXABLE >MARRIED-DOLLAR (DM) and not >MARRIED-DOLLAR (DM + 1)
F
T
Increment indexes DM and PM
Set CM equal to PM
Compute FEDERAL-TAX = (TAXABLE – MARRIED-DOLLAR (DM) X MARRIED-PERCENT (PM) + MARRIED-CONSTANT (CM)
COMPUTE-STATE-TAX
Compute STATE-TAX = .02 X FEDERAL-TAX
COMPUTE-GROSS-NYD

Figure 163


COMPUTE-GROSS-NYD
Compute new year-to-date gross earnings* = year-to-date gross earnings + current gross earnings
CHECK-YD-MAX-FICA-EARNINGS
year-to-date gross earnings < 7800
N
Y
COMPUTE-NO-MORE-FICA
Move zeros to FICA
CHECK-NYD-MAX-FICA-EARNINGS
new year-to-date gross earnings > 7800
Y
N
COMPUTE-FICA-FOR-BALANCE
Compute FICA = (7800 - year-to-date gross earnings) X .048
COMPUTE-FICA
Compute FICA = gross earnings X .048
COMPUTE-DEDUCTIONS

* Since the year-to-date gross earnings and new year-to-date gross earnings are compared in FICA processing, the variable GROSS-EARNINGS-NYDWR in TEST-CONSTANT-COMP-RECORD is used for new year-to-date gross earnings.

Figure 164

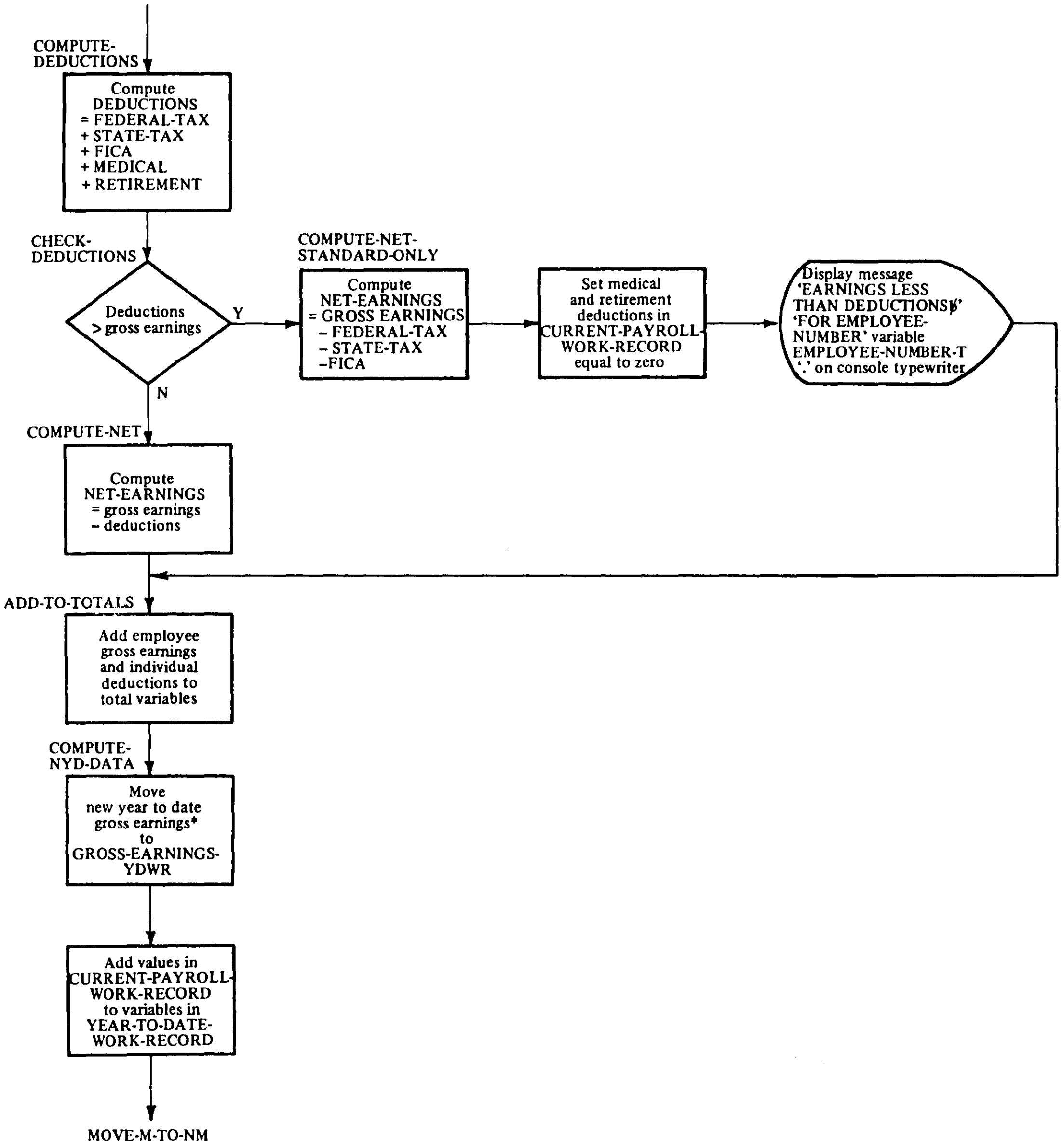
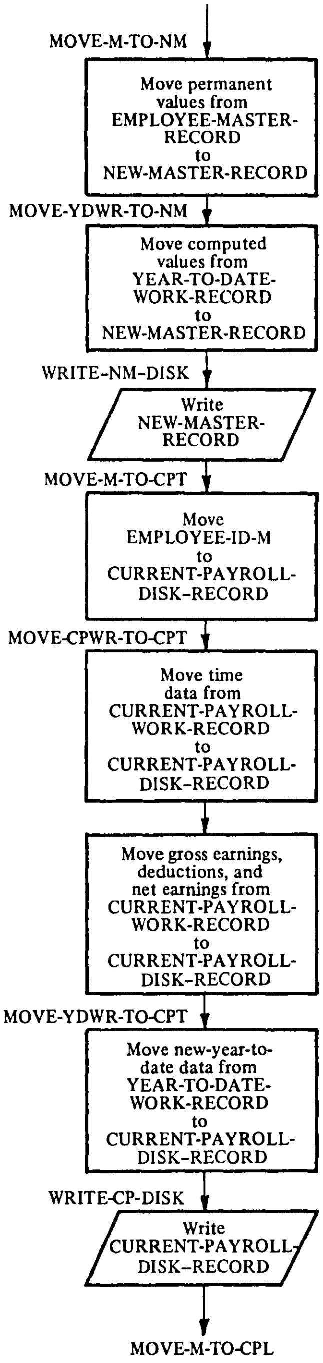
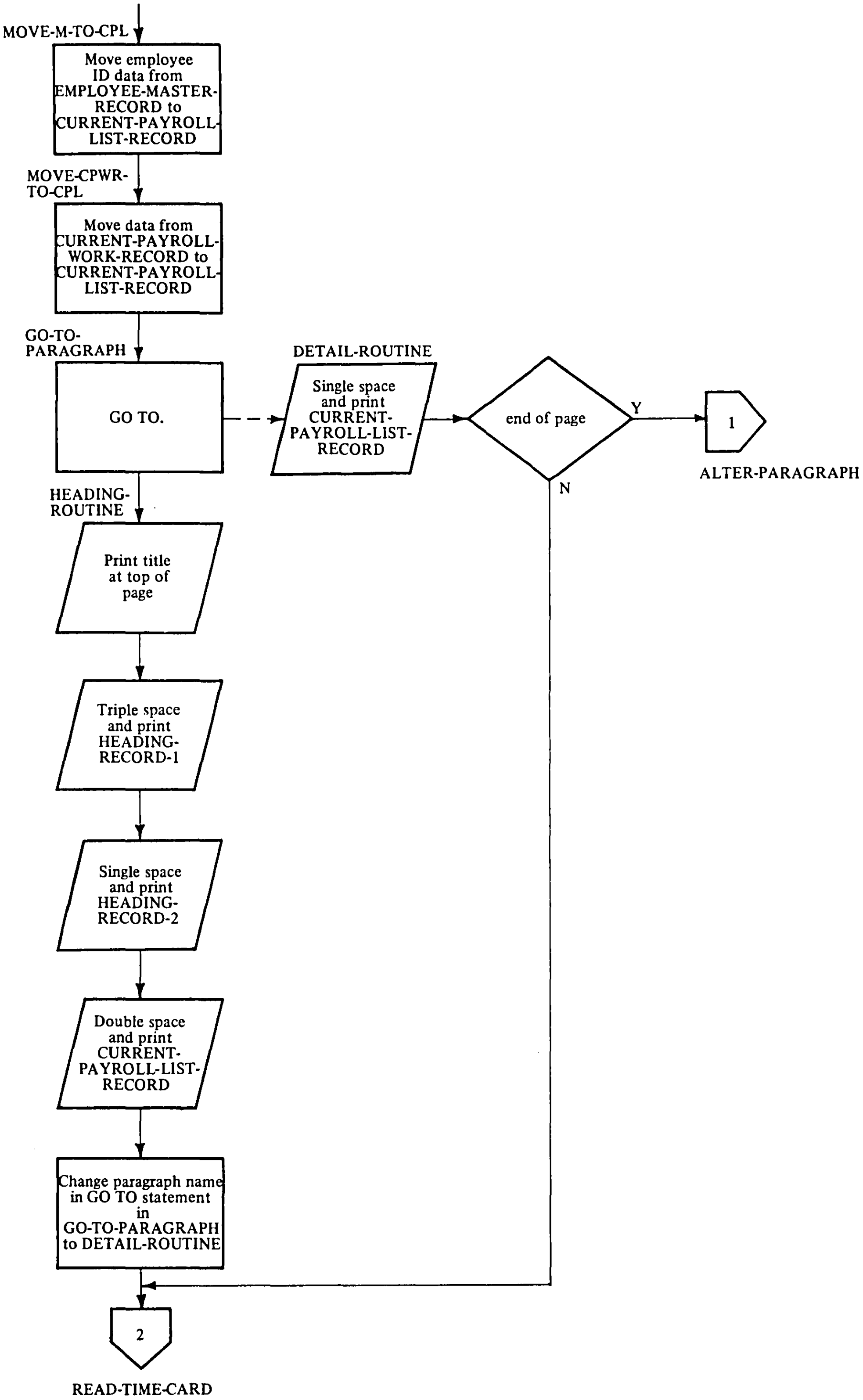

COMPUTE-DEDUCTIONS
Compute DEDUCTIONS = FEDERAL-TAX + STATE-TAX + FICA + MEDICAL + RETIREMENT
CHECK-DEDUCTIONS
Deductions > gross earnings
Y
N
COMPUTE-NET-STANDARD-ONLY
Compute NET-EARNINGS = GROSS EARNINGS - FEDERAL-TAX - STATE-TAX -FICA
Set medical and retirement deductions in CURRENT-PAYROLL-WORK-RECORD equal to zero
Display message 'EARNINGS LESS THAN DEDUCTIONSƀ' 'FOR EMPLOYEE-NUMBER' variable EMPLOYEE-NUMBER-T '.' on console typewriter
COMPUTE-NET
Compute NET-EARNINGS = gross earnings - deductions
ADD-TO-TOTALS
Add employee gross earnings and individual deductions to total variables
COMPUTE-NYD-DATA
Move new year to date gross earnings* to GROSS-EARNINGS-YDWR
Add values in CURRENT-PAYROLL-WORK-RECORD to variables in YEAR-TO-DATE-WORK-RECORD
MOVE-M-TO-NM

Figure 165


MOVE-M-TO-NM
Move permanent values from EMPLOYEE-MASTER-RECORD to NEW-MASTER-RECORD
MOVE-YDWR-TO-NM
Move computed values from YEAR-TO-DATE-WORK-RECORD to NEW-MASTER-RECORD
WRITE-NM-DISK
Write NEW-MASTER-RECORD
MOVE-M-TO-CPT
Move EMPLOYEE-ID-M to CURRENT-PAYROLL-DISK-RECORD
MOVE-CPWR-TO-CPT
Move time data from CURRENT-PAYROLL-WORK-RECORD to CURRENT-PAYROLL-DISK-RECORD
Move gross earnings, deductions, and net earnings from CURRENT-PAYROLL-WORK-RECORD to CURRENT-PAYROLL-DISK-RECORD
MOVE-YDWR-TO-CPT
Move new-year-to-date data from YEAR-TO-DATE-WORK-RECORD to CURRENT-PAYROLL-DISK-RECORD
WRITE-CP-DISK
Write CURRENT-PAYROLL-DISK-RECORD
MOVE-M-TO-CPL

```
0   0    1    1    2    2    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-M-TO-M.
           MOVE EMPLOYEE-ID-M TO EMPLOYEE-ID-NM.
           MOVE PAY-RATE-M TO PAY-RATE-NM.
           MOVE MEDICAL-M TO MEDICAL-NM.
           MOVE RETIREMENT-M TO RETIREMENT-NM.
       MOVE-YDWR TO-NM.
           MOVE SICKLEAVE-YDWR TO SICKLEAVE-YD-NM.
           MOVE VACATION-YDWR TO VACTION-YD-NM.
           MOVE GROSS-EARNINGS-YDWR TO GROSS-EARNINGS-YD-NM.
           MOVE FEDERAL-TAX-YDWR TO FEDERAL-TAX-YD-NM.
           MOVE STATE-TAX-YDWR TO STATE-TAX-YD-NM.
           MOVE FICA-YDWR TO FICA-YD-NM.
           MOVE RETIREMENT-YDWR TO RETIREMENT-YD-NM.
       WRITE-NM-DISK.
           WRITE NEW-MASTER-RECORD.
       MOVE-M-TO-CPD.
           MOVE EMPLOYEE-ID-M TO EMPLOYEE-ID-CPD.
       MOVE-CPWR-TO-CPD.
           MOVE REGULAR TO REGULAR-CPD.
           MOVE OVERTIME TO OVERTIME-CPD.
           MOVE SICKLEAVE TO SICKLEAVE-CPD.
           MOVE VACATION TO VACATION-CPD.
           MOVE GROSS-EARNINGS TO GROSS-EARNINGS-CPD.
           MOVE FEDERAL-TAX TO FEDERAL-TAX-CPD.
           MOVE STATE-TAX TO STATE-TAX-CPD.
           MOVE FICA TO FICA-CPD.
           MOVE MEDICAL TO MEDICAL-CPD.
           MOVE RETIREMENT TO RETIREMENT-CPD.
           MOVE NET-EARNINGS TO NET-EARNINGS-CPD.
       MOVE-YDWR-TO-CPD.
           MOVE-GROSS-EARNINGS-YDWR TO GROSS-EARNINGS-YD-CPD.
           MOVE FEDERAL-TAX-YDWR TO FEDERAL-TAX-YD-CPD.
           MOVE STATE-TAX-YDWR TO STATE-TAX-YD-CPD.
           MOVE FICA-YDWR TO FICA-YD-CPD.
           MOVE RETIREMENT-YDWR TO RETIREMENT-YD-CPD.
       WRITE-CP-DISK.
           WRITE-CURRENT-PAYROLL-DISK-RECORD.
```

Figure 166


MOVE-M-TO-CPL
Move employee ID data from EMPLOYEE-MASTER-RECORD to CURRENT-PAYROLL-LIST-RECORD
MOVE-CPWR-TO-CPL
Move data from CURRENT-PAYROLL-WORK-RECORD to CURRENT-PAYROLL-LIST-RECORD
GO-TO-PARAGRAPH
GO TO.
DETAIL-ROUTINE
Single space and print CURRENT-PAYROLL-LIST-RECORD
end of page
Y
1
ALTER-PARAGRAPH
N
HEADING-ROUTINE
Print title at top of page
Triple space and print HEADING-RECORD-1
Single space and print HEADING-RECORD-2
Double space and print CURRENT-PAYROLL-LIST-RECORD
Change paragraph name in GO TO statement in GO-TO-PARAGRAPH to DETAIL-ROUTINE
2
READ-TIME-CARD

```
0   0    1    1    2    2    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-M-TO-CPL.
           MOVE EMPLOYEE-NUMBER OF EMPLOYEE-ID-M
               TO EMPLOYEE-NUMBER-CPL.
           MOVE EMPLOYEE-NAME OF EMPLOYEE-ID-M
               TO EMPLOYEE-NAME-CPL.
           MOVE SOCIAL-SECURITY OF EMPLOYEE-ID-M
               TO SOCIAL-SECURITY-CPL.
       MOVE-CPWR-TO-CPL.
           MOVE REGULAR TO REGULAR-CPL.
           MOVE OVERTIME TO OVERTIME-CPL.
           MOVE SICKLEAVE TO SICKLEAVE-CPL.
           MOVE VACATION TO VACATION-CPL.
           MOVE GROSS-EARNINGS TO GROSS-EARNINGS-CPL.
           MOVE FEDERAL-TAX TO FEDERAL-TAX-CPL.
           MOVE STATE-TAX TO STATE-TAX-CPL.
           MOVE FICA TO FICA-CPL.
           MOVE MEDICAL TO MEDICAL-CPL.
           MOVE RETIREMENT TO RETIREMENT-CPL.
           MOVE NET-EARNINGS TO NET-EARNINGS-CPL.
```

Figure 167

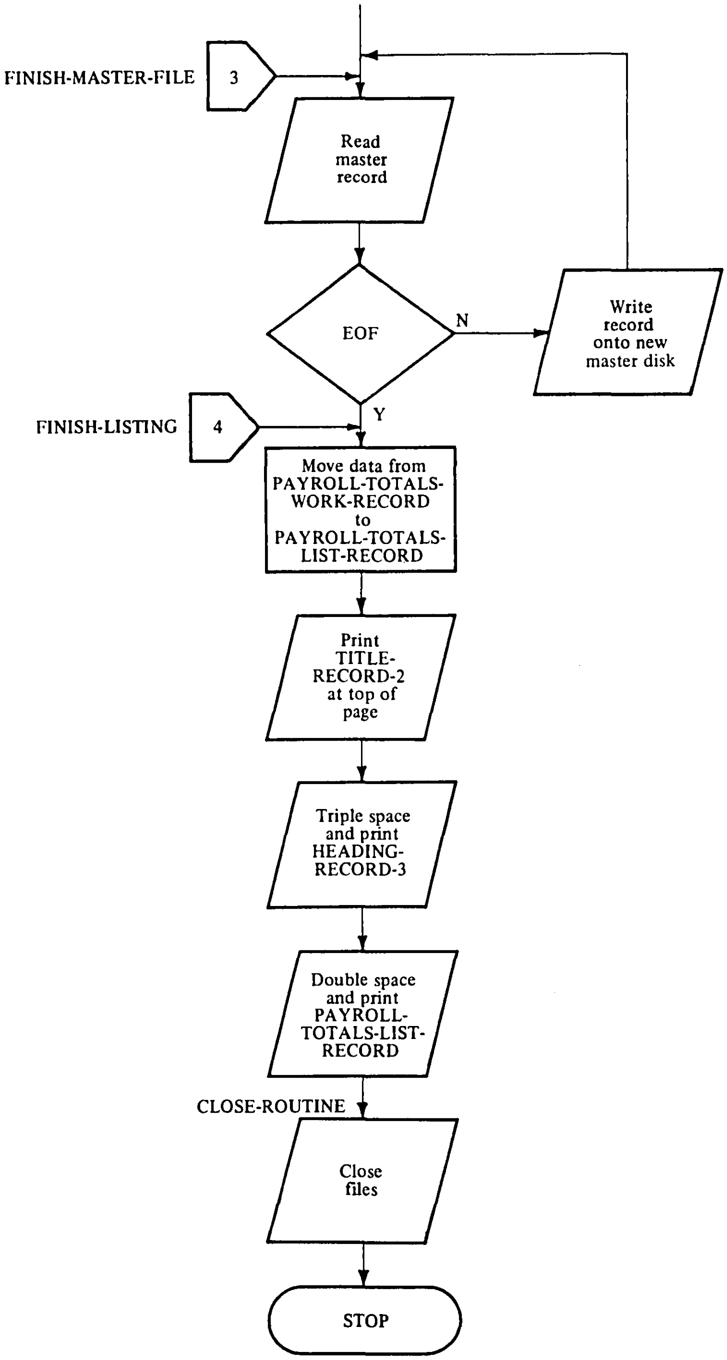

FINISH-MASTER-FILE
3
Read master record
EOF
N
Write record onto new master disk
Y
FINISH-LISTING
4
Move data from PAYROLL-TOTALS-WORK-RECORD to PAYROLL-TOTALS-LIST-RECORD
Print TITLE-RECORD-2 at top of page
Triple space and print HEADING-RECORD-3
Double space and print PAYROLL-TOTALS-LIST-RECORD
CLOSE-ROUTINE
Close files
STOP

Figure 168

9. Code the Procedure Division entries for the flow chart segment in Figure 161.

* * *

```
0   0    1    1    2    2    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.
       INITIAL-ROUTINE.
           OPEN INPUT EMPLOYEE-MASTER-FILE
               EMPLOYEE-TIME-FILE
               OUTPUT NEW-MASTER-FILE
               CURRENT-PAYROLL-DISK-FILE
               CURRENT-PAYROLL-LIST-FILE.
           DISPLAY 'ENTER PAY PERIOD END DATE '
               'IN FORMAT 011670.' UPON CONSOLE.
           ACCEPT END-DATE FROM CONSOLE.
           MOVE END-DATE
               TO DATE OF TITLE-RECORD-1
               DATE OF TITLE-RECORD-2.
       ALTER-PARAGRAPH.
           ALTER GO-TO-PARAGRAPH TO PROCEED TO
               HEADING-ROUTINE.
       READ-TIME-CARD.
           READ EMPLOYEE-TIME-FILE
               AT END GO TO FINISH-MASTER-FILE.
       FIND-MATCHING-MASTER-RECORD.
           READ EMPLOYEE-MASTER-FILE
               AT END DISPLAY
           'FINISHED MASTER ON EMPLOYEE-NUMBER '
                   EMPLOYEE-NUMBER-T'.'
                   UPON CONSOLE
                   GO TO FINISH-LISTING.
       COMPARE-RECORDS.
           IF EMPLOYEE-NUMBER OF EMPLOYEE-ID-M
               LESS THAN EMPLOYEE-NUMBER-T
               WRITE NEW-MASTER-RECORD
                   FROM EMPLOYEE-MASTER-RECORD
               GO TO
                   FIND-MATCHING-MASTER-RECORD.
           IF EMPLOYEE-NUMBER OF EMPLOYEE-ID-M
               EQUAL TO EMPLOYEE-NUMBER-T
               GO TO MOVE-DATA-M-TO-CPWR
               ELSE DISPLAY
                   'NO MASTER RECORD FOR '
                   EMPLOYEE-NUMBER-T '.'
                   UPON CONSOLE
                   READ EMPLOYEE-TIME-FILE
                       AT END GO TO
                           FINISH-MASTER-FILE
                   GO TO COMPARE-RECORDS.
```

---

10. After execution of COMPARE-RECORDS when matching records from EMPLOYEE-MASTER-FILE and EMPLOYEE-TIME-FILE have been read, appropriate values are to be moved from the input areas to computational variables. Then the program is to test whether any hours charged to sickleave or vacation exceed the unused hours of paid sickleave and vacation for that employee. If so, the number of hours charged is to be adjusted to the number of unused hours for use in computing gross earnings. The logic for this portion of the program is shown in Figure 162. Code the entries represented by Figure 162.

* * *

```
0   0    1    1    2    2    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-M-TO-CPWR.
           MOVE DEPENDENTS-M TO DEPENDENTS.
           MOVE PAY-RATE-M TO PAY-RATE.
           MOVE MEDICAL-M TO MEDICAL.
           MOVE RETIREMENT-M TO RETIREMENT.
       MOVE-DATA-T-TO-CPWR.
           MOVE REGULAR-T TO REGULAR.
           MOVE OVERTIME-T TO OVERTIME.
           MOVE SICKLEAVE-T TO SICKLEAVE.
           MOVE VACATION-T TO VACATION.
       MOVE-DATA-M-TO-YDWR.
           MOVE SICKLEAVE-YD-M
               TO SICKLEAVE-YDWR.
           MOVE VACATION-YD-M TO VACATION-YDWR.
           MOVE GROSS-EARNINGS-YD-M
               TO GROSS-EARNINGS-YDWR.
           MOVE FEDERAL-TAX-YD-M
               TO FEDERAL-TAX-YDWR.
           MOVE STATE-TAX-YD-M
               TO STATE-TAX-YDWR.
           MOVE FICA-YD-M TO FICA-YDWR.
           MOVE RETIREMENT-YD-M
               TO RETIREMENT-YDWR.
       CHECK-MAXIMUM-SICKLEAVE.
           IF SICKLEAVE EQUAL TO ZERO-TEST
               GO TO CHECK-MAXIMUM-VACATION
               ELSE COMPUTE UNUSED-HOURS
                   = MAXIMUM-HOURS
                   - SICKLEAVE-YDWR.
           IF SICKLEAVE GREATER THAN
               UNUSED-HOURS
               COMPUTE EXCESS-HOURS
                   = SICKLEAVE
                   - UNUSED-HOURS
               DISPLAY
                 'UNUSED SICKLEAVE EXCEEDED BY '
                 EXCESS-HOURS
                 ' HOURS FOR EMPLOYEE NUMBER '
                 EMPLOYEE-NUMBER-T '.'
                 MOVE UNUSED-HOURS TO SICKLEAVE.
       CHECK-MAXIMUM-VACATION.
           IF VACATION EQUAL TO ZERO-TEST
               GO TO COMPUTER-GROSS
               ELSE COMPUTE UNUSED-HOURS
                   = MAXIMUM-HOURS
                   - VACATION-YDWR.
           IF VACATION GREATER THAN
               UNUSED-HOURS
               COMPUTE EXCESS-HOURS
                   = VACATION
                   - UNUSED-HOURS
               DISPLAY
                  'UNUSED VACATION EXCEEDED BY '
                  EXCESS-HOURS
                  ' HOURS FOR EMPLOYEE NUMBER '
                  EMPLOYEE-NUMBER-T '.'
                  MOVE UNUSED-HOURS TO VACATION.
```

```
COMPUTE-GROSS.
    COMPUTER GROSS-EARNINGS
        = (REGULAR
        + TIME-AND-A-HALF * OVERTIME
        + SICKLEAVE
        + VACATION)
        * PAY-RATE.
```

---

11. Computation of federal income tax is a necessary step in every payroll program. The percentage method for computing the tax, which has been selected for the weekly payroll program you are coding, is described in Figure 159 along with illustrations of the corresponding COBOL variables. State tax is to be computed as a percentage of federal tax. The logic for the federal tax and state tax computations is shown in Figure 163. Follow the flow chart and code the appropriate 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..

       FIT-ROUTINE.
           COMPUTE TAXABLE
               = GROSS-EARNINGS
               - (EXEMPTION * DEPENDENTS).
           IF SINGLE
               GO TO COMPUTE-SINGLE-FIT
               ELSE GO TO COMPUTE-MARRIED-FIT.
           SET DM PM TO 1.
           SET DS PS TO 1.
       COMPUTE-SINGLE-FIT.
           IF TAXABLE GREATER THAN SINGLE-DOLLAR (DS) OF
               SINGLE-DOLLAR-OVERLAY AND
           TAXABLE NOT GREATER THAN SINGLE-DOLLAR (DS+1) OF
               SINGLE-DOLLAR-OVERLAY
           GO TO COMPUTE-FED-TAX.
           ADD 1 TO DS.
           IF DS GREATER THAN 8 GO TO COMPUTE-MARRIED-FIT.
           ADD 1 TO PS.
           GO TO COMPUTE-SINGLE-FIT.
       COMPUTE-FED-TAX.
           SET CS TO PS.
           COMPUTE FEDERAL-TAX = (TAXABLE-SINGLE-DOLLAR (DS))
               * SINGLE-PERCENT (PS)
               + SINGLE-CONSTANT (CS).
           GO TO COMPUTE STATE-TAX.
       COMPUTE-MARRIED-FIT.
           IF TAXABLE GREATER THAN MARRIED-DOLLAR (DM) OF
               MARRIED-DOLLAR-OVERLAY AND
           TAXABLE NOT GREATER THAN MARRIED-DOLLAR (DM+1) OF
               MARRIED-DOLLAR-OVERLAY
           GO TO COMPUTE-M-FED-TAX.
           ADD 1 TO DM.
           ADD 1 TO PM.
           IF DM IS GREATER THAN 8 GO TO COMPUTE-STATE-TAX.
           GO TO COMPUTE-MARRIED-FIT.
       COMPUTE-M-FED-TAX.
           SET CM TO PM.
           COMPUTE FEDERAL-TAX = (TAXABLE-MARRIED-DOLLAR (DM))
               * MARRIED-PERCENT (PM)
               + MARRIED-CONSTANT (CM).
       COMPUTE-STATE-TAX.
           COMPUTE STATE-TAX
               = STATE-TAX-PERCENT
               * FEDERAL-TAX.

----------------------------------------------------------------------
```

12. Computation of social security tax (FICA) is another necessary step in any payroll program. Code the entries for the flow chart segment in Figure 164.

*   *   *

```
0   0    1    1    2    2    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-GROSS-NYD.
           COMPUTE GROSS-EARNINGS-NYDWR
               = GROSS-EARNINGS-YDWR
               + GROSS-EARNINGS.
       CHECK-YD-MAX-FICA-EARNINGS.
           IF GROSS-EARNINGS-YDWR LESS THAN
               MAXIMUM-FICA-EARNINGS
               GO TO
                   CHECK-NYD-MAX-FICA-EARNINGS.
       COMPUTE-NO-MOVE-FICA.
           MOVE ZEROS TO FICA.
           GO TO COMPUTE-DEDUCTIONS.
       CHECK-NYD-MAX-FICA-EARNINGS.
           IF GROSS-EARNINGS-NYDWR GREATER THAN
               MAXIMUM-FICA-EARNINGS
               GO TO COMPUTE-FICA-FOR-BALANCE.
       COMPUTE-FICA.
           COMPUTE FICA
               = GROSS-EARNINGS
               * FICA-PERCENT.
           GO TO COMPUTE-DEDUCTIONS.
       COMPUTE-FICA-FOR-BALANCE.
           COMPUTE FICA
               = (MAXIMUM-FICA-EARNINGS
               - GROSS-EARNINGS-YDWR)
               * FICA-PERCENT.
```

(You have now completed the entries to compute federal tax, state tax, and social security tax. These are sometimes called standard deductions.)

---

13. Employees may choose to have insurance premiums, retirement contributions, union dues, community fund contributions, and investment deposits deducted from their earnings. These deductions are called voluntary deductions. Voluntary deductions from the employee master record are to be added to the standard deductions computed during the current payroll processing. The result is to be tested to determine whether it exceeds gross earnings. If so, only the standard deductions are to be subtracted from gross earnings. The computation of net earnings completes the values in the CURRENT-PAYROLL-WORK-RECORD for the employee. Appropriate values from this variable are to be added to payroll totals in PAYROLL-TOTALS-WORK-RECORD. Values from CURRENT-PAYROLL-WORK-RECORD are also to be added to year-to-date values in YEAR-TO-DATE-WORK-RECORD to provide new year-to-date values for NEW-MASTER-RECORD. Follow the flow chart in Figure 165 and code the entries for this portion of WEEKLY-PAYROLL.

*   *   *

```
0   0    1    1    2    2    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-DEDUCTIONS.
           COMPUTE DEDUCTIONS
               = FEDERAL-TAX
               + STATE-TAX
               + FICA
               + MEDICAL
               + RETIREMENT.
       CHECK-DEDUCTIONS.
           IF DEDUCTIONS GREATER THAN
               GROSS-EARNINGS
               GO TO COMPUTE-NET-STANDARD-ONLY.
       COMPUTE-NET.
           COMPUTE NET-EARNINGS
               = GROSS-EARNINGS
               - DEDUCTIONS.
           GO TO ADD-TO-TOTALS.
       COMPUTE-NET-STANDARD-ONLY.
           COMPUTE NET-EARNINGS
               = GROSS-EARNINGS
               - FEDERAL-TAX
               - STATE-TAX
               - FICA.
           MOVE ZEROS TO MEDICAL LIFE
               RETIREMENT.
           DISPLAY
               'EARNINGS LESS THAN DEDUCTIONS '
               'FOR EMPLOYEE NUMBER'
               EMPLOYEE-NUMBER-T '.'
               UPON CONSOLE.
       ADD-TO-TOTALS.
           ADD GROSS-EARNINGS TO GROSS-TOTAL.
           ADD FEDERAL-TAX TO FEDERAL-TAX-TOTAL.
           ADD STATE-TAX TO STATE-TAX-TOTAL.
           ADD FICA TO FICA-TOTAL.
           ADD MEDICAL TO MEDICAL-TOTAL.
           ADD RETIREMENT TO RETIREMENT-TOTAL.
           ADD NET-EARNINGS TO NET-TOTAL.
       COMPUTE-NYD-DATA.
           MOVE GROSS-EARNINGS-NYDWR
               TO GROSS-EARNINGS-YDWR.
           ADD SICKLEAVE TO SICKLEAVE-YDWR.
           ADD VACATION TO VACATION-YDWR.
           ADD FEDERAL-TAX TO FEDERAL-TAX-YDWR.
           ADD STATE-TAX TO STATE-TAX-YDWR.
           ADD FICA TO FICA-YDWR.
           ADD RETIREMENT TO RETIREMENT-YDWR.
```

You have now completed the computations in the program. The remainder of the program will move values to output areas and write records into the output files.

14. Since the coding to build and write the new master record and the current payroll disk record is fairly routine it has been provided for you in Figure 166 along with the flow chart segment. The coding to build the current payroll list record is also provided in Figure 167. You are to follow Figure 163 beginning with GO-TO-PARAGRAPH to code the entries to print the first report illustrated in Figure 156.

* * *

```
0   0    1    1    2    2    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.
           GO TO.
       HEADING-ROUTINE.
           WRITE CURRENT-PAYROLL-LIST-LINE
               FROM TITLE-RECORD-1.
           WRITE CURRENT-PAYROLL-LIST-LINE
               FROM HEADING-RECORD-1
               AFTER ADVANCING 3.
           WRITE CURRENT-PAYROLL-LIST-LINE
               FROM HEADING-RECORD-2
               AFTER ADVANCING 1.
           WRITE CURRENT-PAYROLL-LIST-LINE
               FROM CURRENT-PAYROLL-LIST-RECORD
               AFTER ADVANCING 2.
           ALTER GO-TO-PARAGRAPH TO PROCEED TO
               DETAIL-ROUTINE.
           GO TO READ-TIME-CARD.
       DETAIL-ROUTINE.
           WRITE CURRENT-PAYROLL-LIST-LINE
               FROM CURRENT-PAYROLL-LIST-RECORD
               AFTER ADVANCING 1
               AT END-OF-PAGE
                   GO TO ALTER-PARAGRAPH.
           GO TO READ-TIME-CARD.

-----------------------------------------------------------------------
```

15. Figure 168 shows the final steps in WEEKLY-PAYROLL, including printing the second report illustrated in Figure 156. Code the necessary entries.

*   *   *

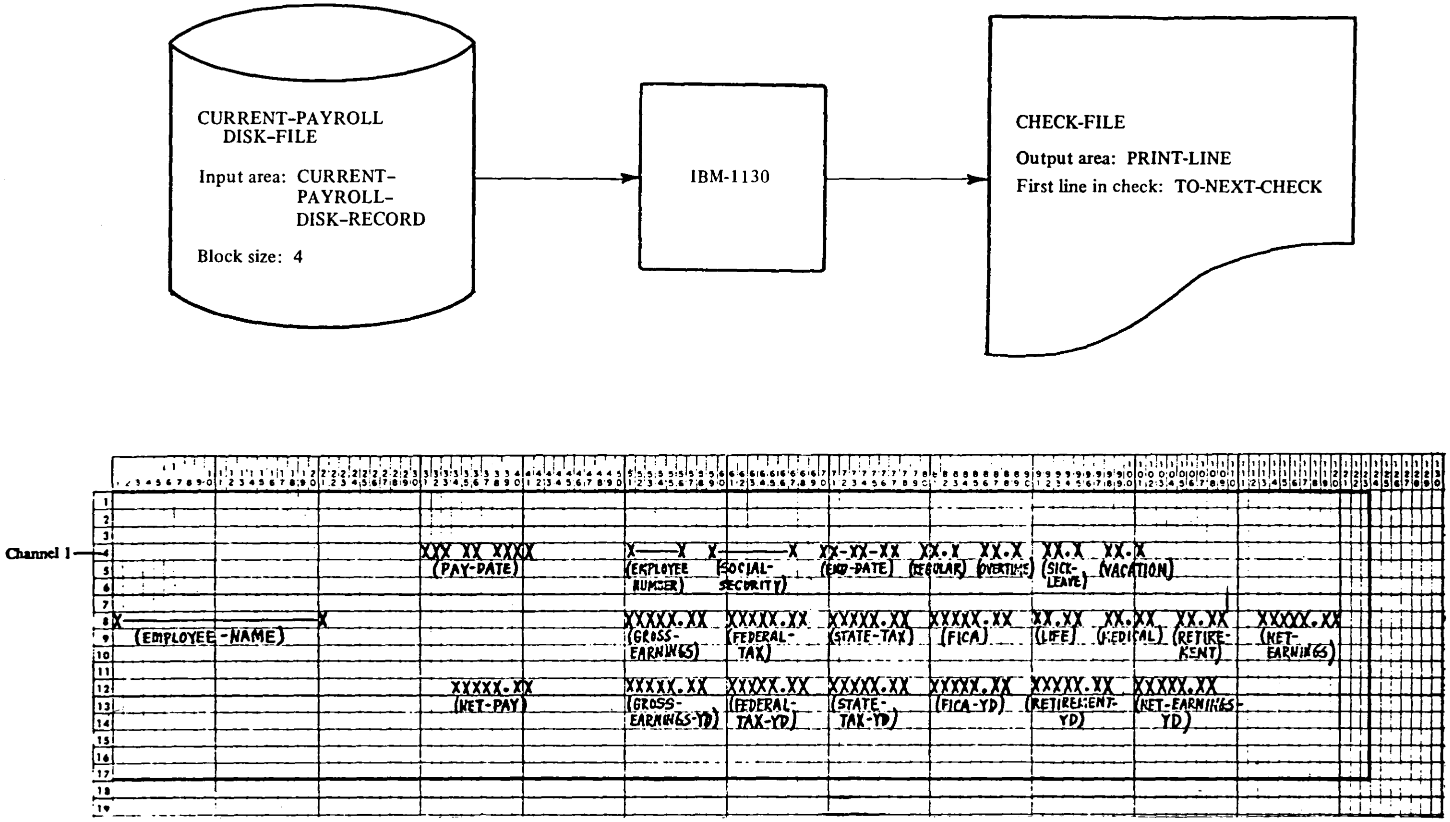
```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

        FINISH-MASTER-FILE.
            READ EMPLOYEE-MASTER-FILE
                AT END GO TO FINISH-LISTING.
            WRITE NEW-MASTER-RECORD
                FROM EMPLOYEE-MASTER-RECORD.
            GO TO FINISH-MASTER-FILE.
        FINISH-LISTING.
            MOVE GROSS-TOTAL TO GROSS-TOTAL-PTL.
            MOVE FEDERAL-TAX-TOTAL
                TO FEDERAL-TAX-TOTAL-PTL.
            MOVE STATE-TAX-TOTAL
                TO STATE-TAX-TOTAL-PTL.
            MOVE FICA-TOTAL TO FICA-TOTAL-PTL.
            MOVE MEDICAL-TOTAL
                TO MEDICAL-TOTAL-PTL.
            MOVE RETIREMENT-TOTAL
                TO RETIREMENT-TOTAL-PTL.
            MOVE NET-TOTAL TO NET-TOTAL-PTL.
            WRITE CURRENT-PAYROLL-LIST-LINE
                FROM TITLE-RECORD-2.
            WRITE CURRENT-PAYROLL-LIST-LINE
                FROM HEADING-RECORD-3
                AFTER ADVANCING 3.
            WRITE CURRENT-PAYROLL-LIST-LINE
                FROM PAYROLL-TOTALS-LIST-RECORD
                AFTER ADVANCING 2.
        CLOSE-ROUTINE.
            CLOSE EMPLOYEE-MASTER-FILE
                EMPLOYEE-TIME-FILE
                NEW-MASTER-FILE
                CURRENT-PAYROLL-DISK-FILE
                CURRENT-PAYROLL-LIST-FILE.
            STOP RUN.
```

You have now completed WEEKLY-PAYROLL. The current payroll disk file produced by this program is to be used to print checks and earnings statements in the program PRINT-CHECKS.

----

16. The program for printing payroll checks is described in Figure 169. Code the first two divisions of PRINT-CHECKS.

Figure 169


CURRENT-PAYROLL
DISK-FILE
Input area: CURRENT-PAYROLL-DISK-RECORD
Block size: 4
IBM-1130
CHECK-FILE
Output area: PRINT-LINE
First line in check: TO-NEXT-CHECK
Channel 1
XXX XX XXXX
(PAY-DATE)
X——X
(EMPLOYEE NUMBER)
X——X
(SOCIAL-SECURITY)
XX-XX-XX
(END-DATE)
XX.X
(REGULAR)
XX.X
(OVERTIME)
XX.X
(SICK-LEAVE)
XX.X
(VACATION)
X——X
(EMPLOYEE-NAME)
XXXXX.XX
(GROSS-EARNINGS)
XXXXX.XX
(FEDERAL-TAX)
XXXXX.XX
(STATE-TAX)
XXXXX.XX
(FICA)
XX.XX
(LIFE)
XX.XX
(MEDICAL)
XX.XX
(RETIREMENT)
XXXXX.XX
(NET-EARNINGS)
XXXXX.XX
(NET-PAY)
XXXXX.XX
(GROSS-EARNINGS-YD)
XXXXX.XX
(FEDERAL-TAX-YD)
XXXXX.XX
(STATE-TAX-YD)
XXXXX.XX
(FICA-YD)
XXXXX.XX
(RETIREMENT-YD)
XXXXX.XX
(NET-EARNINGS-YD)

After the CURRENT-PAYROLL-DISK-FILE is produced by WEEKLY-PAYROLL, a program called PRINT-CHECKS is to edit and rearrange the data in each record and print a check for each employee. A system flowchart is shown above along with the Printer Spacing Chart showing the desired format and editing for the checks. A printer carriage control tape will contain a punch in channel 1 corresponding to the first line position in each check. The value of NET-EARNINGS in the check is to be printed with a dollar sign preceding the leading non-zero digit and a decimal point. In the earnings statement leading zeros are to be suppressed and a decimal point is to be inserted into each dollar value.

* * *

```
0   0    1    1    2    2    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-CHECKS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       SPECIAL-NAMES.
           C01 IS TO-NEXT-CHECK.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CURRENT-PAYROLL-DISK-FILE
               ASSIGN TO DF-3-800-X.
           SELECT CHECK-FILE
               ASSIGN TO PR-1132-C.
```

---

17. The Data Division for PRINT-CHECKS is shown in Figure 170. The computer operation is to key in the number of the last check from the previous run. Then the program is to add 1 to this value for each check printed and display the number of the last check on the console typewriter at the end of the program. A program flow chart for PRINT-CHECKS is given in Figure 171. Coding for the paragraph BUILD-CHECK-LINES is given in Figure 171. Code the other portions of the Procedure Division in Figure 171.

```
0   0    1    1    2    2    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  CURRENT-PAYROLL-DISK-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 10 RECORDS.
       01  CURRENT-PAYROLL-DISK-RECORD.
           02  EMPLOYEE-NUMBERCPD PIC X(6).
           02  EMPLOYEE-NAME-CPD PIC X(21).
           02  SOCIAL-SECURITY-CPD PIC X(9).
           02  REGULAR-CPD PIC 99V9.
           02  OVERTIME-CPD PIC 99V9.
           02  SICKLEAVE-CPD PIC 99V9.
           02  VACATION-CPD PIC 99V9.
           02  GROSS-EARNINGS-CPD PIC 9(5)V99.
           02  FEDERAL-TAX-CPD PIC 9(5)V99.
           02  STATE-TAX-CPD PIC 9(5)V99.
           02  FICA-CPD PIC 9(5)V99.
           02  MEDICAL-CPD PIC 99V99.
           02  RETIREMENT-CPD PIC 99V99.
           02  NET-EARNINGS-CPD PIC 9(5)V99.
           02  GROSS-EARNINGS-YD-CPD PIC 9(5)V99.
           02  FEDERAL-TAX-YD-CPD PIC 9(5)V99.
           02  RETIREMENT-YD-CPD PIC 9(5)V99.
           02  NET-EARNINGS-YD-CPD PIC 9(5)V99.
       FD  CHECK-FILE
           LABEL RECORDS ARE OMITTED.
       01  PRINT-LINE PIC X(121).
       WORKING-STORAGE SECTION.
       77  ZERO-TEST PIC 9(5)V99 VALUE ZEROS.
       77  CHECK-NUMBER-DISP PIC 9(5).
       77  CHECK-NUMBER-COMP PIC 9(5) USAGE COMP.
       01  CHECK-LINE-1.
           02  FILLER PIC X(39) VALUE SPACES.
           02  PAY-DATE PIC X(11).
           02  FILLER PIC X(9) VALUE SPACES.
           02  EMPLOYEE-NUMBER PIC X(6).
           02  FILLER PIC XX VALUE SPACES.
           02  SOCIAL-SECURITY PIC X(9).
           02  FILLER PIC XX VALUE SPACES.
           02  END-DATE PIC X(8).
           02  FILLER PIC XX VALUE SPACES.
           02  REGULAR PIC ZZ.9.
           02  FILLER PIC XX VALUE SPACES.
           02  OVERTIME PIC ZZ.9.
           02  FILLER PIC XX VALUE SPACES.
           02  SICKLEAVE PIC ZZ.9.
           02  FILLER PIC XX VALUE SPACES.
           02  VACATION PIC ZZ.9.
           02  FILLER PIC X(13) VALUE SPACES.
```

```
01  CHECK-LINE-2
    02  FILLER PIC X(10) VALUE SPACES.
    02  EMPLOYEE-NAME PIC X(21).
    02  FILLER PIC X(19) VALUE SPACES.
    02  GROSS-EARNINGS PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  FEDERAL-TAX PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  STATE-TAX PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  FICA PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  MEDICAL PIC ZZ.99.
    02  FILLER PIC XX VALUE SPACES.
    02  RETIREMENT PIC ZZ.99.
    02  FILLER PIC XX VALUE SPACES.
    02  NET-EARNINGS PIC Z(5).99.
    02  FILLER PIC X(9) VALUE SPACES.
01  CHECK-LINE-3.
    02  FILLER PIC X(23) VALUE SPACES.
    02  NET-PAY PIC $(5).99
    02  FILLER PIC X(9) VALUE SPACES.
    02  GROSS-EARNINGS-YD PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  FEDERAL-TAX-YD PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  STATE-TAX PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  FICA-YD PIC Z(5).99.
    02  FILLER PIC X(15) VALUE SPACES.
    02  FILLER PIC XX VALUE SPACES.
    02  RETIREMENT-YD PIC Z(5).99.
    02  FILLER PIC XX VALUE SPACES.
    02  NET-EARNINGS-YD PIC Z(5).99.
    02  FILLER PIC X(8) VALUE SPACES.
```

Figure 170

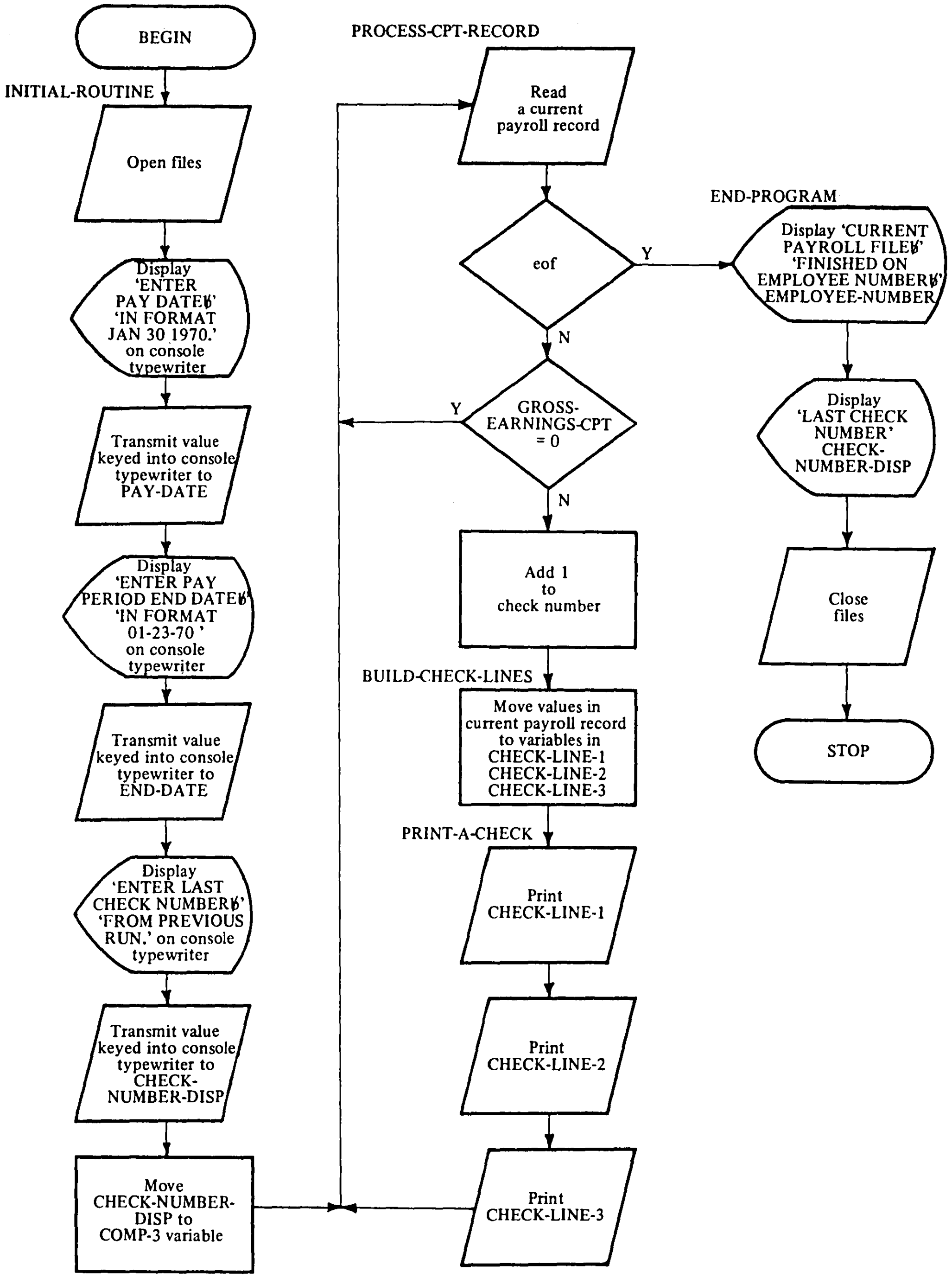

BEGIN
INITIAL-ROUTINE
Open files
Display 'ENTER PAY DATEƀ' 'IN FORMAT JAN 30 1970.' on console typewriter
Transmit value keyed into console typewriter to PAY-DATE
Display 'ENTER PAY PERIOD END DATEƀ' 'IN FORMAT 01-23-70 ' on console typewriter
Transmit value keyed into console typewriter to END-DATE
Display 'ENTER LAST CHECK NUMBERƀ' 'FROM PREVIOUS RUN.' on console typewriter
Transmit value keyed into console typewriter to CHECK-NUMBER-DISP
Move CHECK-NUMBER-DISP to COMP-3 variable
PROCESS-CPT-RECORD
Read a current payroll record
eof
Y
N
GROSS-EARNINGS-CPT = 0
Y
N
Add 1 to check number
BUILD-CHECK-LINES
Move values in current payroll record to variables in CHECK-LINE-1 CHECK-LINE-2 CHECK-LINE-3
PRINT-A-CHECK
Print CHECK-LINE-1
Print CHECK-LINE-2
Print CHECK-LINE-3
END-PROGRAM
Display 'CURRENT PAYROLL FILEƀ' 'FINISHED ON EMPLOYEE NUMBERƀ' EMPLOYEE-NUMBER
Display 'LAST CHECK NUMBER' CHECK-NUMBER-DISP
Close files
STOP

Figure 171

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

       BUILD-CHECK-LINES.
           MOVE EMPLOYEE-NUMBER-CPD
           TO EMPLOYEE-NUMBER.
           MOVE SOCIAL-SECURITY-CPD TO SOCIAL-SECURITY.
           MOVE REGULAR-CPD TO REGULAR.
           MOVE OVERTIME-CPD TO OVERTIME.
           MOVE SICKLEAVE-CPD TO SICKLEAVE.
           MOVE VACATION-CPD TO VACATION.
           MOVE EMPLOYEE-NAME-CPD TO EMPLOYEE-NAME.
           MOVE GROSS-EARNINGS-CPD TO GROSS-EARNINGS.
           MOVE FEDERAL-TAX-CPD TO FEDERAL-TAX.
           MOVE STATE-TAX-CPD TO STATE-TAX.
           MOVE FICA-CPD TO FICA.
           MOVE MEDICAL-CPD TO MEDICAL.
           MOVE RETIREMENT-CPD TO RETIREMENT.
           MOVE NET-EARNINGS-CPD TO NET-EARNINGS-NET-PAY.
           MOVE GROSS-EARNINGS-YD-CPD TO GROSS-EARNINGS-YD.
           MOVE FEDERAL-TAX-YD-CPD TO FEDERAL-TAX-YD.
           MOVE STATE-TAX-YD-CPD TO STATE-TAX-YD.
           MOVE FICA-YD-CPD TO FICA-YD.
           MOVE RETIREMENT-YD-CPD TO RETIREMENT-YD.
           MOVE NET-EARNINGS-YD-CPD TO NET-EARNINGS-YD.
```

Figure 172

*   *   *

```
0   0    1    1    2    2    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.
       INITIAL-ROUTINE.
           OPEN INPUT CURRENT-PAYROLL-DISK-FILE
               OUTPUT CHECK-FILE.
           DISPLAY 'ENTER PAY DATE '
               'IN FORMAT JAN 30 1970.'
               UPON CONSOLE.
           ACCEPT PAY-DATE FROM CONSOLE.
           DISPLAY 'ENTER PAY PERIOD END DATA'
               'IN FORMAT 01-23-70.'
               UPON CONSOLE.
           ACCEPT END-DATE FROM CONSOLE.
           DISPLAY 'ENTER LAST CHECK NUMBER '
               'FROM PREVIOUS RUN.'
               UPON CONSOLE.
           ACCEPT CHECK-NUMBER-DISP
               FROM CONSOLE.
           MOVE CHECK-NUMBER-DISP
               TO CHECK-NUMBER-COMP.
       PROCESS-CPT-RECORD.
           READ CURRENT-PAYROLL-DISK-FILE
               AT END GO TO END-PROGRAM.
           IF GROSS-EARNINGS-CPT
               EQUAL TO ZERO-TEST
               GO TO PROCESS-CPT-RECORD.
           ADD 1 TO CHECK-NUMBER-COMP.


       PRINT-A-CHECK.
           WRITE PRINT-LINE FROM CHECK-LINE-1
               AFTER ADVANCING TO-NEXT-CHECK.
           WRITE PRINT-LINE FROM CHECK-LINE-2
               AFTER ADVANCING 4 LINES.
           WRITE PRINT-LINE FROM CHECK-LINE-3
               AFTER ADVANCING 4 LINES.
           GO TO PROCESS-CPT-RECORD.
       END-PROGRAM.
           DISPLAY 'CURRENT PAYROLL FILE '
               'FINISHED ON EMPLOYEE-NUMBER '
               EMPLOYEE-NUMBER UPON CONSOLE.
           MOVE CHECK-NUMBER-COMP
               TO CHECK-NUMBER-DISP.
           DISPLAY 'LAST CHECK NUMBER '
               CHECK-NUMBER-DISP.
           CLOSE CURRENT-PAYROLL-DISK-FILE
               CHECK-FILE.
           STOP RUN.
```

SUMMARY:

You have now completed Lesson 31 in which you have combined many COBOL language features and programming techniques in coding programs to do payroll processing.

END OF LESSON 31

LESSON 32

LESSON 32 - SEQUENTIAL DISK PROCESSING

INTRODUCTION

In this lesson you will review the sequential disk file.

Disks (device number 2310 with 2315 disk cartridge) are frequently referred to as mass-storage devices. Updating a sequential file that is located on a mass-storage device is more efficient than updating a card file because a mass-storage file can be updated without creating a new file. After opening the file, it is possible to read a record, update it, and write it back into the same place in the file. Thus the file can be used for both input and output activity without closing and reopening it between operations.

When this is the case, the I-O option of the OPEN statement is used, as will subsequently be explained. The mass-storage file, however, may be differently organized, so that any record can be accessed merely by specifying the "key" or unique field that tells the system how to locate the desired record. This differs from sequential organization in that records can be accessed at random without accessing all previous records. When sequential organization is used, the records in a file are positioned sequentially in the order in which they are created and accessed sequentially in the order in which they were placed in the file. In this lesson you will learn to use sequential organization for files on mass-storage devices. Then in later lessons you will learn to use files with other than sequential organization.

The first program you will write in this lesson will be one to create a sequential disk file from input cards. In the next lesson you will write a program to update the file you have created by reading records, making changes in the records, and replacing them in the file.

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

ASSIGN clause variation for standard sequential disk files
INVALID option of the WRITE statement

This lesson will require approximately one half hour.

1. Recall that in order to specify that a file will be on a mass-storage device, the appropriate device number must be specified in the ASSIGN clause. According to Figure 21, device numbers that are associated with mass-storage devices are:

   a. DF-FILENUMBER-NUMBERREC-[-X]

   b. 1442

   c. 1132

   d. 2400

*   *   *

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..

           SELECT OFILE
               ASSIGN TO RD-1442.

           SELECT OFILE
               ASSIGN TO DF-1-800.
```

The first statement shown above specifies that the sequential file, OFILE, is to be on cards and the second specifies a disk. Which elements are different in the ASSIGN clause for a sequential disk file?

*   *   *

System name

----

3. Figure 21 shows that the class indicator of a file on a mass-storage device may be either DF or RD. Since sequential disk files are used in much the same way as card files, the class indicator ordinarily used for a disk file would be:

   a. DF

   b. RD

*   *   *

a

----

4. PAYFILE is to be a sequential file stored on a mass-storage device. Specify that the file is to be on a 2310 disk drive, filenumber 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 PAYFILE
               ASSIGN TO DF-1-600-X.
```

Guide to File Description Entries

Device Type	Labels *	Blocking
Card	N	N
Printer	N	N
Disk Sequential	R	O
Disk Random	R	O

N - Not permitted
O - Optional
R - Required

* LABEL RECORDS clause is always required.

Figure 173

5. This variation of the ASSIGN clause is the only new coding in the Environment Division that you must know in order to use a sequential disk file.

Figure 173 is a guide to file description entries in the Data Division. Whenever no labels are used, you write the LABEL RECORDS ARE OMITTED clause. Whenever labels are used, you write the LABEL RECORDS ARE STANDARD clause.

Figure 173 indicates that a correct file description entry for a sequential disk file 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..

       FD  PAYFILE
           LABEL RECORDS ARE STANDARD.
```

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..

       FD  PAYFILE
           LABEL RECORDS ARE OMITTED.
```

*   *   *

a
(All files on mass-storage devices must have standard labels.)

---

6. The records in PAYFILE are to be blocked in groups of four. Write an FD entry for this 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  PAYFILE
           BLOCK CONTAINS 4 RECORDS
           LABEL RECORDS ARE STANDARD.
```

---

7. DISK-SEQ is to be a sequential utility file located on a 2315 disk. The file is to have standard labels. Records in the file are to be blocked in groups of six.

1) Write a SELECT entry for the Environment Division for file DISK-SEQ.

2) Write an FD entry for file DISK-SEQ for the Data 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..

           SELECT DISK-SEQ
               ASSIGN TO DF-2-900-X.
```

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..

       FD  DISK-SEQ
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 6 RECORDS.
```

---

8. To create any file you must use:

a. an output file.

b. a READ statement.

c. a WRITE statement.

*   *   *

a, c

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..

           WRITE DISKDATA
               INVALID GO TO ENDING.
```

The statement above includes the INVALID option that is required in every WRITE statement that refers to an OUTPUT file on a mass-storage device. The INVALID option must be used when a:

a. sequential card file is being created.

b. sequential disk file is being created.

*   *   *

b

---

10. When a sequential disk file is being created, the statement following the reserved word INVALID is activated when an attempt is made to write beyond the limit of space reserved for the file. An appropriate action to specify in an INVALID option might be a branch to a routine that will:

a. display a message such as FILE FILLED.

b. execute another WRITE statement.

*　　　*　　　*

a
(The amount of space reserved for the file would be specified by you on control cards according to the operating system in your installation. Calculation of the amount of space required for a file is not a part of the COBOL language and as such is not included in this course.)

----

11. When a sequential disk file is being created, the statement that follows the reserved word INVALID is activated when:

a. the space reserved for the file is filled.

b. the data in the output record is invalid.

*　　　*　　　*

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..

        WRITE record-name [FROM identifier-1]

            INVALID imperative-statement
```

The format of a WRITE statement is shown above. Which of the following could be used to create a sequential disk 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..

        WRITE DISKFACT
            INVALID GO TO FULL.
```

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 DISKFACT FROM INRECORD
            INVALID GO TO FULL.
```

*　　　*　　　*

Either

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..

           WRITE record-name [FROM identifier-1]

               INVALID imperative-statement
```

Which of the following INVALID options 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..

           INVALID COPY 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..

           INVALID PERFORM EXACT
```

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..

           INVALID MULTIPLY A BY B
```

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..

           INVALID IF A IS GREATER
               THAN B
```

*   *   *

b, c

---

14. The COBOL word INVALID must be followed by:

a. a conditional statement.

b. an imperative statement.

c. a statement such as COPY OUTPUT-ENTRY.

*   *   *

b

---

15. Which of the following statements could be used to create a file on a mass-storage device?

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 OUTREC FROM INREC.
```

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 OUTREC
            INVALID GO TO END-ALL.
```

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 OUTREC
            INVALID GO TO END-ALL.
```

*   *   *

b

---

16. A sequential disk file called PAYFILE is being created from records called PAYROLL. Records from a card reader are accessed with 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 INDATA INTO PAYROLL.
```

If the output file is filled before the input file INDATA has been processed, control is to be transferred to a paragraph called END-ROUTINE. Write the output statement that would be used to create PAYFILE.

*   *   *

```
0  0    1    1    2    2    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
            INVALID GO TO END-ROUTINE.
```

---

17. As a programmer you might be asked to write a program to create a sequential disk file of master records. Write a statement to be used in this program, using MASTER-RECORD for the records that are to be placed in the file DISKFILE. Specify in the statement that control is to be transferred to a routine called FILE-FULL when the file is filled.

*   *   *

```
0   0    1    1    2    2    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 MASTER-RECORD
               INVALID GO TO FILE-FULL.
```

---

18. Write a single statement to:

    1) move a record from IN-REC to OUT-REC.

    2) place OUT-REC in a sequential disk file.

    3) transfer control to FILE-FULL when all the space reserved for the file has been filled.

*   *   *

```
0   0    1    1    2    2    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 OUT-REC FROM IN-REC
               INVALID GO TO FILE-FULL.
```

Figure 174 represents part of a program that will be used to create a sequential disk file from an input deck of cards. In the next sequence of frames you will write the additional statements that will complete the 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. CREATED.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CARD-FILE ASSIGN TO RD-1442.

       DATA DIVISION.
       FILE-SECTION.
       FD CARDFILE
           LABEL RECORDS ARE OMITTED.
       01  CARDLIST.
           02  SALESMAN PIC X(7).
           02  S-ADDRESS PIC X(33).
           02  SALES-HISTORY PIC 99.
           02  COMMISSION-RATE PIC V99.
           02  FILLER PIC X(36).
       PROCEDURE DIVISION.
       BEGIN.

       CREATE FILE.

           GO TO CREATE-FILE.
       PRE-CLOSE.
           DISPLAY 'DISK AREA IS FILLED'
               UPON CONSOLE.
       TERMINATE.
       CLOSE CARDFILE SEQ-DISK.
       STOP RUN.
```

Figure 174

19.

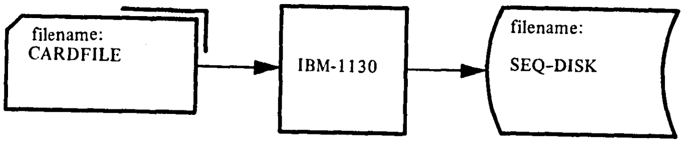


Figure 175

The system flow chart above shows the files that are used in the program in Figure 174. Write a statement that would complete the FILE-CONTROL paragraph.

```
                                          *        *        *

0   0    1    1    2    2    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 SEQ-DISK
               ASSIGN TO DF-2-800-X.
```

----

20. Now write the File Section entries for SEQ-DISK in Figure 174.

```
                                          *        *        *

0   0    1    1    2    2    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  SEQ-DISK
           LABEL RECORDS ARE STANDARD.
       01  MASTER1 PIC X(80).
```

----

21. Complete paragraph BEGIN in the Procedure Division of Figure 174 by writing a statement to prepare the files CARDFILE and SEQ-DISK so that CARDFILE can be used to create SEQ-DISK.

```
                                          *        *        *

0   0    1    1    2    2    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 CARDFILE
               OUTPUT SEQ-DISK.
```

----

22. Now you are ready to write the statements that will actually create the file by transferring data from the input file to the output file. The records in both files are the same length, therefore moving data will require no additional statements. Write the statements (two or three) to complete paragraph CREATE-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..

          READ CARDFILE
              AT END GO TO TERMINATE.
          MOVE CARDLIST TO MASTER1.
          WRITE MASTER1
              INVALID GO TO PRE-CLOSE.
```

(If INTO MASTER1 is used in the READ statement or FROM CARDLIST is used in the WRITE statement, the MOVE statement may be omitted.)

---

23. A program to create a sequential disk file always differs from one to create printed report in the:

a. ASSIGN clause.

b. level 01 entry.

c. FD entry.

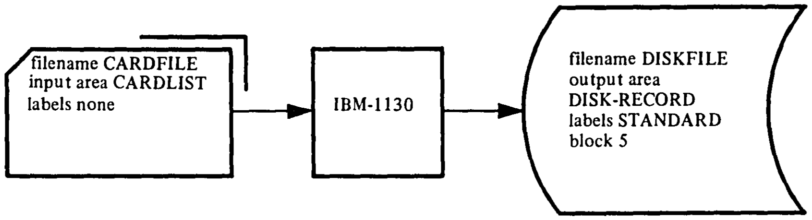
d. FILE-CONTROL paragraph.

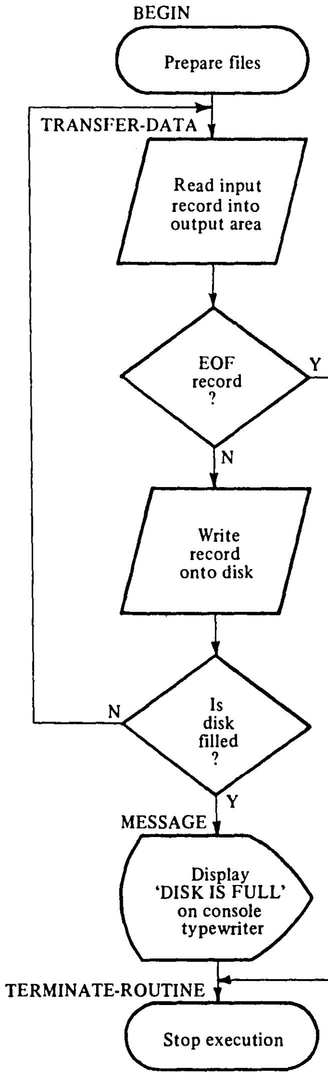
e. READ statement.

f. WRITE statement.

*　　*　　*

a, d, e, f


filename CARDFILE
input area CARDLIST
labels none
IBM-1130
filename DISKFILE
output area
DISK-RECORD
labels STANDARD
block 5


BEGIN
Prepare files
TRANSFER-DATA
Read input record into output area
EOF record ?
Y
N
Write record onto disk
Is disk filled ?
N
Y
MESSAGE
Display 'DISK IS FULL' on console typewriter
TERMINATE-ROUTINE
Stop 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. WYANDOTTE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.

       DATA DIVISION.
       FILE SECTION.
       FD  DISKFILE
           BLOCK CONTAINS 4 RECORDS
           LABEL RECORDS ARE STANDARD.
       01  DISK-RECORD
           02  PERSONAL.
               03  NAME PIC X(21).
               03  CUSTOMER-NUMBER PIC X(6).
               03  ADDRESS-HOME.
                   04  STREET PIC X(15).
                   04  CITYSTATE PIC X(15).
           02  PAYRECORD.
               03  YEAR-OPENED PIC 99.
               03  MAXIMUM CREDIT PIC S9999V99 USAGE COMP.
               03  MAXIMUM-BILL PIC S9999V99 USAGE COMP.
               03  BALANCE-DUE PIC S9999V99 USAGE COMP.
               03  PAYCODE PIC 9.
                   88  BAD VALUE 1.
                   88  POOR VALUE 2.
                   88  SLOW VALUE 3.
                   88  AVERAGE VALUE 4.
                   88  GOOD VALUE 5.
                   88  EXCELLENT VALUE 6.
                   88  NONE VALUE 7.
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CARDLIST.
           02  IPERSONAL.
               03  NAME PIC X(21).
               03  CUSTOMER-NUMBER PIC X(6).
               03  ADDRESS-HOME.
                   04  STREET PIC X(15).
                   04  CITYSTATE PIC X(15).
           02  PAYRECORD.
               03  IYEAR-OPENED PIC 99.
               03  IMAXIMUM-CREDIT PIC S9999V99.
               03  IMAXIMUM-BILL PIC S9999V99.
               03  IBALANCE-DUE PIC S9999V99.
               03  IPAYCODE PIC 9.
```

Figure 176

24. Wyandotte Water Works has requested a program to convert its master card file to a master disk file with the same organization. The master file contains records for all water users in the city of Wyandotte. Figure 176 gives most of the first three divisions of program WYANDOTTE, along with a system flow chart and a program flow chart for creating a sequential disk file. Write this:

1) FILE-CONTROL paragraph, referring to the system flow chart.

2) Procedure Division, following the program flow chart.

* * *

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..

       FILE-CONTROL.
           SELECT DISKFILE
               ASSIGN TO DF-1-700-X.                      (1)
           SELECT CARDFILE
               ASSIGN TO RD-1442.
```

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 CARDFILE
               OUTPUT DISKFILE.
       TRANSFER-DATA.
           READ CARDFILE
               AT END GO TO TERMINATE-ROUTINE.
           MOVE IPERSONAL TO PERSONAL.
           MOVE IYEAR-OPENED TO YEAR-OPENED.
           MOVE IMAXIMUM-CREDIT TO MAXIMUM-CREDIT.
           MOVE IMAXIMUM-BILL TO MAXIMUM-BILL.
           MOVE IBALANACE-DUE TO BALANCE-DUE.
           MOVE IPAYCODE TO PAYCODE.
           WRITE DISK-RECORD                              (9)
               INVALID GO TO MESSAGE.
           GO TO TRANSFER-DATA.
       MESSAGE.
           DISPLAY 'DISK IS FULL'
               UPON CONSOLE.
       TERMINATE-ROUTINE.
           CLOSE CARDFILE DISKFILE.
           STOP RUN.
```

SUMMARY

In this lesson you have learned to create a sequential file on a mass-storage device by opening the file as OUTPUT.

In the next lesson you will open the same sequential disk file as I-O to update the file, using it for both input and output operations.

END OF LESSON 32

LESSON 33

LESSON 33 - SEQUENTIAL DISK UPDATING

INTRODUCTION

In this lesson you will learn to open a sequential disk file as I-O in order to update the file. You will access the records as though you were reading input data, and you will write records on disk as though the disk were an output file. A major part of the lesson will be the updating of a sequential disk file.

Specific COBOL language feature you will learn in this lesson is:

I-O option of the OPEN 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..
           OPEN I-O DISK-FILE.
```

After a sequential disk file is created, it may be maintained by updating records and then placing the records back into the file. This can be done only if the file has been opened with the I-O option as shown above. The I-O option must be used when a sequential disk file is to be:

a. created.

b. used for both input and output operations.

* * *

b

---

2. The I-O option of the OPEN statement may be specified only for files that are stored on mass-storage devices. Match the following OPEN options with the types of files with which they may be used.

1) Card	a. INPUT
2) Disk	b. OUTPUT
3) Printer	c. I-O

* * *

1) a, b
2) a, b, c
3) 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..
           OPEN I-O UP-FILE.
```

The above statement would be acceptable if UP-FILE were:

a. on a disk.

b. not yet created.

c. to be updated.

* * *

a, c

---

4. In order to read a record, make changes in its fields, and write the record back into the file, the file must have been opened with the .... option and stored on a .... device.

* * *

I-O
mass-storage, direct access, or disk

---

5. Write a statement to open the disk file SEQ-DISK for updating.

* * *

```
0   0    1    1    2    2    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 I-O SEQ-DISK.
```

---

6. After a sequential file has been opened as I-O, a record is accessed just as in an input file. A programmer would access a record with a .... statement. After updating, the record is placed back into a sequential file as though it were being placed in an output file. The programmer would place a record back into an I-O file with a .... statement.

* * *

READ
WRITE

---

7. Match each OPEN option below with the statement(s) that can be used to refer to a file opened that way.

1) READ	a. INPUT
2) WRITE	b. OUTPUT
	c. I-O

* * *

1) a, c
2) b, c

---

8. Which statement would be used to place a record back into a sequential file that was opened as I-O?

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 RECORD-1
               INVALID GO TO EOJ.
```

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 RECORD-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..

           WRITE I-O RECORD-1.
```

* * *

a

---

9. Write a statement to place a record called PAYLIST back into a sequential I-O file named LISTING.

* * *

```
0   0    1    1    2    2    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 PAYLIST, INVALID GO TO THERE.
```

---

10. A CLOSE statement for a file that was opened as I-O is written like a CLOSE statement for any other file. Which statement could close an I-O file named LISTING?

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..

           CLOSE I-O LISTING.
```

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..

           CLOSE LISTING.
```

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 LISTING I-O.
```

*   *   *

b

----

11. Write a statement to close a file called IOFILE.

*   *   *

```
0   0    1    1    2    2    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 IOFILE.
```

----

12. As a programmer employed by a large department store, you are required to update the sequential disk file MASTER1 annually. One of the steps you will have to perform is add 12 months to the element TIME-CUSTOMER in each record in the file. The record variable MAS-REC is to be used to refer to the records in the file. Write the Procedure Division using the paragraph names BEGIN, PROCESS, and EOJ.

*   *   *

```
0   0    1    1    2    2    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 I-O MASTER1.
       PROCESS.
           READ MASTER1 AT END GO TO EOJ.
           ADD 12 TO TIME-CUSTOMER.
           WRITE MAS-REC, INVALID GO TO EOJ.
           GO TO PROCESS.
       EOJ.
           CLOSE MASTER1.
           STOP RUN.
```

13.

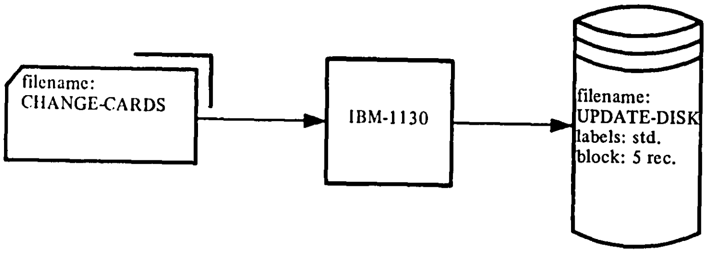


Figure 177

The system flow chart above shows the files that are used in the updating program in Figure 178. You will provide the missing portions of the program in the next few frames. First complete the Environment Division by writing entries for the files shown in the flow chart above.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DISK-UPDATE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-1130.
OBJECT-COMPUTER. IBM-1130.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

DATA DIVISION.
FILE SECTION.
FD  CHANGE-CARDS
    LABEL RECORDS ARE OMITTED.
01  CHANGE-RECORD.
    02  PART-RECEIVED PIC X(7).
    02  NUMBER-RECEIVED PIC 9(4).
    02  FILLER PIC X(69).


01  MASTER-DISK-RECORD.
    02  PART-NUMBER PIC X(7).
    02  PART-DESCRIPTION PIC X(40).
    02  PART-PRICE PIC 999V99.
    02  QUANTITY-ON-HAND PIC 9(6).
    02  QUANTITY-ON-ORDER PIC 9(6).
WORKING-STORAGE SECTION.
77  ON-HAND PIC 9(6) USAGE COMP.
77  ON-ORDER PIC 9(6) USAGE COMP.
77  RECEIVED PIC 9(4) USAGE COMP.

PROCEDURE DIVISION.
BEGIN.
```

```
       REVISIONS.

       CHECK-FILE.

           IF PART-NUMBER LESS THAN PART-RECEIVED
               GO TO CHECK-FILE.
           IF PART-NUMBER GREATER THAN PART-RECEIVED
               DISPLAY PART-RECEIVED
               'OUT OF ORDER OR NOT HERE'
               UPON CONSOLE GO TO STOP-RUN.
           MOVE QUANTITY-ON-HAND TO ON-HAND.
           MOVE QUANTITY-ON-ORDER TO ON-ORDER.
           MOVE NUMBER-RECEIVED TO RECEIVED.
           SUBTRACT RECEIVED FROM ON-ORDER.
           ADD RECEIVED TO ON-HAND.
           MOVE ON-HAND TO QUANTITY-ON-HAND.
           MOVE ON-ORDER TO QUANTITY-ON-ORDER.

           GO TO REVISIONS.
       STOP-RUN.

           STOP RUN.
```

Figure 178

*   *   *

```
0   0    1    1    2    2    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 CHANGE-CARDS
               ASSIGN TO RD-1442.
           SELECT UPDATE-DISK
               ASSIGN TO DF-1-500-X.
```

---

14. Now write the file description entry for the disk file. You may want to refer to the system flow chart for this file in the preceding 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..

       FD  UPDATE-DISK
           LABEL RECORDS ARE STANDARD.
```

---

15. There are several statements missing in the Procedure Division in Figure 178. Write Procedure Division entries to do the following:

   1) In paragraph BEGIN prepare the file CHANGE-CARDS to be used for input and the file UPDATE-DISK to be updated.

   2) In paragraph REVISIONS access an input record and transfer control to STOP-RUN when the end-of-file card is read.

   3) In paragraph CHECK-FILE access a record from the disk file and transfer control to STOP-RUN after all records have been read.

* * *

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..

           OPEN INPUT CHANGE-CARDS
               I-O UPDATE-DISK.
```

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 CHANGE-CARDS
               AT END GO TO STOP-RUN.
```

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..

           READ UPDATE-DISK
               AT END GO TO STOP-RUN.
```

---

16. To complete the program in Figure 178, write statements to:

   1) cause an updated record to be placed back into the disk file.

   2) close all files.

* * *

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 MASTER-1 INVALID GO TO 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..

           CLOSE CHANGE-CARDS UPDATE-DISK.
```

---

17. Now write an entire program WYWORKS to update the sequential disk file that you created for Wyandotte Water Works earlier in this lesson by inserting address changes into specific records. The flow charts in Figure 179 will provide you with a guide to files and logic, and the library texts in Figure 180 will provide the data names to be used in the Procedure Division.

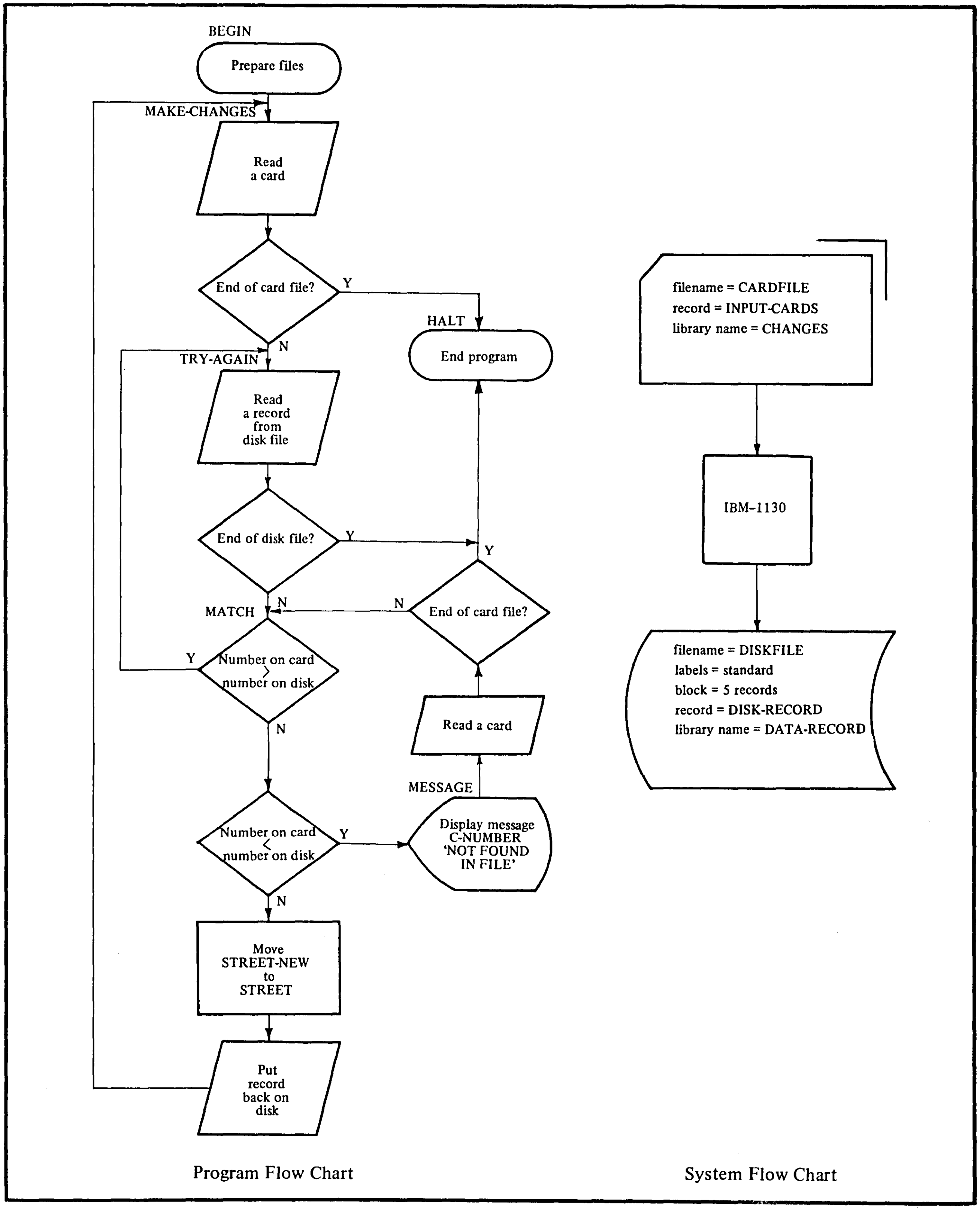


Figure 179

Library name: DATA-RECORD

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  PERSONAL.
               03  NAME PIC X(21).
               03  CUSTOMER-NUMBER PIC X(6).
               03  ADDRESS-HOME.
                   04  STREET PIC X(15).
                   04  CITYSTATE PIC X(15).
           02  PAYRECORD.
               03  YEAR-OPENED PIC 99.
               03  MAXIMUM-CREDIT PIC S9999V99 USAGE COMP.
               03  MAXIMUM-BILL PIC S9999V99 USAGE COMP.
               03  BALANCE-DUE PIC S9999V99 USAGE COMP.
               03  PAYCODE PIC 9.
                   88  BAD VALUE 1.
                   88  POOR VALUE 2.
                   88  SLOW VALUE 3.
                   88  AVERAGE VALUE 4.
                   88  GOOD VALUE 5.
                   88  EXCELLENT VALUE 6.
                   88  NONE VALUE 7.
```

Library name: CHANGES

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  C-NUMBER PIC X(6).
           02  STREET-NEW PIC X(15).
           02  FILLER PIC X(59).
```

Figure 180

* * *

```
0   0    1    1    2    2    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. WYWORKS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT DISKFILE
               ASSIGN TO DF-1-500.                     (1)
           SELECT CARDFILE
               ASSIGN TO RD-1442.
       DATA DIVISION.
       FILE SECTION.
       FD  DISKFILE
           LABEL RECORDS ARE STANDARD.
       01  DISK-RECORD COPY DATA-RECORD.
       FD  CARDFILE
           LABEL RECORDS ARE OMITTED.
       01  INPUT-CARDS COPY CHANGES.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT CARDFILE
               I-O DISKFILE.                           (25)
       MAKE-CHANGES.
           READ CARDFILE AT END GO TO HALT.
       TRY-AGAIN.
           READ DISKFILE AT END GO TO HALT.
       MATCH.
           IF C-NUMBER GREATER THAN
               CUSTOMER-NUMBER
               GO TO TRY-AGAIN.
           IF C-NUMBER LESS THAN
               CUSTOMER-NUMBER
               GO TO MESSAGE.
           MOVE STREET-NEW TO STREET.
           WRITE DISK-RECORD INVALID DISPLAY 'BEYOND LIMIT OF FILE'.
           GO TO MAKE-CHANGES.
       MESSAGE.
           DISPLAY C-NUMBER
               'NOT FOUND IN FILE' UPON CONSOLE.
           READ CARDFILE AT END GO TO HALT.
           GO TO MATCH.
       HALT.
           CLOSE CARDFILE DISKFILE.
           STOP RUN.
```

SUMMARY:

You have learned to update a sequential disk file by opening the file as I/O and using it for both input and output operations.

In the next lesson, you will OPEN the same sequential disk file as INPUT, and use the data in it to create a new file with a different type of organization.

END OF LESSON 33

LESSON 34

LESSON 34 - RANDOM FILES ACCESSED SEQUENTIALLY

INTRODUCTION

In this lesson you will write a program to create a file that will have random organization. A random file must be stored on a mass-storage, or direct-access device.

In this lesson you will write a program to create a random file from the sequential file that you created and updated in the preceding lesson. In subsequent lessons you will learn to access this file both sequentially and randomly and to add records to it.

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

ASSIGN clause variation
ACTUAL KEY clause
INVALID KEY option of WRITE statement

This lesson will require approximately three quarters of an hour.
Optional problem: approximately one half hour.

Introduction to Random-Files

In a sequential disk file records are accessed in the same order they are written. Starting with the first record and continuing accessing one at a time stepping only one record's length to access the next. You have seen such applications in the previous lesson. In randomly organized files any record can be accessed at any time. For example, record number 8 can be accessed first, then number 1, then number 5 and so on. Records are accessed through a KEY, as we will see in subsequent lessons. The record KEY is the physical location number of the record. For example, if you wish to access record number 10 the key must contain the number 10. If you wish to write a record in physical location 15 the KEY must contain the number 15.

The disk drive used in 1130-COBOL is the 2310 with the 2315 single disk pack.

(The following lessons have a review of sequential files and continue presentation on random files.)

1. Every record in a random file must have a control field, called a key, for filing purposes. In order that any record may be referred to, the key would have to be:

   a. the same in every record but unique in every file.

   b. unique in every record.

* * *

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..

          ACTUAL KEY IS data-name
```

The elementary variable that contains the key of the record must be specified in the ACTUAL KEY clause. The FILE-CONTROL entry for every file processed randomly must include this clause along with the SELECT and ASSIGN clauses. Using the format of the ACTUAL KEY clause given above, write a FILE-CONTROL entry to link a file named PAYROLL-REGISTER to the 2315 device and to specify the control field SOCIAL-SECURITY-NUMBER.

The ACTUAL KEY is the key that is directly used to locate a logical record on a mass-storage device. The KEY is the logical number of the record. If you wish to retrieve record number 100 the KEY must contain the integer 100. The KEY must be a binary integer five (5) decimal digits long, with usage specified as computational.

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..

        77 KEY-NO PICTURE S99999 USAGE COMP.
```

* * *

```
0   0    1    1    2    2    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 PAYROLL-REGISTER
            ASSIGN TO DF-20-900-X
            ACTUAL KEY IS
                KEY-NUMBER.
```

(A period must be placed only at the end of the entire entry. The clauses may be separated by commas.)

---

3. The ACTUAL KEY clause specifies an elementary variable. The ACTUAL KEY clause could specify:

   a. The location of a record within a file of magazine subscribers.

   b. An elementary variable containing the letters 'KEY'.

*　　　*　　　*

a

---

4. The ACTUAL KEY clause must be specified for:

   a. any file processed randomly.

   b. every file on a mass-storage device.

*　　　*　　　*

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..

       01  WAGE-RECORD.
           02  HOURS.
               03 REGULAR PIC X(3).
               03 OVERTIME PIC X(3).
           02  WAGES.
               03 REGULAR PIC X(3).
               03 OVERTIME PIC X(3).
           02  FILLER PIC X(61).
```

The record description entry above is associated with a file to be processed randomly named WAGE-FILE. A field, EMPLOYEE-NUMBER defines the location of each record in the file. Write a FILE-CONTROL entry to link this file to the 2315 device and the external name WAGESOUT.

*　　　*　　　*

```
0   0    1    1    2    2    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 WAGE-FILE
           ASSIGN TO DF-100-500-X.
           ACTUAL KEY IS EMPLOYEE-NUMBER.
```

---

6. Match the correct facts with each portion of a FILE-CONTROL paragraph.

1) SELECT clause

2) ASSIGN clause

3) ACTUAL KEY clause

a. Identifies the file name

b. Specifies an elementary variable that identifies the record

c. Links the file to a device

d. Is required for all files accessed randomly

e. Is required for all files

* * *

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

---

7. Which of the following must be included in a FILE-CONTROL entry for a file accessed randomly.

a. FD clause

b. SELECT clause

c. INVALID option

d. ACTUAL KEY clause

e. AT END option

f. LABEL RECORDS clause

g. ASSIGN clause

* * *

b, d, g
b and g are also needed for a sequential file.

---

8. Check Figure 173 and then decide which of the following FD entries could refer to a file accessed randomly.

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..

        FD  STUDENT-DATA
                LABEL RECORDS ARE OMITTED.
```

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..

        FD  STUDENT-DATA
                LABEL RECORDS ARE STANDARD.
```

*   *   *

b
These entries must also be used for a sequential disk file.

9.

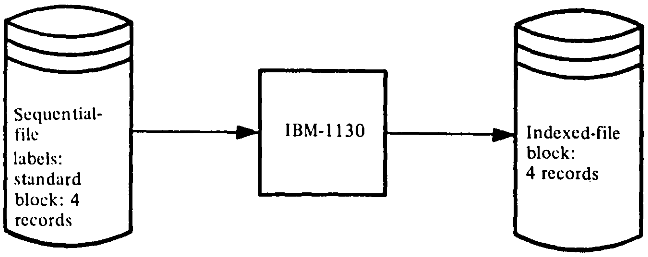


Figure 181

The program PRACTICE uses a sequential disk as input and produces a disk output file organized by a number that identifies the client. The system flow chart is shown above and portions of the first three divisions of PRACTICE are shown in Figure 182. Write the FILE-CONTROL paragraph and additions to complete the FD entries for both files. Refer to Figures 21 and 173 if necessary.

```
0   0   1   1   2   2   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. PRACTICE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
   [
1  [
   [
   [
       DATA DIVISION.
       FILE SECTION.
       FD  SEQUENTIAL-FILE
2  [
       01  TRANSACT-RECORD.
           02  CLIENT-NUMBER PIC S99999 USAGE COMP.
           02  ITEM-NUMBER PIC X(5).
           02  UNIT-PRICE PIC 999V99.
           02  QUANTITY PIC 999.
           02  ITEM-DESCRIPTION PIC X(62).
       FD  DISK-FILE
3  [
       01  DISK-RECORD.
           02  CLIENT-KEY PIC S99999 USAGE COMP.
           02  PART-ID PIC X(5).
           02  AMOUNT-EACH PIC 999V99.
           02  NUMBER-ORDERED PIC 999.
           02  PART-DESCRIPTION PIC X(62).
```

Figure 182

*   *   *

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 SEQUENTIAL-FILE
               ASSIGN TO DF-1-500-X.
           SELECT DISK-FILE
               ASSIGN TO DF-2-600
               ACTUAL KEY IS CLIENT-NUMBER.
```

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..

           LABEL RECORDS ARE STANDARD.
```

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.
```

----

10. In the preceding lesson you learned that the INVALID option must be used in every WRITE statement that refers to:

    a. an output file on a mass-storage device.

    b. a standard disk file opened as OUTPUT.

*   *   *

Both

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 [FROM identifier-1]

               INVALID KEY imperative-statement
```

When the WRITE statement is used to refer to a disk file, the optional COBOL word KEY is also used, as indicated in the format above. Write a statement to place the record TRANSACTIONS in the random file RANDOM-ONE, specifying that, when the INVALID KEY option is activated, control is to be transferred to paragraph KEY-ROUTINE. Remember that INVALID is required here, but that KEY is optional.

*   *   *

```
0   0    1    1    2    2    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 TRANSACTIONS
               INVALID KEY GO TO KEY-ROUTINE.
```

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..

        WRITE PERSONNEL
            INVALID KEY GO TO ERRORS.
```

PERSONNEL is the record name associated with a sequential file. When the statement above is executed, control will be transferred to ERRORS if:

a. a write is attempted beyond the area reserved for the file by control cards.

b. a duplicate key is encountered.

c. a key is found to be out of sequence.

* * *

a

---

13. When a WRITE statement is used to create a sequential file the INVALID KEY option is activated when an attempt is made to write beyond the allocated space. The INVALID KEY clause would be activated when:

a. a record is out of sequence.

b. a duplicate record is encountered.

c. the file area is filled.

* * *

c

---

14. When a READ statement is used to read a sequential file the AT END option is activated when an attempt is made to read beyond the end of the file.

READ DATA-IN AT END GO TO FINISH.

---

15. The INVALID KEY option must be specified for files in the random access mode. The imperative-statement following INVALID KEY is executed when the contents of the ACTUAL KEY field are invalid.

The key is considered invalid:

a. For the disk file that is accessed randomly, when the record number is not within the file limits.

b. For the disk file that is accessed sequentially, when the end of file is read.

* * *

a

---

16. When a WRITE statement with the INVALID KEY option is executed, the key of the record is checked for correct sequence before the record is written. If the INVALID KEY option is activated in an attempt to write a specific record:

    a. the record for which the option was activated will not have been written into the file.

    b. the file will contain the record that was a duplicate or out of sequence.

    c. the file must be recreated to eliminate the invalid record.

*   *   *

a

---

17. The INVALID KEY option is activated in an attempt to write a record beyond the allocated sequential file. To which of the following routines could control be transferred upon activation of the INVALID KEY clause when creating a sequential file with records named SEQ-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..

       INVALID-KEY.
           WRITE SEQ-RECORD.
           GO TO READ-INPUT.
```

    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..

       ERROR-MESSAGE.
           WRITE PRINTOUT
               FROM SEQ-RECORD.
           GO TO READ-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..

       CLOSE-UP.
           CLOSE SEQ-FILE PRINT-FILE.
           STOP RUN.
```

*   *   *

b, c

18.

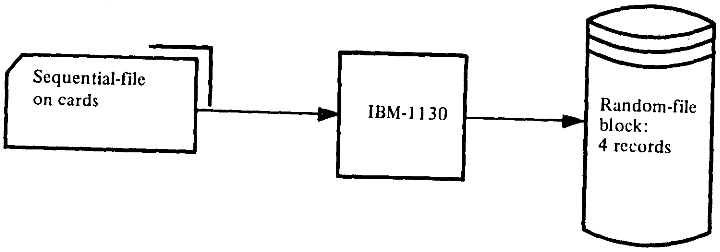


Figure 183

Using the system flow chart above and Figure 182, write the missing entries in the FILE-CONTROL paragraph and the File Section for a program to be used to create a randomly accessed file with a record for each client.

*   *   *

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 SEQUENTIAL-FILE
            ASSIGN TO RD-1442.
        SELECT DISK-FILE
            ASSIGN TO DF-4-500-X
            ACTUAL KEY IS CLIENT-NUMBER.
```

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..

        LABEL RECORDS ARE OMITTED.
```

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.
```

---

19. In the preceding frame you coded portions of the first three divisions of PRACTICE to complete Figure 182. The following flow chart represents the Procedure Division of that program. Program PRACTICE will be used in creating a randomly accessed file. Paragraph ERRORS is to be executed if a record for the output file has an ACTUAL KEY out of range of the disk file. The message "OUT OF RANGE CLIENT-NUMBER" and the client's identification number are to be displayed on the console when an error occurs, but execution is not to be terminated until all input cards have been read. Write the Procedure Division for PRACTICE.

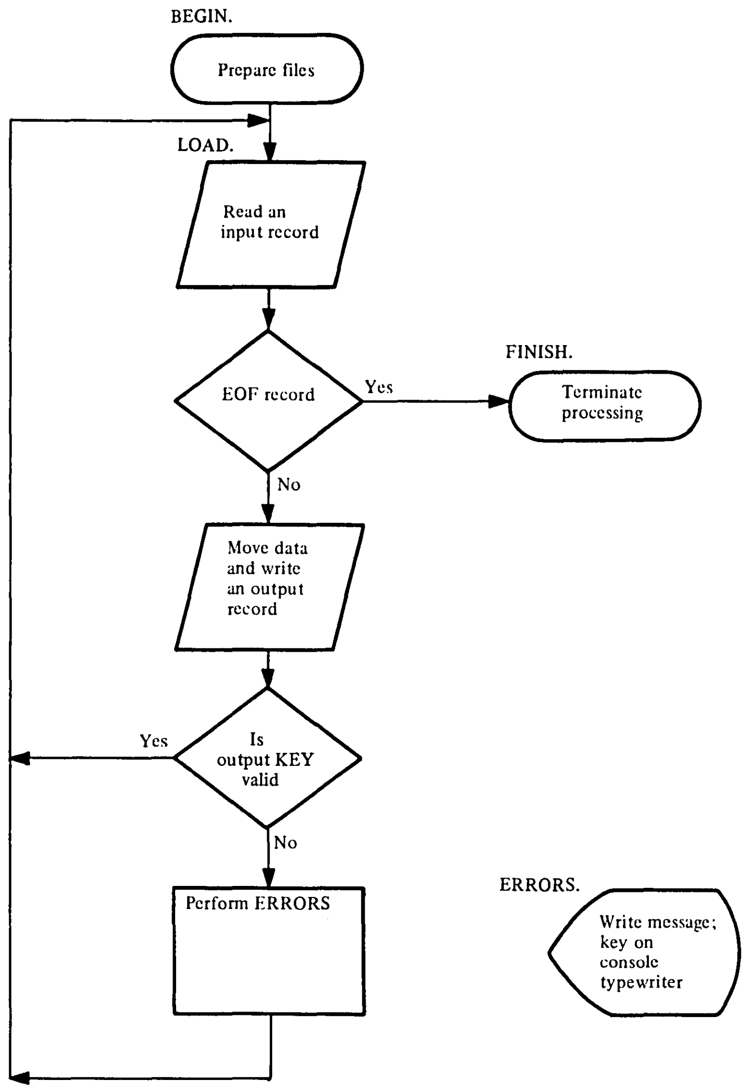

BEGIN.
Prepare files
LOAD.
Read an input record
EOF record
Yes
FINISH.
Terminate processing
No
Move data and write an output record
Yes
Is output KEY valid
No
Perform ERRORS
ERRORS.
Write message; key on console typewriter

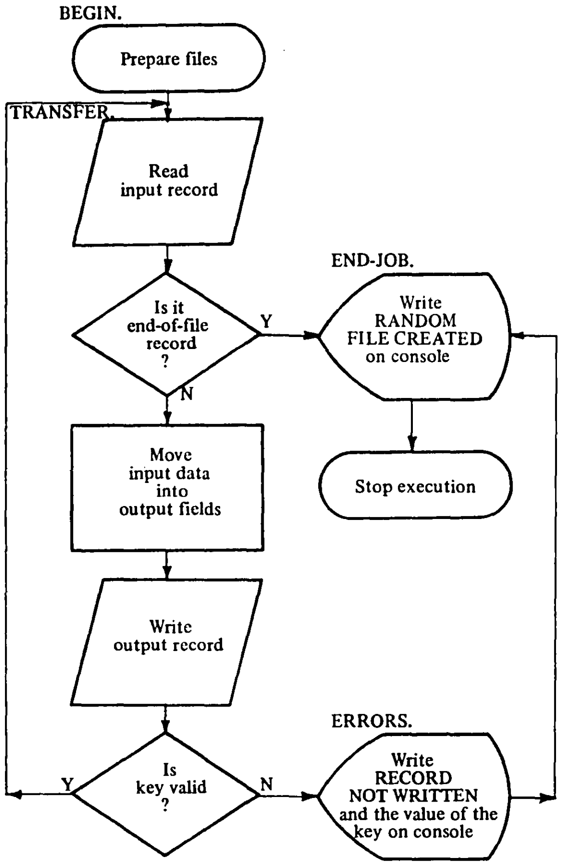
Figure 184

```
                           *        *        *

0   0    1    1    2    2    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 SEQUENTIAL-FILE
               OUTPUT DISK-FILE.
       LOAD.
           READ SEQUENTIAL-FILE
               AT END GO TO FINISH.
           WRITE DISK-RECORD
               FROM TRANSACT-RECORD
               INVALID KEY PERFORM ERRORS.
           GO TO LOAD.
       ERRORS.
           DISPLAY 'OUT OF RANGE CLIENT-NUMBER'
               CLIENT-NUMBER UPON CONSOLE.
       FINISH.
           CLOSE SEQUFNTIAL-FILE
               DISK-FILE.
           STOP RUN.
```

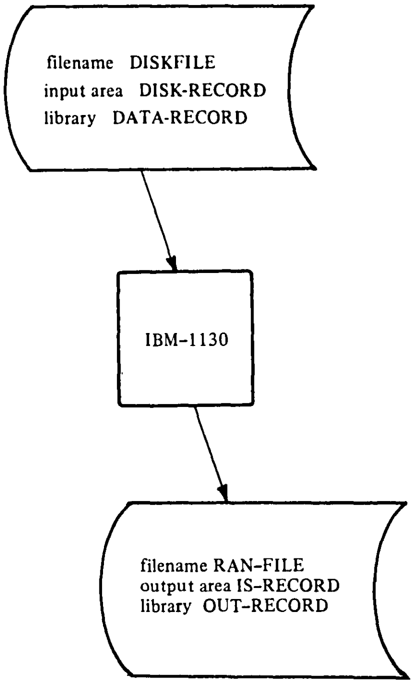
The following problem incorporates the COBOL features you have learned in this lesson. Since you are not asked to do anything new in this problem it is optional for you to code it. If you choose not to code the problem, be sure to read it carefully to make certain you understand it.

---

20. You wrote a program to create a sequential disk file. Now that disk file will be used to create a random file. Figure 185 gives flow charts for the program and the system, along with library entries that can be copied into your program. Now write the entire program called CREATES. Assume that the output file is not in sequence by customer number, thereby making the file random in organization.


BEGIN.
Prepare files
TRANSFER.
Read input record
Is it end-of-file record ?
Y
N
END-JOB.
Write RANDOM FILE CREATED on console
Stop execution
Move input data into output fields
Write output record
ERRORS.
Is key valid ?
Y
N
Write RECORD NOT WRITTEN and the value of the key on console

Program Flow Chart



System Flow Chart

Library name: DATA-RECORD

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  PERSONAL.
               03  NAME PIC X(21).
               03  CUSTOMER-NUMBER PIC X(6).
               03  HOME-ADDRESS.
                   04  STREET PIC X(15).
                   04  CITYSTATE PIC X(15).
           02  PAYRECORD.
               03  YEAR-OPENED PIC 99.
               03  MAXIMUM-CREDIT PIC 9999V99 USAGE COMP.
               03  MAXIMUM-BILL PIC 9999V99 USAGE COMP.
               03  BALANCE-DUE PIC 9999V99 USAGE COMP.
               03  PAYCODE PIC 9.
                   88  BAD VALUE 1.
                   88  POOR VALUE 2.
                   88  SLOW VALUE 3.
                   88  AVERAGE VALUE 4.
                   88  GOOD VALUE 5.
                   88  EXCELLENT VALUE 6.
                   88  NONE VALUE 7.
```

Library name: OUT-RECORD

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  IS-PERSONAL.
               03  IS-NAME PIC X(21).
               03  IS-CUSTOMER-NUMBER PIC X(6).
               03  IS-ADDRESS.
                   04  IS-STREET PIC X(15).
                   04  IS-CITYSTATE PIC X(15).
           02  IS-PAYRECORD.
               03  IS-YEAR-OPENED PIC 99.
               03  IS-MAXIMUM-CREDIT PIC 9999V99 USAGE COMP.
               03  IS-MAXIMUM-BILL PIC 9999V99 USAGE COMP.
               03  IS-BALANCE-DUE PIC 9999V99 USAGE COMP.
               03  IS-PAYCODE PIC 9.
                   88  BAD VALUE 1.
                   88  POOR VALUE 2.
                   88  SLOW VALUE 3.
                   88  AVERAGE VALUE 4.
                   88  GOOD VALUE 5.
                   88  EXCELLENT VALUE 6.
                   88  NONE VALUE 7.
```

```
WORKING-STORAGE.
77  RECORD-ID PIC S9(5) USAGE COMP.
```

Figure 185

*   *   *

```
0   0    1    1    2    2    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. CREATEIS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT DISKFILE
               ASSIGN TO DF-15-500-X.
           SELECT RAN-FILE
               ASSIGN TO DF-26-800-X                      (2)
               ACTUAL KEY IS RECORD-ID.                   (2)
       DATA DIVISION.
       FILE SECTION.
       FD  DISKFILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 4 RECORDS.
       01  DISK-RECORD COPY DATA-RECORD.
       FD  RAN-FILE
           LABEL RECORDS ARE STANDARD.
       01  IS-RECORD COPY OUT-RECORD.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT DISKFILE
               OUTPUT RAN-FILE.
           MOVE 1 TO RECORD-ID.
       TRANSFER.
           READ DISKFILE
               AT END GO TO END-JOB.
           MOVE NAME TO IS-NAME.
           MOVE CUSTOMER-NUMBER TO IS-CUSTOMER-NUMBER.
           MOVE STREET TO IS-STREET
           MOVE CITYSTATE TO IS-CITYSTATE.
           MOVE PAYRECORD TO IS-PAYRECORD.
           WRITE IS-RECORD
               INVALID KEY GO TO ERRORS.                  (11)
           ADD 1 TO RECORD-ID.
           GO TO TRANSFER.
       ERRORS.
           DISPLAY 'RECORD NOT WRITTEN'
               RECORD-ID UPON CONSOLE.
       END-JOB.
           DISPLAY 'RANDOM-FILE CREATED'
               UPON CONSOLE.
           CLOSE DISKFILE RAN-FILE.
           STOP RUN.
```

Multiple disks are basically used in two situations:

a. Multiple units on-line simultaneously for random access.

b. Multiple cartridges used for the same drive, in the case when a sequential file overflows one disk.

---

21. In the preceding frames you have learned to use only one disk for a file. In this frame you will learn to use more than one disk simultaneously. When a file extends into more than one disk it is possible to read records randomly from two or more disks. For example, the first record can be retrieved from the second disk, the second record can be retrieved from the first disk, the third record can be retrieved from the third disk up to a maximum of 5 disks. The Programmer must specify the system-names of the disks he intends to use in the ASSIGN clause.

For example:

```
SELECT IN-AREA ASSIGN TO DF-10-100, DF-20-200, DF-30-300.
```

The READ or WRITE statements are the same.

---

22. If the installation has only one drive available and a file extends into more than one disk, it is possible to read or write into another disk, when the one in use has come to the end, by specifying the FOR MULTIPLE UNIT clause into the ASSIGN statement.

```
SELECT IN-AREA ASSIGN TO DF-1-1000-X FOR MULTIPLE UNIT.
```

The above clause applies only to sequential access.

When the disk reaches the end the computer will type out a message notifying the operator to change disks. The operator will change disks and type the name of the file. The computer will check this name against the disk-directory and if it is valid will continue the operation.

---

23. The SEEK statement is meant to initiate the accessing of a disk-storage data record for subsequent reading or writing. It is used to optimize programming efficiency.

A SEEK statement pertains only to disk files in the random access mode and may be executed prior to execution of a READ or WRITE statement. The main purpose of the statement is to position the READ-WRITE head so that the right disk location is available when a READ or WRITE statement is initiated.

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

        SEEK file-name RECORD.
```

---

24. The file-name must be defined by a file description entry in the Data Division. The SEEK statement uses the contents of the data name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution, the determination is made as to the validity of the file. If the key is invalid, when the next READ or WRITE statement for the associated file is executed, control will be given to the imperative statement in the INVALID KEY option.

Two SEEK statements for the same file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of the second SEEK statement.

If the contents of the ACTUAL KEY are altered between the SEEK statement and the subsequent READ or WRITE statement, any validity check associated with the SEEK statement is negated, and the READ or WRITE statement is processed as if no SEEK statement preceded it.

```
FILE SECTION.
FD  IN-FILE
    RECORD CONTAINS 100 CHARACTERS
    LABEL RECORDS ARE STANDARD.
            .
            .
            .
    SEEK IN-FILE RECORD
            .
            .
            .
    READ IN-FILE INVALID KEY GO TO ERROR-KEY.
            .
            .
            .
```

Note: It is important in planning a routine for loading disk storage that the cylinder concept be taken into consideration. Related data should be grouped in the same cylinder, when possible to eliminate unnecessary seek operations. Therefore, when disk addresses are assigned to a group of related data, the disk location made available should be limited to the number required, plus an expansion factor. The most frequently used data should be stored in the low-numbered cylinders to minimize seek time.

SUMMARY:

In this lesson you have learned to create a random file. In subsequent lessons you will learn various ways of accessing and processing records from this file.

END OF LESSON 34

LESSON 35

LESSON 35 - SEQUENTIAL AND RANDOM ACCESSING PROGRAMS

INTRODUCTION

This lesson will review much of what you have learned about files accessed sequentially and files accessed randomly.

The first program you write will access each record from the file sequentially until the file is closed.

The second program will access records from the file starting at a record that is not the first record in the file and continuing until the file is closed. Since it is not possible to begin accessing records at some record other than the first record in a standard sequential file, you will learn to specify the desired beginning key and to position the file at that desired key prior to accessing records.

This lesson will require approximately one hour.

1. The SELECT, ASSIGN, and ACTUAL KEY clauses must all be used in a FILE-CONTROL paragraph that refers to:

   a. a random file that will be created.

   b. a sequential disk file that will be used as input.

   c. a random file that will be used as input.

* * *

a, c

---

2. The Identification Division of a program that will access all records sequentially from a random file must include the:

   a. PROGRAM-ID paragraph.

   b. Configuration Section.

   c. name of the program.

* * *

a, c

---

3. In order to access all records sequentially from a file opened as INPUT, you would have to use:

   a. the INVALID KEY option in the WRITE statement referring to the file.

   b. the AT END option in the READ statement referring to the file.

   c. the INVALID KEY option in the READ statement referring to the file.

* * *

b

---

4. Match the entries required in a program to use a disk file for sequential input of all records with the divisions in which the clauses would be written.

1) Identification Division

2) Environment Division

3) Data Division

4) Procedure Division

a. LABEL clause

b. PROGRAM-ID paragraph

c. INVALID KEY option

d. BLOCK clause

e. ASSIGN clause

f. OPEN statement

g. SELECT clause

h. AT END option

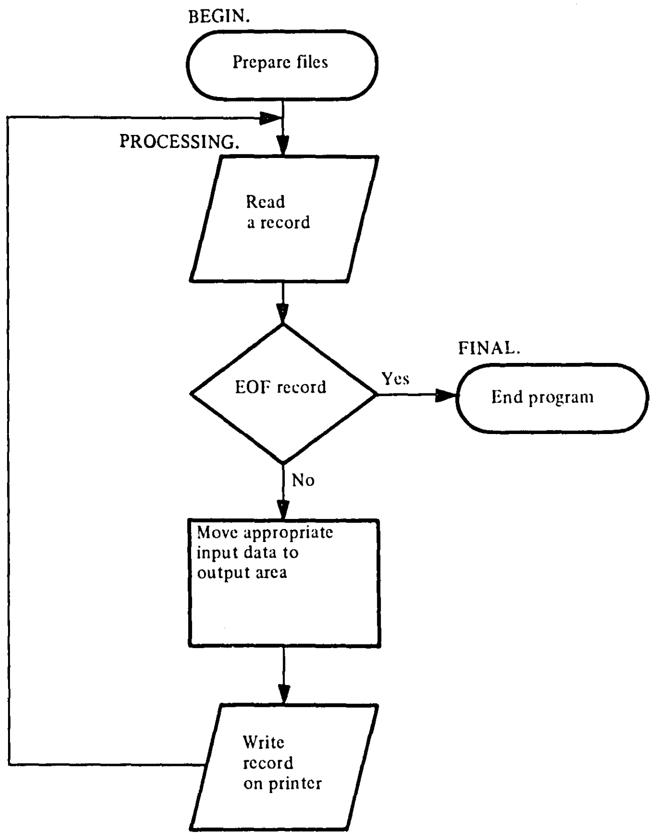
* * *

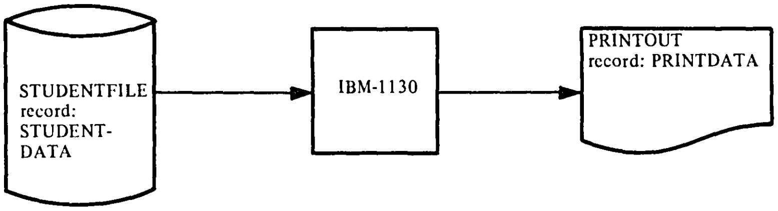
1) b
2) e, g
3) a
4) f, h

(The INVALID KEY option is not written in a program to sequentially access all records from an input file. The BLOCK clause is never required.)

---

5. Figure 186 shows the flow charts and part of a program called STUDYALL. Write the FILE-CONTROL paragraph, and then a Procedure Division to access sequentially all records in the file and list them on the printer in the format indicated in the description of the output record PRINTDATA.


BEGIN.
Prepare files
PROCESSING.
Read
a record
EOF record
Yes
FINAL.
End program
No
Move appropriate
input data to
output area
Write
record
on printer


STUDENTFILE
record:
STUDENT-
DATA
IBM-1130
PRINTOUT
record: PRINTDATA

```
0   0    1    1    2    2    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. STUDYALL.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.

       DATA DIVISION.
       FILE SECTION.
       FD  STUDENT-FILE
           LABEL RECORDS ARE STANDARD.
       01  STUDENT-DATA.
           02  PERSONAL.
               03  YEAR-IN PIC XX.
               03  STUDENT-NUMBER PIC X(5).
               03  NAME PIC X(25).
               03  ADDRESS-I PIC X(35).
           02  SCHOLASTIC.
               03  GRAD PIC XX.
               03  DEGREE PIC XX.
               03  GPA PIC X(4).
               03  MAJOR PIC XXX.
               03  MINOR PIC XXX.
       FD  PRINTOUT
           LABEL RECORDS ARE OMITTED.
       01  PRINTDATA.
           02  PERSONAL.
               03  BLANK1 PIC X(4) VALUE SPACES.
               03  S-NUMBER PIC X(5).
               03  BLANK2 PIC XX VALUE SPACES.
               03  NAME PIC X(25).
               03  BLANK3 PIC X(5) VALUE SPACES.
               03  ADDRESS-I PIC X(35).
               03  BLANK4 PIC X(5) VALUE SPACES.
           02  SCHOLASTIC.
               03  GRAD PIC XX.
               03  BLANK5 PIC X(5) VALUE SPACES.
               03  DEGREE PIC XX.
               03  BLANK6 PIC X(5) VALUE SPACES.
               03  MAJOR PIC XXX.
               03  BLANK7 PIC X(6) VALUE SPACES.
               03  MINOR PIC XXX.
               03  BLANK8 PIC X(14) VALUE SPACES.
```

Figure 186

*   *   *

```
0   0    1    1    2    2    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 STUDENTFILE
               ASSIGN TO DF-6-900-X.
           SELECT PRINTOUT
               ASSIGN TO PR-1132-C.

       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENTFILE
               OUTPUT PRINTOUT.
       PROCESSING.
           READ STUDENTFILE
               AT END GO TO FINAL.
           MOVE STUDENT-NUMBER TO S-NUMBER.
           MOVE PERSONAL OF STUDENT-DATA
               TO PERSONAL OF PRINTDATA
           MOVE SCHOLASTIC OF STUDENT-DATA
               TO SCHOLASTIC OF PRINTDATA.
           WRITE PRINTDATA.
           GO TO PROCESSING.
       FINAL.
           CLOSE STUDENTFILE PRINTOUT.
           STOP RUN.
```

---

6. Whenever the access of records from a disk file will begin at some record other than the first record in the file, the KEY must contain the number of the starting record and then every time a new record is required the KEY must be stepped up by one.

---

7. In order to access the records sequentially from a random file beginning at some record other than the first record in the file, the Programmer must specify the:

   a. ACTUAL KEY clause.

   b. Put into KEY the number of the record to be accessed first and step up the KEY by one every time a new record is required.

*   *   *

Both.

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..

        SELECT STUDENTFILE
            ASSIGN TO DF-6-100-X
            ACTUAL KEY IS NUMBER.
```

The FILE-CONTROL entry above specifies that the elementary variable that contains the key within the record associated with STUDENTFILE is NUMBER.

Write a FILE-CONTROL entry that will name a random file PAY-REGISTER. Specify that the elementary variable that contains the key associated with PAY-REGISTER is MAN-NUMBER.

*   *   *

```
0   0    1    1    2    2    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 PAY-REGISTER
            ASSIGN TO DF-9-300-X
            ACTUAL KEY IS MAN-NUMBER.
```

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..

        ACTUAL KEY IS MAN-NUMBER
```

According to the clause above, the variable MAN-NUMBER:

a. contains the key of the record.

b. is an elementary variable associated with the random file.

*   *   *

Both of these

---

10. Write the clauses necessary to define the variable

PART-NUMBER in Working-Storage and specify that it contains the key of the 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..

        ACTUAL KEY IS PART-NUMBER
            .
            .
            .
    WORKING-STORAGE SECTION.
    77  PART-NUMBER PIC S99999 COMP.
```

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..

       DATA DIVISION.
       FILE SECTION.
       FD  RANDOM-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 5 RECORDS.
       01  IF-RECORD.
           02  FILLER PIC X.
           02  FIELD-1 PIC X(20).
           02  FIELD-2 PIC X(10).
           02  FILLER PIC X(62).
       WORKING-STORAGE SECTION.
       77  RECORD-ID PIC S9(5).
```

Write the clauses necessary to specify RECORD-ID as the actual key for the file 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..

           ACTUAL KEY IS RECORD-ID.
```

---

12. In order to access sequentially the records in a random file beginning at some record other than the first record in the file, the value of the actual key variable must be set equal to the value of the record number of the record at which sequential access is to begin. The actual key could be set by execution of:

    a. a MOVE statement.

    b. an ACCEPT statement.

    c. a DISPLAY statement.

    d. a VALUE clause.

*   *   *

a, b, d

---

13. The ACTUAL KEY clause specifies a variable that will contain a value equal to the record key at which accessing is to begin. The actual key variable must be given a value before accessing of records begins.

    In order that the actual key variable, YEAR, contain the value 1961, you could execute 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..

           MOVE 1961 TO YEAR.
```

    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..

           ACCEPT YEAR FROM CONSOLE.
```

after instructing the operator to key in 1961.

*   *   *

Either

---

14. In order that either statement in frame 13 will actually set the value of the actual key, you must first:

    a. specify ACTUAL KEY IS YEAR.

    b. describe YEAR within the record description of the file to be accessed.

    c. describe YEAR in the Working-Storage Section.

*   *   *

a, c

---

15. Match the following:

1) ACTUAL KEY clause

a. Required whenever sequential access of records in a file is to begin at a record other than the first record in the file

b. Required in every FILE-CONTROL entry for a random file

c. Specifies a variable described in the Working-Storage Section of a program that accesses a random file

* * *

All of these.

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..

        ACTUAL KEY IS PATIENT-NUMBER

    77  PATIENT-NUMBER PIC S99999 USAGE COMP.
```

An Environment Division entry and a Data Division entry for a program to access records from a random file are given above. Write the statements necessary to position the file PATIENT-FILE at the record that has the record key 2025 and to transfer control to END-ROUTINE if this record is not in 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..

        MOVE 2025 TO PATIENT-NUMBER.
        READ PATIENT-FILE
            INVALID KEY GO TO END-ROUTINE.
```

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..

           MOVE 2025 TO PATIENT-NUMBER.
           READ PATIENT-FILE
               INVALID GO TO END-ROUTINE.
```

To ensure that the desired record will be accessed, the key must contain the number of that record before a READ statement is executed. If the above sequence of two statements were executed in the program segment explained in the preceding frame, the record read would be:

a. the record with key equal to 2025.

b. the first record in the file, if the file has just been opened.

c. the next record in the file, if some records have been read from the file.

*   *   *

a

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..

           ACTUAL KEY IS LOCATION
           .
           .
           .
       77  LOCATION PIC S9(5) USAGE COMP.
           .
           .
           .
           READ POSITION-FILE
               INVALID KEY GO TO TERMINATE.
```

In order to begin the access of records from the random file POSITION-FILE with the record whose record key is 1811, a MOVE statement should be placed in the above sequence:

a. before the ACTUAL KEY statement.

b. between the Level 77 Item and READ statements.

c. after the READ statement.

*   *   *

b

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..

                MOVE NUMERO TO NOM-KEY.
                READ IS-FILE
                    INVALID KEY GO TO FINIS.
```

When records in a random file are being accessed sequentially beginning at some record other than the first record in the file, the programmer must specify the action to be taken if no record with a record key of the same value as the actual key variable is found. This action is specified in the INVALID KEY option of the READ statement. The INVALID KEY option of the READ statement is activated when there is no record in the file corresponding to the current value of the actual key.

The statements above are used in a program to access records sequentially from a random file. All of the record keys between 50 and 55 are given below.

00050   00051   00052   00053   00054   00055

Match the results given below with the value of NUMERO which produces that result.

1) 00053

2) 00090

3) 00095

a. Control will be transferred to FINIS

b. The record corresponding to NUMERO will be read.

* * *

1) b
2) a
3) a

---

20. Match the reasons for activation with the appropriate option.

1) INVALID option of WRITE statement for standard sequential disk files

2) INVALID KEY option of WRITE statement for random files

a. An attempt is made to write beyond the limits of the file.

b. The file is filled.

* * *

1) a, b
2) a (the key is negative or zero or greater than the total number of records specified for the file in the ASSIGN clause in the Environment Division.)

---

21. In order to access records sequentially from a random file beginning at some record other than the first record in the file, the program must include several entries in a specific order. Arrange the following in the appropriate order.

   a. The definition of the key in the Working-Storage Section of the Data Division

   b. ACTUAL KEY clause in the Environment Division

   c. READ statement with the INVALID KEY option in the Procedure Division

   d. Statement to set the actual key variable to the value of the record at which sequential access of records is to begin

* * *

b, a, d, c

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..

       IDENTIFICATION DIVISION.
       PROGRAM-ID. STUDYSOME.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENTFILE
               ASSIGN TO DF-1-100-X
               ACTUAL KEY IS REC-LOCATION.


           SELECT PRINTOUT
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE SECTION.
       FD  STUDENTFILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 10 RECORDS.
       01  STUDENT-DATA COPY STUDENT-RECORD.
       FD  PRINTOUT
           LABEL RECORDS ARE OMITTED.
       01  PRINTDATA COPY PARTIAL-LIST.
       WORKING-STORAGE SECTION.
       77  REC-LOCATION PIC S9(5) USAGE COMP.
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT STUDENTFILE
               OUTPUT PRINTOUT.
```

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..

       PROCESSING.


           MOVE STUDENT-NUMBER TO S-NUMBER.
           MOVE NAME OF STUDENT-DATA
               TO NAME OF PRINTDATA
           MOVE ADDRESS OF STUDENT-DATA
               TO ADDRESS PRINTDATA
           MOVE GRAD OF STUDENT-DATA
               TO GRAD OF PRINTDATA
           MOVE DEGREE OF STUDENT-DATA
               TO DEGREE OF PRINTDATA
           MOVE MAJOR OF STUDENT-DATA
               TO MAJOR OF PRINTDATA
           MOVE MINOR OF STUDENT-DATA
               TO MINOR OF PRINTDATA
           WRITE PRINTDATA.
           ADD 1 TO REC-LOCATION.
           GO TO PROCESSING.
       FINAL.
           CLOSE STUDENTFILE PRINTOUT.
           STOP RUN.
```

In a preceding frame in this lesson, you wrote a program to access sequentially and list all records in a file. Now code segment 1 to complete the program above to begin accessing records at key number 1250. Transfer control to the termination routine if the desired key is not in the file or if the end of the file is reached.

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..

           MOVE 1250 TO REC-LOCATION.
       PROCESSING.
           READ STUDENTFILE
               INVALID KEY GO TO FINAL.
```

---

23. As a programmer, you may be asked to modify a program to list all records of parts that have identification numbers larger than 8000. All numbers in PART-FILE end in 5 or 0. Write a paragraph named FIND-KEY to position the file at the first record to be read after 8000 is moved to GUESS-KEY.

*   *   *

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

       FIND-KEY.
           READ PART-FILE
               INVALID KEY
               ADD 5 TO GUESS-KEY
               GO TO FIND-KEY.
```

---

24. A large department store wishes to send a special brochure to a selected sample of customers. The manager has decided to make a preliminary listing of some customers. The first fifty customers in the file will be checked and then fifty customers will be checked starting at a key that the operator will key in at the appropriate time. Of those 100 customers, only addresses of those who have had charge accounts for longer than three years will be listed. Figure 187 shows a system flow chart and a part of the first three divisions of program SEQ-ACCESS. Write the missing section of the Environment Division.

```
0   0    1    1    2    2    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. SEQ-ACCESS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.

       DATA DIVISION.
       FILE SECTION.
       FD  IND-SEQ
           LABEL RECORDS ARE STANDARD.
       01  DISK-RECORD.
           02  IS-PERSONAL.
               03  IS-NAME PIC X(21).
               03  IS-ADDRESS.
                   04  IS-STREET PIC X(15).
                   04  IS-CITYSTATE PIC X(15).
           02  IS-PAYRECORD.
               03  IS-YEAR-OPENED PIC 99.
               03  IS-MAXIMUM-CREDIT PIC 9999V99
                   USAGE COMP.
               03  IS-MAXIMUM-BILL PIC 9999V99
                   USAGE COMP.
               03  IS-BALANCE-DUE PIC 9999V99
                   USAGE COMP.
               03  IS-PAYCODE PIC 9.
                   88  BAD VALUE 1.
                   88  POOR VALUE 2.
                   88  SLOW VALUE 3.
                   88  AVERAGE VALUE 4.
                   88  GOOD VALUE 5.
                   88  EXCELLENT VALUE 6.
                   88  NONE VALUE 7.
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  OUTGO PIC X(121).
       WORKING-STORAGE SECTION.
       77  TIME PIC 99.
       77  ACCEPTED PIC 999 USAGE COMP
               VALUE ZEROS.
       77  REJECTED PIC 999 USAGE COMP
               VALUE ZEROS.
       77  RECORD-ID PIC S9(5) USAGE COMP.
       01  PRINT-LINE.
           02  FILLER PIC X(9) VALUE SPACES.
           02  FILLER PIC XX VALUE '19'.
           02  YEARS PIC 99.
           02  FILLER PIC X(10) VALUE SPACES.
           02  IDENT PIC X(6).
           02  FILLER PIC X(10) VALUE SPACES.
           02  CUSTOMER PIC X(20).
           02  FILLER PIC X(15) VALUE SPACES.
           02  O-STREET PIC X(15).
           02  FILLER PIC X(10) VALUE SPACES.
           02  O-CITYSTATE PIC X(15).
           02  FILLER PIC X(7) VALUE SPACES.
```

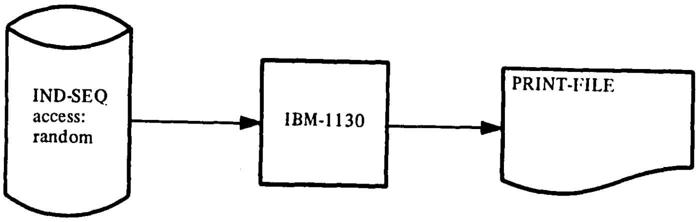


Figure 187

```
                          *         *         *

0   0    1    1    2    2    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 RAN-FILE
               ASSIGN TO DF-15-900-X
               ACTUAL KEY IS RECORD-ID.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
```

SUMMARY:

In this lesson you have learned to access records sequentially from a file, starting at any record in the file. In the next lesson you will learn to access records randomly and to add records to your random file.

END OF LESSON 35

LESSON 36

LESSON 36 - RANDOM FILES ACCESSED RANDOMLY

INTRODUCTION

In this lesson you will learn to access records from a file randomly. You will also learn to add records to a random file without recreating the file.

In the usual business situation involving random files, the master file has indexed organization. This organization is not to be confused with Index Sequential organization found in systems other than IBM 1130. The index here refers only to the record keys, since the file is accessed randomly, not sequentially. When this indexed file is processed, a value from an input transaction file, whose records may be in any order, is used to locate the desired master record. Execution of a READ statement specifying the random file will then access the appropriate record. If the transaction file contains records to be added to the master file, each record can be inserted into the appropriate place in the random file when a WRITE statement is executed after the actual key variable is set as above.

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

ACCESS clause
SEQUENTIAL option of ACCESS clause
RANDOM option of ACCESS clause
INVALID KEY option of READ statement

This lesson will require approximately three quarters of an hour.

FILE-CONTROL clauses File Access	SELECT	ASSIGN	ACTUAL KEY	ACCESS	RESERVE
Sequential Access	R	R	N	O	P
Random Access	R	R	R	R	N

P = Optional for print and card files; not permitted otherwise.
R = Required.
O = Optional
N = Not permitted.

Figure 188

1. Figure 188 shows the required clauses of the Environment Division entries for the types of file access and organization that are studied in this course. Which clauses are required for a FILE-CONTROL paragraph that describes any sequential disk file?

* * *

SELECT and ASSIGN clauses

---

2. Which clauses are required for a FILE-CONTROL paragraph that describes a sequentially accessed file that will be accessed beginning with the first record in the file?

* * *

SELECT, ASSIGN clauses

---

3. Which clause is required for a disk file if access is to begin at some record other than the first record in the file?

* * *

ACTUAL KEY clause

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..

           ACCESS IS SEQUENTIAL
```

Although you have not used the clause shown above in any of the programs you have written, it may be specified for any sequentially accessed file. When the ACCESS clause was omitted, the method of access you used was:

a. RANDOM

b. SEQUENTIAL

* * *

b

---

5. Figure 188 shows that the:

a. ACCESS clause is optional for sequentially accessed files.

b. ACCESS clause is required for randomly accessed files.

c. omission of the ACCESS clause has the same effect as specifying ACCESS IS SEQUENTIAL.

* * *

All of these

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..

           ACCESS IS RANDOM
```

The form of the ACCESS clause shown above is required for random access of records from a file. This form of the ACCESS clause could refer to a file on:

a. a mass-storage device.

b. input cards.

* * *

a

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..

           ACCESS IS RANDOM
```

To which of the following FILE-CONTROL entries could the clause above be applied?

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..

           ASSIGN TO RD-1442.
```

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..

           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..

           ASSIGN TO DF-10-500-X.
```

*   *   *

c

---

8. Refer to Figure 188 if necessary and decide which of the following FILE-CONTROL entries contains appropriate clauses for the type of file specified in the ASSIGN clause.

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..

           SELECT FILE-2
               ASSIGN TO DF-10-500-X
               ACCESS IS SEQUENTIAL.
```

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..

           SELECT FILE-3
               ASSIGN TO PR-1132-C
               ACTUAL KEY IS STOCK-CODE
               ACCESS IS RANDOM.
```

*   *   *

a
(It is not permitted to have an ACTUAL KEY clause or RANDOM ACCESS clause when assigning a non-disk file.)

---

9. According to Figure 188, a FILE-CONTROL entry that contains the clause ACCESS IS RANDOM must also include the:

   a. ACTUAL KEY clause.

   b. ACCESS IS SEQUENTIAL clause.

*   *   *

a

---

10. The ACTUAL KEY clause must be specified in the FILE-CONTROL entry for every:

   a. file on a mass-storage device.

   b. random file.

*   *   *

b

---

11. The ACTUAL KEY clause must be specified in the FILE-CONTROL entry for a file when:

   a. sequential access is to begin at some record other than the first record in the file.

   b. ACCESS IS RANDOM is specified.

*   *   *

Either

---

12. Refer to Figure 188 and write a FILE-CONTROL entry for RANDOM-FILE, a randomly accessed file. The key field of the record associated with the file is ID-FIELD.

*   *   *

```
0   0    1    1    2    2    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 RANDOM-FILE
               ASSIGN TO DF-10-700-X
               ACTUAL KEY IS ID-FIELD
               ACCESS IS RANDOM.
```

You have now learned to code the Environment Division entries required for a file that is to be accessed randomly. In the next sequence you will learn to code the appropriate Procedure Division entries.

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..

                SELECT RANDOM-FILE
                    ASSIGN TO DF-7-800-X
                    ACTUAL KEY IS ID-FIELD
                    ACCESS IS RANDOM.
```

When records are accessed randomly, the actual key field must be set to the value of the record key field of the desired record before a READ statement is executed for the file. Refer to the clauses above and select the correct statement(s).

a. Before a READ statement for RANDOM-FILE is executed, ID-FIELD must be set to the value of the desired record key.

b. ID-FIELD must be set to the value of the desired record key as soon as a READ statement for RANDOM-FILE is executed.

*   *   *

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..

                READ file-name [INTO record-name]

                    INVALID KEY imperative-statement
```

Any READ statement that refers to a randomly accessed file must include the INVALID KEY option. The format above indicates that the:

a. INTO option may be included in a READ statement that includes the INVALID KEY option.

b. AT END option may not be included in a READ statement that includes the INVALID KEY option.

*   *   *

Both
(The end-of-file record is not checked for when a file is randomly accessed.)

---

15. Match the following with the appropriate options of the READ statement.

1) AT END

2) INTO

3) INVALID KEY

a. Allows the execution of the READ statement to have the effect of a READ and a MOVE

b. Is not permitted when the READ statement refers to a randomly accessed file

c. Is required when the READ statement refers to a randomly accessed file

d. Is used to specify action to take place when the end of a sequentially accessed file is reached

e. Is optional when the READ statement refers to a randomly accessed file

* * *

1) b, d
2) a
3) c

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..

           SELECT RANDOM-FILE
               ASSIGN TO DF-10-600-X
               ACTUAL KEY IS ID-FIELD
               ACCESS IS RANDOM.
```

Assume that ID-FIELD has been set to the value of the key of the desired record and write a READ statement without the INTO option to access the record from the file described above. Use PERFORM SPECIAL-CASES as the imperative statement in the option you use.

* * *

```
0   0    1    1    2    2    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 RANDOM-FILE INVALID KEY
               PERFORM SPECIAL-CASES.
```

---

17. Match the following causes for activation with the options.

1) INVALID option in a WRITE statement that refers to a standard sequential file

2) INVALID KEY option in a READ statement

that refers to a randomly accessed file

3) INVALID KEY option in a WRITE statement that refers to a random file that is being created

a. The value of the actual key variable is negative or zero, or greater than the total number of records specified for the file in the ASSIGN clause in the Environment Division.

b. The space reserved for the file is filled.

c. No record is present in the file with a record key equal to the value contained by the ACTUAL KEY.

* * *

1) b
2) c
3) a

---

18. The INVALID KEY option must be included in any READ statement that refers to a:

a. file that has sequential organization.

b. randomly accessed file.

* * *

b

---

19. When a sequential file is being accessed randomly, the actual key variable must be set to the value of the record key of the desired record:

a. only before a READ statement for the first record is executed.

b. before each READ statement is executed.

* * *

b

---

20. ACCESS IS RANDOM may be specified for a sequential file. Select the entries below that must also be specified for a file if ACCESS IS RANDOM is specified.

a. INVALID KEY option of the READ statement

b. ACTUAL KEY clause

* * *

Both

21.

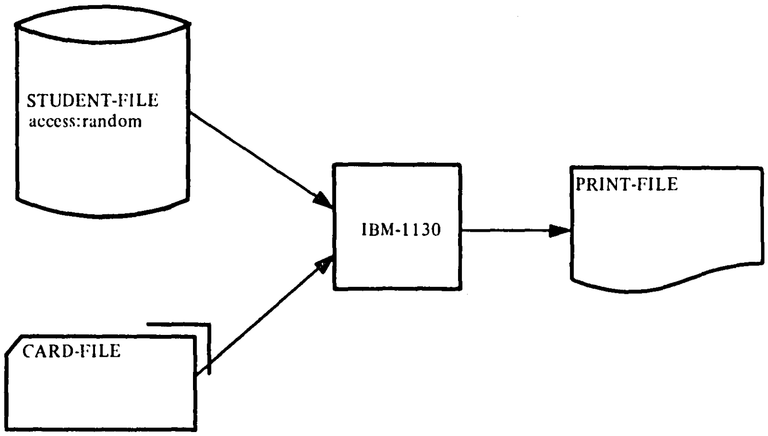


Figure 189

Figure 190 shows portions of a program that randomly accesses a sequentially organized file, STUDENT-FILE. Refer to the system flow chart above and the first three divisions of GRADUATE-CHECK in Figure 190 to complete the FILE-CONTROL paragraph.

```
0   0    1    1    2    2    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. GRADUATE-CHECK.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENT-FILE


           SELECT CARD-FILE
               ASSIGN TO RD-1442.
           SELECT PRINT-FILE
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE-SECTION.
       FD  STUDENT-FILE
           LABEL RECORDS ARE STANDARD.
       01  STUDENT-RECORD.
           02  STUDENT-NUMBER PIC X(5).
           02  STUDENT-NAME PIC X(25).
           02  YEAR-GRADUATED PIC 99.
           02  OTHER-THINGS PIC X(51).
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CARD-RECORD.
           02  CARD-NUMBER PIC X(9).
           02  FILLER PIC X(71).
       FD  PRINT-FILE
           LABEL RECORDS ARE OMITTED.
       01  OUT-RECORD PIC X(121).
       WORKING-STORAGE SECTION.
       77  KEY-NUMBER PIC S99999 USAGE COMP.
       77  YEAR-G PIC 99 USAGE COMP.
       01  PRINTOUT.
           02  CARRIAGE-CONTROL PIC X.
           02  PRINT-DATA.
               03  STUDENT-NUMBER-O PIC X(5).
               03  STUDENT-NAME-O PIC X(25).
               03  YEAR-GRADUATED-O PIC 99.
               03  OTHER-THINGS-O PIC X(51).
               03  COMMENT-O PIC X(37).
               03  FILLER PIC X.


       PROCEDURE DIVISION.
       BEGIN.
       OPEN INPUT STUDENT-FILE
               CARD-FILE
               OUTPUT PRINT-FILE.
       PROCESSING.

       MESSAGE.
       DISPLAY CARD-NUMBER
               'NOT IN FILE' UPON CONSOLE.
           GO TO PROCESSING.
       ENDING.
       CLOSE STUDENT-FILE
               CARD-FILE PRINT-FILE.
           STOP RUN.
```

Figure 190

```
                          *        *        *

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

           ASSIGN TO DF-6-900-X
           ACTUAL KEY IS KEY-NUMBER
           ACCESS IS RANDOM.

-----------------------------------------------------------------------
```

22.

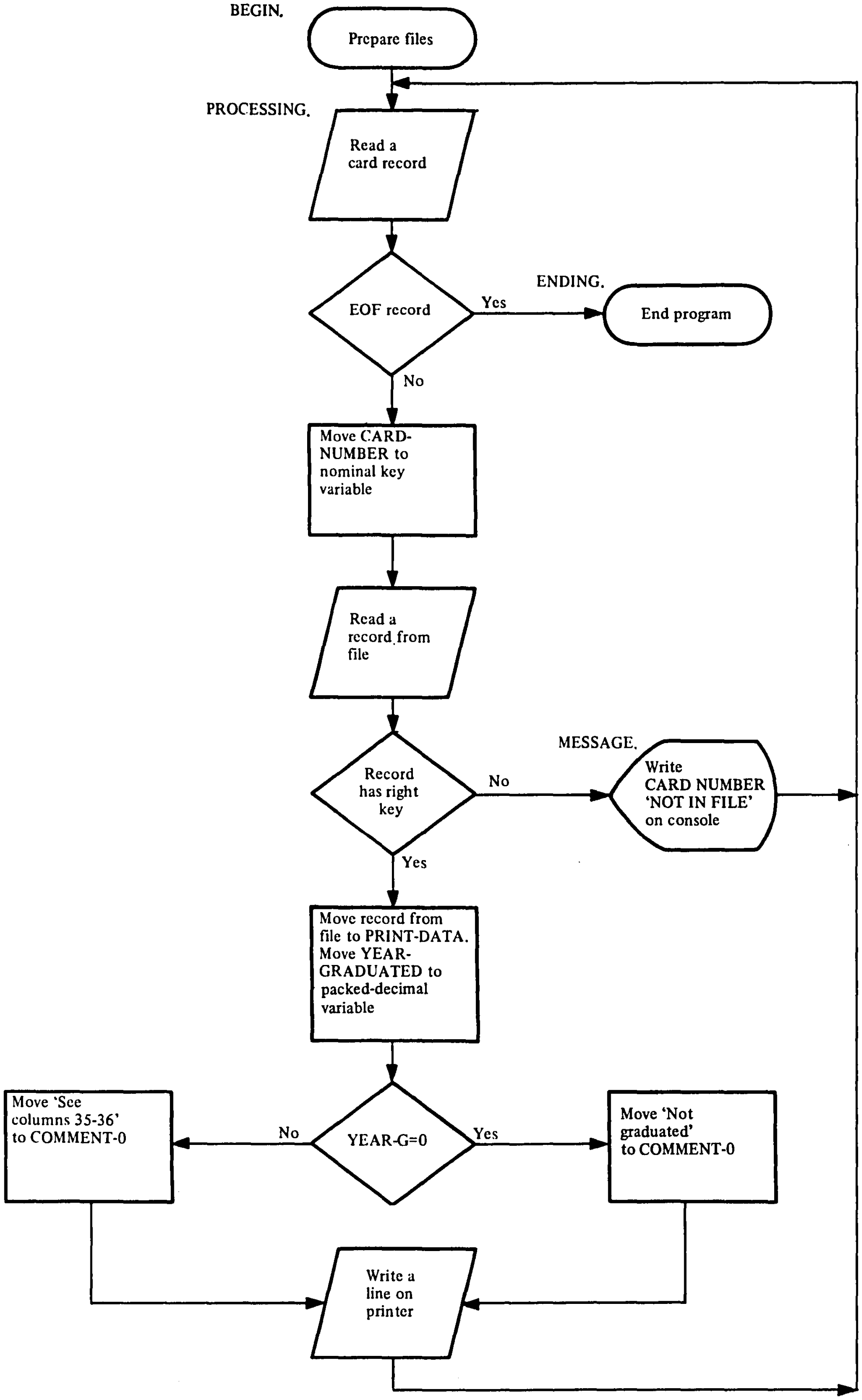

BEGIN.
Prepare files
PROCESSING.
Read a card record
EOF record
Yes
ENDING.
End program
No
Move CARD-NUMBER to nominal key variable
Read a record from file
MESSAGE.
Record has right key
No
Write CARD NUMBER 'NOT IN FILE' on console
Yes
Move record from file to PRINT-DATA. Move YEAR-GRADUATED to packed-decimal variable
Move 'See columns 35-36' to COMMENT-0
No
YEAR-G=0
Yes
Move 'Not graduated' to COMMENT-0
Write a line on printer

Figure 191

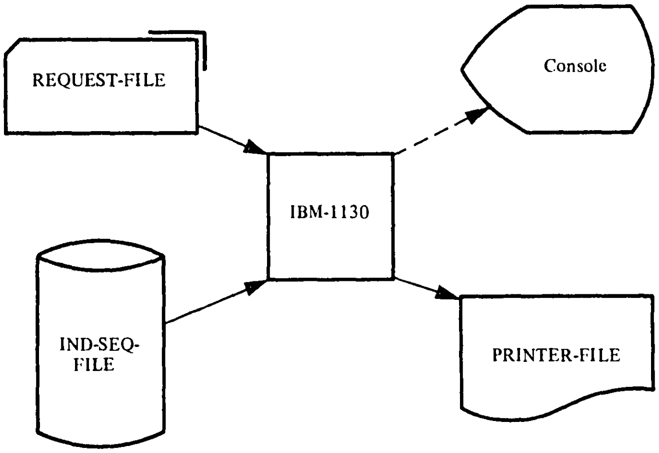
The prior flow chart shows the logic for the Procedure Division of the program in Figure 190. In this program the records from STUDENT-FILE for students who have records in CARD-FILE are to be printed in PRINT-FILE. A comment about the graduation of the student is also printed in PRINT-FILE. Follow the flow chart and write the main routine PROCESSING for the program GRADUATE-CHECK.

* * *

```
0   0    1    1    2    2    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
               AT END GO TO ENDING.
           MOVE CARD-NUMBER TO KEY-NUMBER.
           READ STUDENT-FILE
               INVALID KEY GO TO MESSAGE.
           MOVE STUDENT-RECORD TO PRINT-DATA.
           MOVE YEAR-GRADUATED TO YEAR-G.
           IF YEAR-G NOT EQUAL TO 0
               MOVE 'SEE COLUMNS 35-36'
                   TO COMMENT-0
               ELSE MOVE 'NOT GRADUATED'
                   TO COMMENT-0.
           WRITE OUT-RECORD FROM PRINTOUT.
           GO TO PROCESSING.
```

23. One practical use for a random access file would be to check the credit of customers who have ordered large amounts of merchandise through a mail-order warehouse. Figure 192 shows a system flow chart and the first three divisions of a program that is used for this purpose. The card file contains the customer's unique number (the position of this customer's record on the random access file) and the dollar value of the merchandise he wishes delivered. This file is prepared by the mail clerk as orders for the day are received and is therefore in random order. Certain records from the input files will be placed in PRINTER-FILE.

Follow the program flow chart in Figure 193 and write the Procedure Division for the program CREDIT-CHECK.



```
0   0    1    1    2    2    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. CREDIT-CHECK.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT REQUEST-FILE
               ASSIGN TO RD-1442.
           SELECT RANDOM-FILE
               ASSIGN TO DF-1-600-X
               ACTUAL KEY IS CREDIT-CARD
               ACCESS IS RANDOM.
           SELECT PRINTER-FILE
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE SECTION.
       FD  REQUEST-FILE
           LABEL RECORDS ARE OMITTED.
       01  CUSTOMER-DATA
```

```
    02  CREDIT-NUMBER PIC X(6).
    02  REQUEST PIC 9999V99.
    02  FILLER PIC X(68).
FD  RANDOM
    LABEL RECORDS ARE STANDARD
    BLOCK CONTAINS 5 RECORDS.
01  DISK-RECORD.
    02  IS-PERSONAL.
        03  IS-NAME PIC X(21).
        03  RECORD-ID PIC S99999 USAGE COMP.
        03  IS-ADDRESS.
            04  IS-STREET PIC X(15).
            04  IS-STATE PIC X(15).
    02  IS-PAYRECORD.
        03  IS-YEAR-OPENED PIC 99.
        03  IS-MAXIMUM-CREDIT PIC 9999V99
            USAGE COMP.
        03  IS-MAXIMUM-BILL PIC 9999V99
            USAGE IS COMP.
        03  IS-BALANCE DUE PIC 9999V99
            USAGE IS COMP.
        03  IS-PAYCODE PIC 9.
            88  BAD VALUE 1.
            88  POOR VALUE 2.
            88  SLOW VALUE 3.
            88  AVERAGE VALUE 4.
            88  GOOD VALUE 5.
            88  EXCELLENT VALUE 6.
            88  NONE VALUE 7.
FD  PRINTER-FILE
    LABEL RECORDS ARE OMITTED.
01  PRINTOUT PIC X(121).
WORKING-STORAGE SECTION.
77  CREDIT-CARD PIC S99999 USAGE COMP.
77  ASK PIC 9999V99 USAGE COMP.
77  TOTAL-NEEDED PIC 9999V99 USAGE COMP.
77  DIFFERENCE PIC 9999V99 USAGE COMP.
77  ALLOW PIC X(22) VALUE
        'MARGINAL BUT PERMITTED'.
77  DISALLOW PIC X(24) VALUE
        'CHECK WITH CREDIT OFFICE'.
01  FILE-DATA.
    02  FILLER PIC X VALUE IS SPACES.
    02  NAME PIC X(21).
    02  IDENT PIC X(6).
    02  AMOUNT-ASKED PIC 9,999.99.
    02  FILLER PIC XXX VALUE IS SPACES.
    02  COMMENT PIC X(83).
```

Figure 192

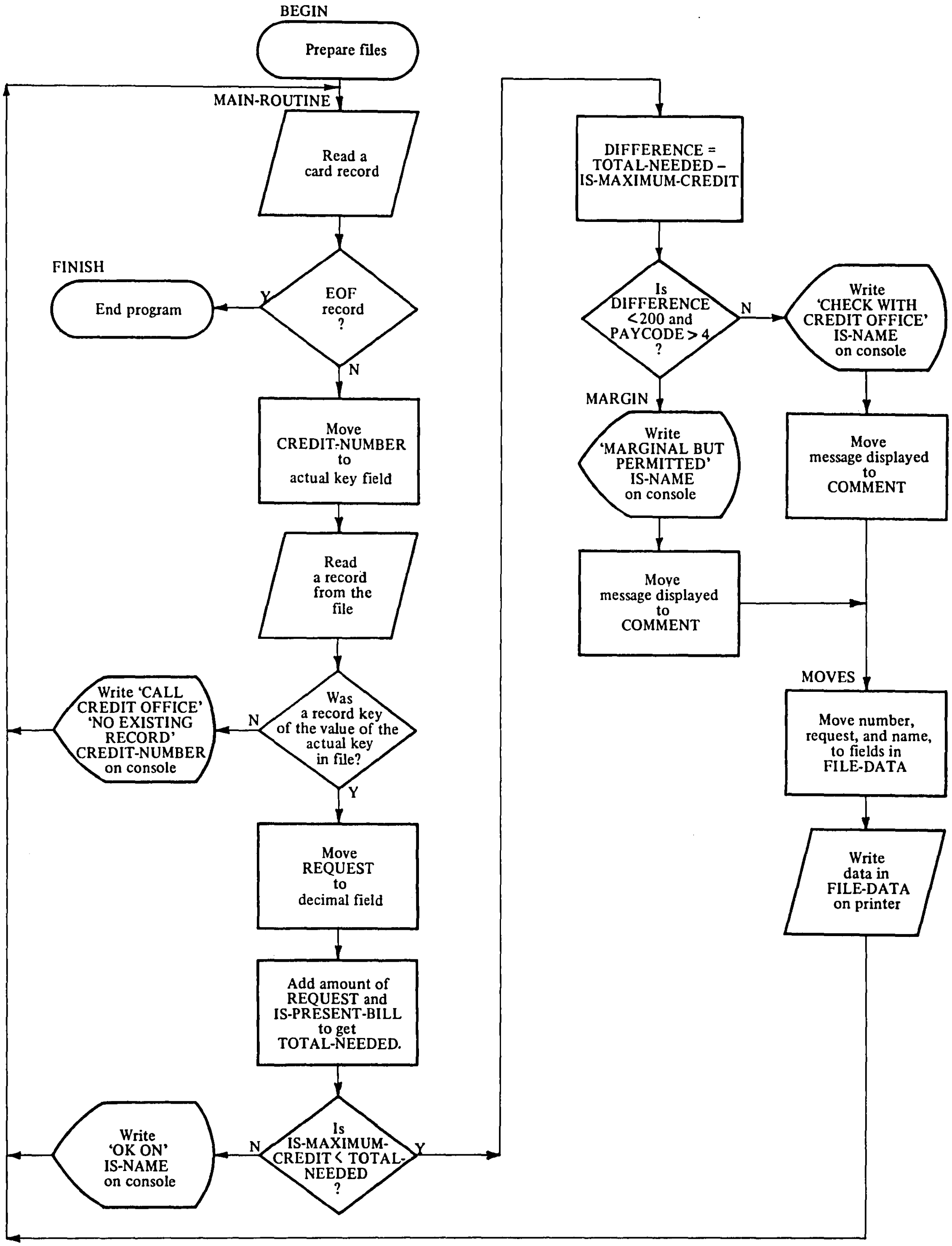

BEGIN
Prepare files
MAIN-ROUTINE
Read a card record
FINISH
End program
Y
EOF record ?
N
Move CREDIT-NUMBER to actual key field
Read a record from the file
Write 'CALL CREDIT OFFICE' 'NO EXISTING RECORD' CREDIT-NUMBER on console
N
Was a record key of the value of the actual key in file?
Y
Move REQUEST to decimal field
Add amount of REQUEST and IS-PRESENT-BILL to get TOTAL-NEEDED.
Write 'OK ON' IS-NAME on console
N
Is IS-MAXIMUM-CREDIT < TOTAL-NEEDED ?
Y
DIFFERENCE = TOTAL-NEEDED – IS-MAXIMUM-CREDIT
Is DIFFERENCE <200 and PAYCODE > 4 ?
N
Write 'CHECK WITH CREDIT OFFICE' IS-NAME on console
MARGIN
Write 'MARGINAL BUT PERMITTED' IS-NAME on console
Move message displayed to COMMENT
Move message displayed to COMMENT
MOVES
Move number, request, and name, to fields in FILE-DATA
Write data in FILE-DATA on printer

Figure 193

```
                          *        *        *

0   0    1    1    2    2    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 REQUEST-FILE
               RANDOM-FILE
               OUTPUT PRINTER-FILE.
       MAIN-ROUTINE.
           READ REQUEST-FILE
               AT END GO TO FINISH.
           MOVE CREDIT-NUMBER TO CREDIT-CARD.
           READ RANDOM-FILE INVALID KEY                (14)
               DISPLAY 'CALL CREDIT OFFICE'
               'NO EXISTING RECORD'
               CREDIT-NUMBER
               GO TO MAIN-ROUTINE.
           MOVE REQUEST TO ASK.
           ADD ASK IS-BALANCE-DUE
               GIVING TOTAL-NEEDED.
           IF IS-MAXIMUM-CREDIT NOT LESS
               THAN TOTAL-NEEDED
               DISPLAY 'OK ON' IS-NAME
                   UPON CONSOLE
               GO TO MAIN-ROUTINE.
           COMPUTE DIFFERENCE = TOTAL-NEEDED
               - IS-MAXIMUM-CREDIT.
           IF DIFFERENCE LESS THAN 200
               AND PAYCODE GREATER THAN 4
               GO TO MARGIN.
           DISPLAY DISALLOW IS-NAME
               UPON CONSOLE.
           MOVE DISALLOW TO COMMENT.
       MOVES.
           MOVE CREDIT-NUMBER TO IDENT.
           MOVE REQUEST TO AMOUNT-ASKED.
           MOVE IS-NAME TO NAME.
           WRITE PRINTOUT FROM FILE-DATA.
           GO TO MAIN-ROUTINE.
       MARGIN.
           DISPLAY IS-NAME ALLOW
               UPON CONSOLE.
           MOVE ALLOW TO COMMENT.
           GO TO MOVES.
       FINISH.
           CLOSE REQUEST-FILE RANDOM-FILE
               PRINTER-FILE.
           STOP RUN.
```

(If your solution is different and you are not sure it is correct, consult your advisor.)

In the next sequence of frames you will learn to add records to your random file.

----

24. A Disk File may be accessed by a READ and a WRITE in the same program if the OPEN statement specifies I-O. The OPEN statement for a file to be updated should specify:

a. INPUT

b. OUTPUT

c. I-O

* * *

c

---

25. If a mass storage file (disk file) is opened I-O in a sequential access mode, a READ for the file must precede any WRITE. The following statements should be in what order in a program (Assume sequential access) ?

a. WRITE FILE1

b. OPEN FILE1

c. READ FILE1

* * *

b,c,a

---

26. When records are to be added to a random file, the file must have .... access and be opened as ....

* * *

random
I-O

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..
        WRITE record-name

            INVALID KEY imperative-sentence
```

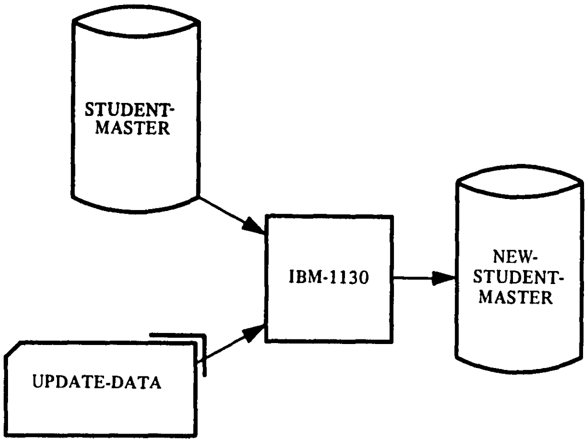
A statement of the form shown above is used to add a record to a random file. In a program which is used only to add records to a random file, the relevant file should be opened as ....

* * *

I-O
(See preceding if your answer was OUTPUT.)

---

28. When a record is being added to a random file, the INVALID KEY option of the WRITE statement is activated if the actual key field associated with the record to be added contains a value outside the limits of the file. The actual key variable must be set equal to the disk location of the record to be added.



```
0   0    1    1    2    2    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-2-700-X.
           SELECT UPDATE-DATA
               ASSIGN TO RD-1442.
       DATA DIVISION.
       FILE SECTION.
       FD  STUDENT-MASTER
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 5 RECORD.
       01  STUDENT-DATA.
           02  PERSONAL.
               03  YEAR-IN PIC XX.
               03  STUDENT-NUM PIC X(4).
               03  FILLER PIC X(35).
           02  SCHOLASTIC PIC X(14).
       FD  NEW-STUDENT-MASTER
           LABEL RECORDS ARE STANDARD.
       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).
       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.
```

Figure 194

29. Figure 194 shows a program that you wrote to add records to a sequential file. In this program there were two files - an input Master and an output Master file. If records are added to a random file, only one file is needed, if it is opened as I-O, since it is used for both input and output. The following changes must be made in the program previously written in order to change the two sequential disk files to one random file:

    a. Delete one SELECT sentence.

    b. Delete one FD entry.

    c. Delete the record description entry associated with the FD.

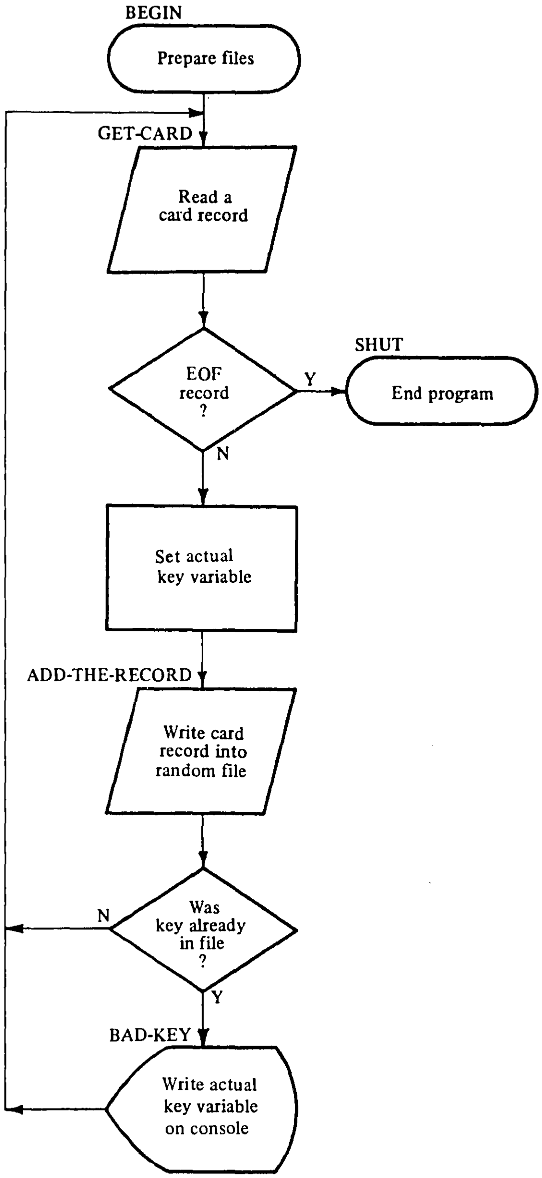
    d. Rewrite the FILE-CONTROL paragraph to specify one disk file - a random file called STUDENT-MASTER and the card file shown in Figure 194.

    e. Add a Working-Storage Section to describe the ACTUAL KEY variable.

    Assume that a variable named KEY-NUMBER is described as a level 77 variable. Write the new SELECT statements.

*   *   *

```
0   0    1    1    2    2    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 STUDENT-MASTER
              ASSIGN TO DF-1-700-X
              ACTUAL KEY IS KEY-NUMBER
              ACCESS IS RANDOM.
          SELECT UPDATE-DATA
              ASSIGN TO RD-1442.
```


BEGIN
Prepare files
GET-CARD
Read a card record
EOF record ?
Y
SHUT
End program
N
Set actual key variable
ADD-THE-RECORD
Write card record into random file
Was key already in file ?
N
Y
BAD-KEY
Write actual key variable on 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. ADDRECORDS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT STUDENT-MASTER
               ASSIGN TO DF-9-700-X
               ACTUAL KEY IS KEY NUMBER
               ACCESS IS RANDOM.
           SELECT UPDATE-DATA
               ASSIGN TO RD-1442.
       FILE SECTION.
       FD  STUDENT-MASTER
           LABEL RECORDS ARE STANDARD.
       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  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).
       WORKING-STORAGE SECTION.
       77  KEY-NUMBER PIC S9(5) COMP.
```

Figure 195

30. Figure 195 shows the version of the first three divisions that you have just completed for the program ADDRECORDS, along with a program flow chart for a Procedure Division to add records to STUDENT-MASTER. Follow the flow chart and write the Procedure Division.

* * *

```
0   0    1    1    2    2    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 UPDATE-DATA
               I-O STUDENT-MASTER.
       GET-CARD.
           READ UPDATE-DATA
               AT END GO TO SHUT.
           MOVE CARD-NUMBER TO KEY-NUMBER.
       ADD-THE-RECORD.
           WRITE STUDENT-DATA FROM TRANSFER
               INVALID KEY PERFORM BAD-KEY.
           GO TO GET-CARD.
       BAD-KEY.
           DISPLAY KEY-NUMBER
               'INVALID KEY' UPON CONSOLE.
       SHUT.
           CLOSE UPDATE-DATA STUDENT-MASTER.
           STOP RUN.
```

The I-O option of the OPEN statement pertains only to mass-storage files. The following example demonstrates the OPEN clause in all aspects.

```
0   0    1    1    2    2    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 DISK-FILE ASSIGN TO DF-1-800-X
               ACCESS IS RANDOM
               ACTUAL KEY IS KEY-ID.
           SELECT CARD-FILE ASSIGN TO RD-1442.
           SELECT PRINT-FILE ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREAS.
       I-O-CONTROL.
           RERUN ON DF-1-700-X EVERY 600 RECORDS
           OF DISK-FILE.
                       .
                       .
                       .
                       .
                       .
           OPEN INPUT CARD-FILE.
           OPEN OUTPUT PRINT-FILE.
           OPEN I-O DISK-FILE.
```

The I-O-CONTROL paragraph defines some of the special techniques to be used in a program.

```
I-O-CONTROL.
RERUN clause.
SAME AREA clause ... .
```

The I-O-CONTROL paragraph is an optional part of the Environment Division. The presence of a RERUN clause specifies that checkpoint records are to be taken. A checkpoint record is a recording of the entire contents of main storage at a desired interval. The contents of main storage are recorded on a disk and can be read back into core storage to restart the program from that point. The following Environment Division demonstrates the optional features.

```
0   0    1    1    2    2    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.
           SELECT IN-FILE ASSIGN TO DF-2-5000-X ACCESS IS
           SEQUENTIAL.
           SELECT OUT-FILE ASSIGN TO PR-1132-C
               RESERVE NO ALTERNATE AREAS.
           SELECT PR-FILE ASSIGN TO DF-2-300-X
               ACCESS IS SEQUENTIAL.

       I-O-CONTROL.
           RERUN ON DF-1-3000-X EVERY 2000 RECORDS
           OF IN-FILE.
           SAME AREA FOR IN-FILE PR-FILE.
```

The SAME AREA clause indicates that two or more files are to use the same core storage during processing. The area to be shared includes all storage areas assigned to the file. Therefore files that are grouped in the SAME AREA clause should be all open at the same time.

SUMMARY:

In this lesson you have learned to access a random file randomly when it is opened as INPUT and to add records to a random file when it is opened as I-O. In the next lesson you will learn to update random files, both sequentially and randomly.

END OF LESSON 36

LESSON 37

LESSON 37 - RANDOM FILE UPDATING

INTRODUCTION

In this lesson you will learn to update a random file and replace updated records in the file. All features are discussed as they apply to 1130 installations.

At the end of this lesson, you will code a complete program that will include a routine to add disk records and two updating disk routines. When you have coded the program, you will have demonstrated many skills required of a COBOL programmer.

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..

           READ file-name [INTO identifier]

               INVALID KEY imperative-statement
```

Write a statement to access a record from a randomly accessed file named RECORD-FILE that was opened as I-O. Use the statement GO TO MESSAGE in the INVALID KEY 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..

           READ RECORD-FILE
               INVALID KEY GO TO MESSAGE.
```

----

2. The INVALID KEY option of a READ statement that refers to a randomly accessed file opened as I-O is activated under the same circumstances as the INVALID KEY option of a READ statement that refers to a randomly accessed file opened as INPUT. The INVALID KEY option of any READ statement would be activated when:

   a. a record is out of sequence.

   b. the file is filled.

   c. no record with a record key equal to the current value of the actual key variable can be located in the file.

* * *

c

----

3. Before a READ statement can be executed for a randomly accessed file, the:

   a. actual key variable must be set to the desired value.

   b. file must be opened as I-O or INPUT.

* * *

Both of these

----

4. After a READ statement is executed for a random file opened as I-O, the accessed record may be updated and placed back into the same position in the file. An updated record is placed back into the file with a WRITE statement. The next I/O statement for a random file opened as I-O after a READ statement is executed may be a:

   a. statement to place the record back into the file.

   b. WRITE statement.

* * *

Both
(The READ statement and its associated WRITE statement may be separated by any number of statements as long as they are not separated by any other I/O statement that refers to the random file.)

Guide to File Maintenance for Disk Files in 1130 Installations

Desired Effect	Access	Required Statements	Required Options	Optional Options
Update a record in the file	SEQUENTIAL	READ	AT END	INTO
	RANDOM	READ	INVALID KEY	INTO
Add a record to the file	RANDOM	WRITE	INVALID KEY	FROM

Figure 196

5. Figure 196 shows that the WRITE statement can refer to a file opened as I-O that is being accessed:

   a. sequentially.

   b. randomly.

* * *

Either

---

6. Refer to Figure 196. A WRITE statement must be preceded by a READ statement with the:

   a. INVALID KEY option if accessing is sequential.

   b. AT END option if accessing is random.

* * *

Neither (The opposite is true.)

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]

                   INVALID KEY imperative-statement
```

The format of the WRITE statement is shown above. The record-name in the statement is the name of a record associated with a sequential file that was opened as I-O. The record that is placed back into the file is the last record accessed by a READ statement referring to that file.

Which statement will place the most recent PART-RECORD accessed by a READ statement referring to the sequential file PART-FILE back into the 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..

               WRITE PART-RECORD.
```

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 PART-FILE.
```

* * *

a

---

8. When a file is to be randomly updated, the INVALID KEY option must be included in every READ statement referring to that file. The INVALID KEY option of the READ statement would be activated when:

   a. the key of a record being added to the file duplicates the key of a record already in the file.

   b. a record with a key equal to the current value of the actual key variable is not in the file.

* * *

b

---

9. If the INVALID KEY option of a READ statement is activated, no record has been accessed. The next I/O statement for a random file after the INVALID KEY option of a READ statement is activated could be:

   a. a WRITE statement to place the record back into the file.

   b. a READ statement to access a different record.

* * *

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..

           WRITE record-name [FROM identifier]

               INVALID imperative-statement
```

Write a statement to place the last STUDENT-RECORD accessed from STUDENT-FILE back into 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..

           WRITE STUDENT-RECORD INVALID KEY GO TO ERROR.
```

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 STUDENT-RECORD INVALID GO TO ERROR.
```

The statement above places the last STUDENT-RECORD accessed back into STUDENT-FILE. Refer to Figure 192. STUDENT-FILE must be:

   a. sequentially accessed file opened as I-O.

   b. randomly accessed, sequential file opened as I-O whose KEY field contains the location of the last record read.

* * *

Either of these

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..

           WRITE STUDENT-RECORD INVALID GO TO ERROR.
```

According to Figure 196, the most recent I/O statement executed prior to the WRITE statement above could have been:

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 STUDENT-FILE
               AT END GO TO TERMINAL.
```

if STUDENT-FILE were being accessed randomly.

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 STUDENT-FILE
               INVALID KEY GO TO MISTAKES.
```

if STUDENT-FILE were being accessed sequentially.

*   *   *

Neither (The opposite is true.)

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..

           WRITE STUDENT-RECORD
               FROM WORK-RECORD.
```

The use of the FROM option in a WRITE statement has the same effect as the INTO option in a READ statement. When the statement above is executed:

a. STUDENT-RECORD is moved to WORK-RECORD, and then placed back into the file.

b. WORK-RECORD is moved to STUDENT-RECORD, and then placed back into the file.

*   *   *

b

---

14. Write a statement that will move the data in WORK-SECTION to INVENTORY-RECORD and then place the record back into INVENTORY-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..

        WRITE INVENTORY-RECORD
            FROM WORK-SECTION.
```

---

15. Match the statements that could be used to update records in a file with the access methods for the disk I-O files below.

1) RANDOM

2) SEQUENTIAL

a. READ with AT END

b. READ with INVALID KEY

c. WRITE with INVALID KEY

*　　　*　　　*

1) b, c
2) a, 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. CHANGES.
       ENVIRONMENT DIVISION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.


           SELECT UPDATE-DATA.
               ASSIGN TO RD-1442.
       DATA DIVISION.
       FILE-SECTION.
       FD  STUDENT-MASTER-FILE
           LABEL RECORDS ARE STANDARD.
       01  STUDENT-DATA-RECORD.
           02  PERSONAL.
               03  YEAR-IN PIC XX.
               03  S-NUMBER PIC X(9).
               03  NAME PIC X(25).
               03  S-ADDRESS.
                   04  STREET PIC X(15).
                   04  CITY PIC X(10).
                   04  STATE PIC X(5).
           02  SCHOLASTIC.
               03  GRAD PIC XX.
               03  DEGREE PIC XX.
               03  GPA PIC X(4).
               03  MAJOR PIC X(3).
               03  MINOR PIC X(3).
       FD  UPDATE-DATA-FILE
           LABEL RECORDS ARE OMITTED.
       01  CHANGE-DATA-RECORD.
           02  C-NUMBER PIC X(9).
           02  C-ADDRESS.
               03  C-STREET PIC X(15).
               03  C-CITY PIC X(10).
               03  C-STATE PIC X(5).
       WORKING-STORAGE SECTION.
       77  KEY-NUMBER PIC S99999 USAGE COMP.
```

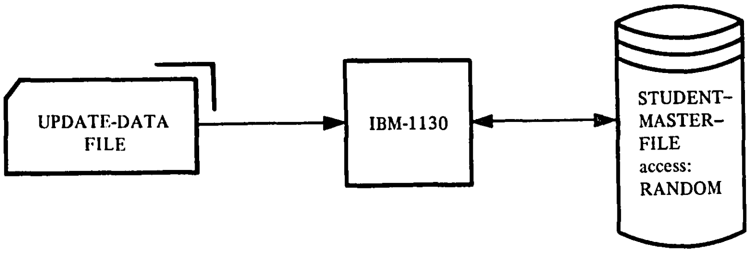


Figure 197

16. Tucumcari Community College has a problem with students who move frequently and needs a program to update addresses in its random master file. Figure 197 shows a system flow chart and parts of the first three divisions of program CHANGES in which records will be updated in random order. Write a FILE-CONTROL entry for file STUDENT-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..

            SELECT STUDENT-MASTER-FILE
                ASSIGN TO DF-6-900-X
                ACTUAL KEY IS KEY-NUMBER
                ACCESS IS RANDOM.
```

17.

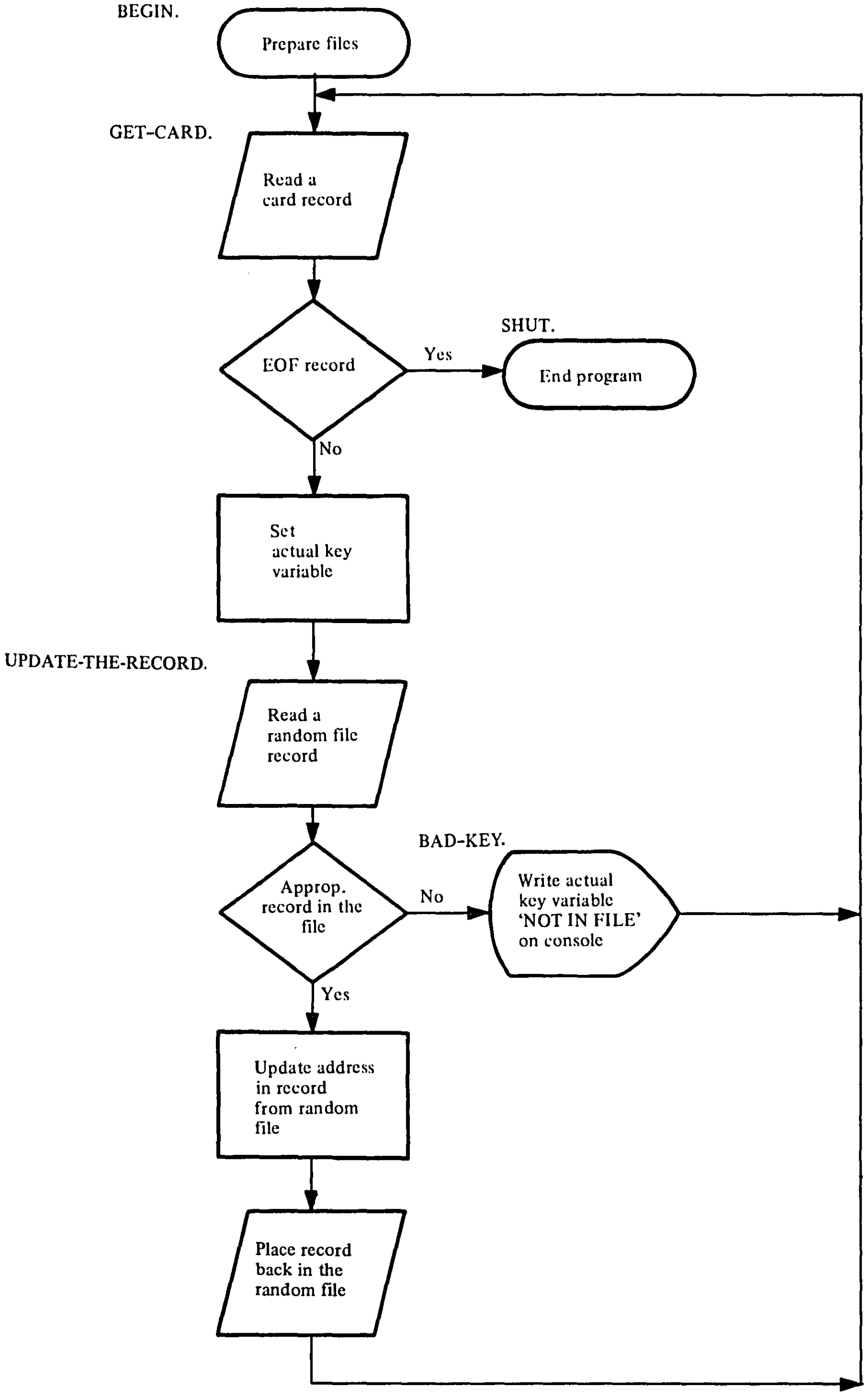
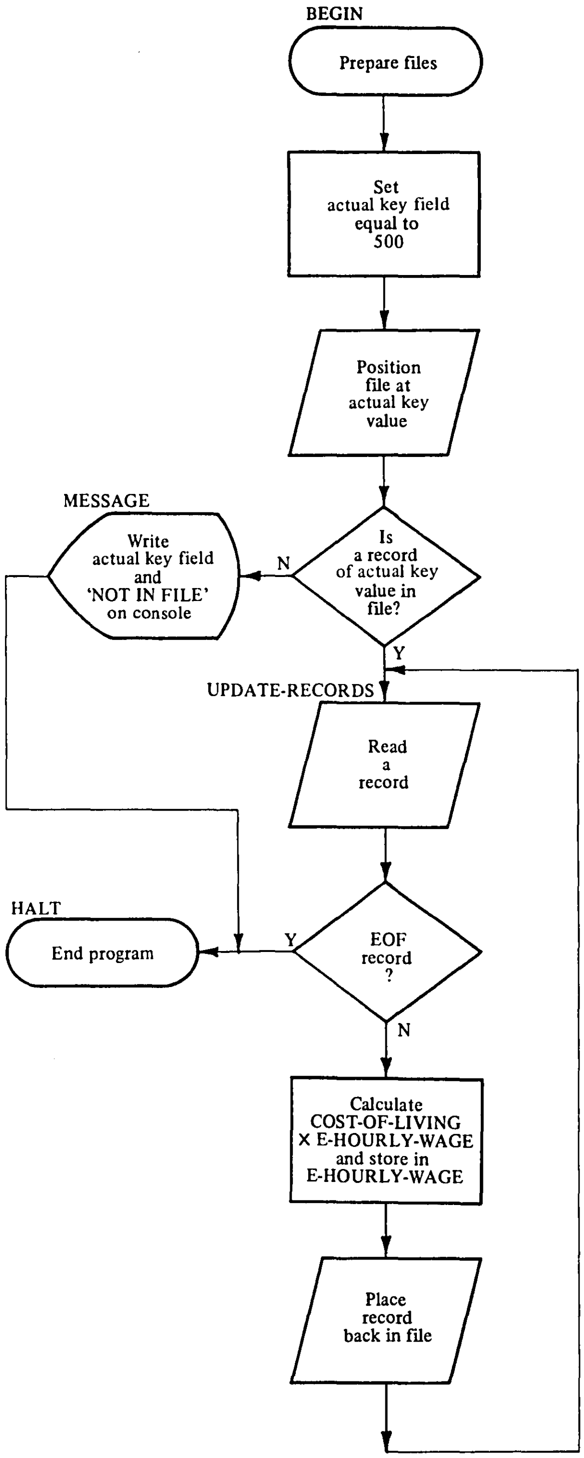

BEGIN.
Prepare files
GET-CARD.
Read a card record
EOF record
Yes
SHUT.
End program
No
Set actual key variable
UPDATE-THE-RECORD.
Read a random file record
Approp. record in the file
No
BAD-KEY.
Write actual key variable 'NOT IN FILE' on console
Yes
Update address in record from random file
Place record back in the random file

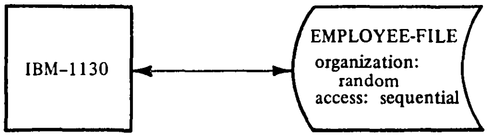
Figure 198

Follow the flow chart above and write the Procedure Division for the program CHANGES. Refer to the preceding Figure 197.

*   *   *

```
0   0    1    1    2    2    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 UPDATE-DATA-FILE
               I-O STUDENT-MASTER-FILE.
       GET-CARD.
           READ UPDATE-DATA-FILE
               AT END GO TO SHUT.
           MOVE C-NUMBER TO KEY-NUMBER.
       UPDATE-THE-RECORD.
           READ STUDENT-MASTER-FILE
               INVALID KEY GO TO BAD-KEY.
           MOVE C-STREET TO STREET.
           MOVE C-CITY TO CITY.
           MOVE C-STATE TO STATE.
           WRITE STUDENT-DATA-RECORD INVALID KEY
               GO TO BAD-KEY.
           GO TO GET-CARD.
       BAD-KEY.
           DISPLAY KEY-NUMBER
               'NOT IN FILE' UPON CONSOLE.
           GO TO GET-CARD.
       SHUT.
           CLOSE UPDATE-DATA-FILE
               STUDENT-MASTER-FILE.
           STOP RUN.
```


BEGIN
Prepare files
Set actual key field equal to 500
Position file at actual key value
MESSAGE
Write actual key field and 'NOT IN FILE' on console
N
Is a record of actual key value in file?
Y
UPDATE-RECORDS
Read a record
HALT
End program
Y
EOF record ?
N
Calculate COST-OF-LIVING X E-HOURLY-WAGE and store in E-HOURLY-WAGE
Place record back in file


IBM–1130
EMPLOYEE-FILE
organization: random
access: sequential

```
0   0    1    1    2    2    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. FIX-SEQUENTIALLY.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.


       DATA DIVISION.
       FILE SECTION.
       FD  EMPLOYEE-FILE
           LABEL RECORDS ARE STANDARD
           BLOCK CONTAINS 10 RECORDS.
       01  EMPLOYEE-RECORD.
           02  EMPLOYEE-NUMBER PIC X(4).
           02  E-NAME PIC X(30).
           02  E-ADDRESS PIC X(30).
           02  E-HOURLY-WAGE PIC S99V99
                   USAGE COMP.
           02  DATE-HIRED PIC 9(6).
       WORKING-STORAGE SECTION.
       77  CLOCK-NUMBER PIC S9(5) COMP.
       77  COST-OF-LIVING
               USAGE COMP PIC S9V99
               VALUE +1.06.
```

Figure 199

18. Figure 199 gives the first three divisions of the program FIX-SEQUENTIALLY except for the FILE-CONTROL paragraph. The system flow chart in this figure indicates that only one file is used in the program. The program flow chart shows that the program updates all records sequentially beginning with the record that has a record key equal to 500. Write a FILE-CONTROL paragraph for the program.

* * *

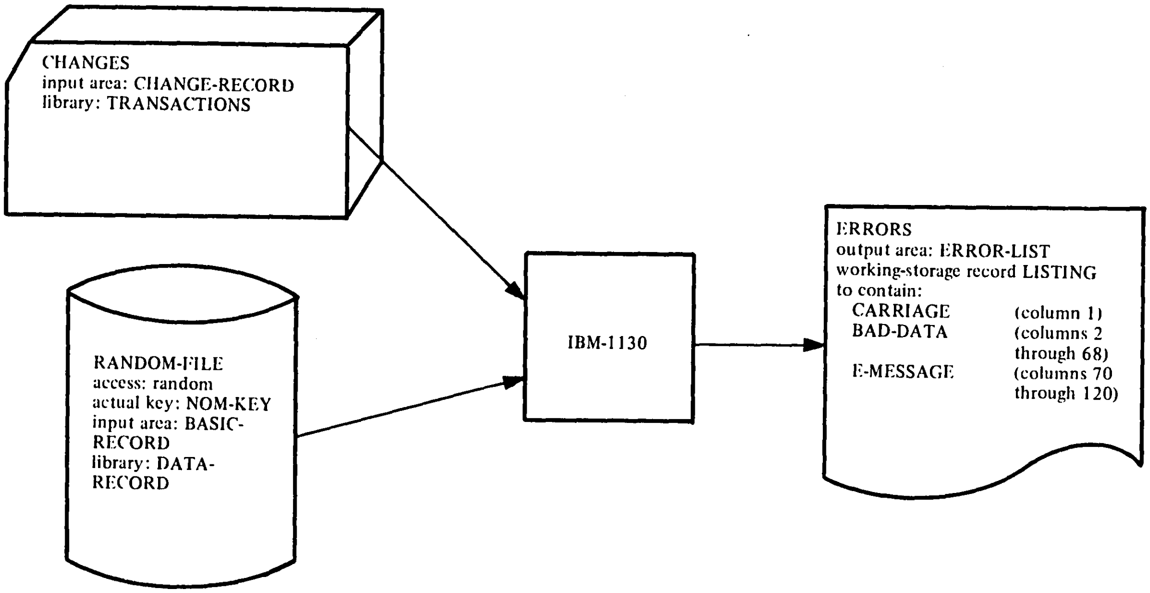
```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..

           SELECT EMPLOYEE-FILE
               ASSIGN TO DF-12-600-X
               ACTUAL KEY IS CLOCK-NUMBER.
```

(ACCESS IS SEQUENTIAL could be included.)

19. The program represented in Figure 199 will update all records whose key values are 500 or higher. The hourly wage rate for these employees will be increased 6 percent to keep up with the cost of living. Follow the flow chart and write the Procedure Division for FIX-SEQUENTIALLY.

* * *

```
0   0    1    1    2    2    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 I-O EMPLOYEE-FILE.
           MOVE 0500 to CLOCK-NUMBER.
       UPDATE-RECORDS.
           READ EMPLOYEE-FILE
               INVALID KEY GO TO HALT.
           COMPUTE E-HOURLY-WAGE =
               E-HOURLY-WAGE *
               COST-OF-LIVING.
           WRITE EMPLOYEE-RECORD INVALID GO TO MESSAGE.
           ADD 1 TO CLOCK-NUMBER
           GO TO UPDATE-RECORDS.
       MESSAGE.
           DISPLAY CLOCK-NUMBER
               'NOT IN FILE' UPON CONSOLE.
       HALT.
           CLOSE EMPLOYEE-FILE.
           STOP RUN.
```



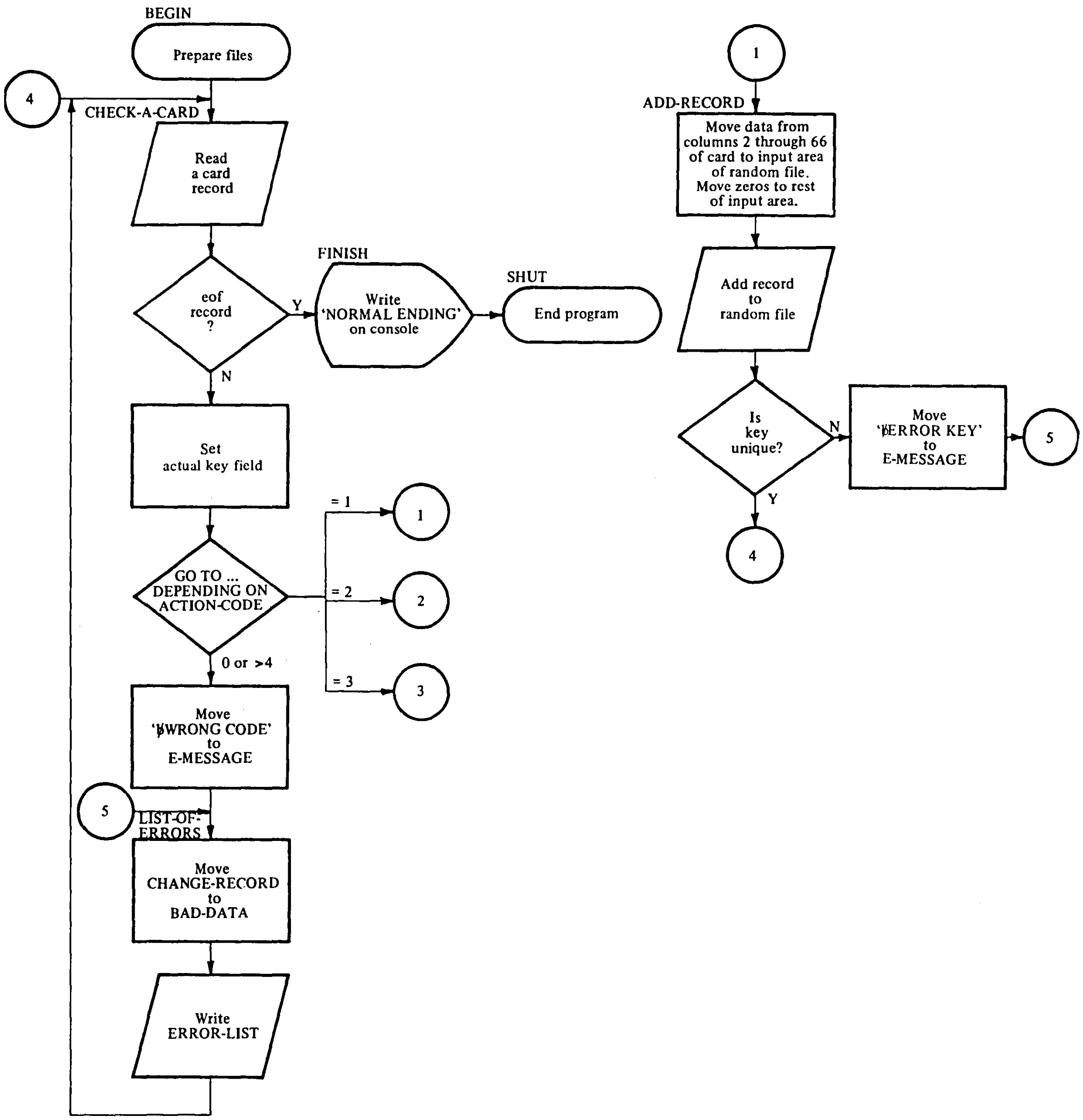
Library name: DATA-RECORD
Text

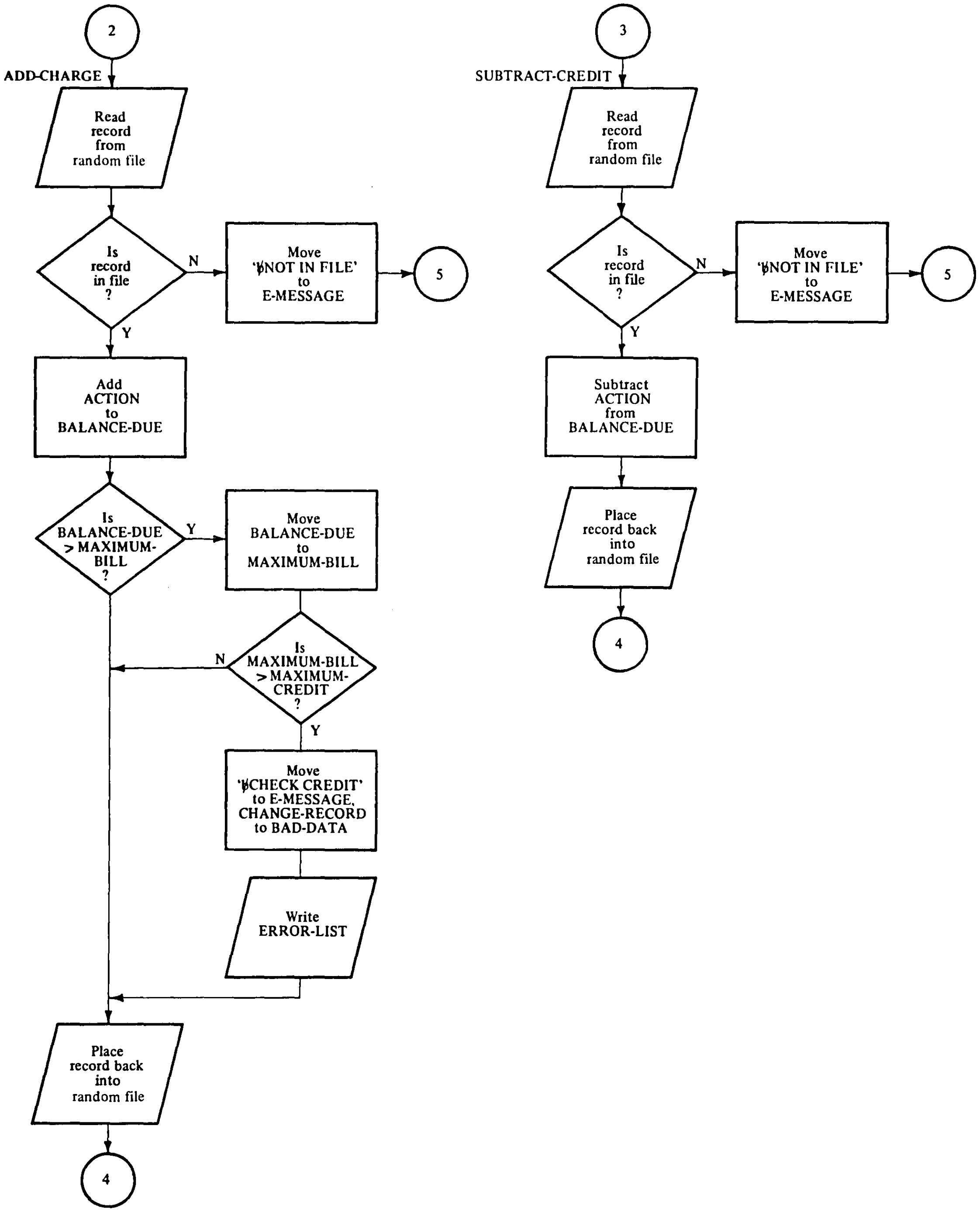
```
0   0    1    1    2    2    3    3    4    4    5    5    6    6    7
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0..
           02 PERSONAL.
               03 NAME PIC X(21).
               03 CUSTOMER-NUMBER PIC X(6).
               03 C-ADDRESS PIC X(30).
           02 PAY-RECORD.
               03 YEAR-OPENED PIC XX.
               03 MAXIMUM-CREDIT
                       USAGE COMP PIC 9999V99.
               03 MAXIMUM-BILL
                       USAGE COMP PIC 9999V99.
               03 BALANCE-DUE
                       USAGE COMP PIC 9999V99.
               03 PAYCODE PIC 9.
```

Library name: TRANSACTIONS
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  ACTION-CODE PIC 9.
           02  CREDIT-CARD PIC S9(5) COMP.
           02  NEW-NAME PIC X(21).
           02  N-ADDRESS PIC X(30).
           02  YEAR PIC XX.
           02  MAX-CREDIT PIC 9999V99.
           02  FILLER PIC X(4).
           02  ACTION PIC 9999V99.
           02  FILLER PIC X(4).
```

Figure 200


BEGIN
Prepare files
4
CHECK-A-CARD
Read a card record
eof record ?
Y
N
FINISH
Write 'NORMAL ENDING' on console
SHUT
End program
Set actual key field
GO TO ... DEPENDING ON ACTION-CODE
= 1
1
= 2
2
= 3
3
0 or >4
Move '␢WRONG CODE' to E-MESSAGE
5
LIST-OF-ERRORS
Move CHANGE-RECORD to BAD-DATA
Write ERROR-LIST
1
ADD-RECORD
Move data from columns 2 through 66 of card to input area of random file. Move zeros to rest of input area.
Add record to random file
Is key unique?
N
Move '␢ERROR KEY' to E-MESSAGE
5
Y
4



2
ADD-CHARGE
Read record from random file
Is record in file ?
N
Move '₿NOT IN FILE' to E-MESSAGE
5
Y
Add ACTION to BALANCE-DUE
Is BALANCE-DUE > MAXIMUM-BILL ?
Y
Move BALANCE-DUE to MAXIMUM-BILL
Is MAXIMUM-BILL > MAXIMUM-CREDIT ?
N
Y
Move '₿CHECK CREDIT' to E-MESSAGE, CHANGE-RECORD to BAD-DATA
Write ERROR-LIST
Place record back into random file
4
3
SUBTRACT-CREDIT
Read record from random file
Is record in file ?
N
Move '₿NOT IN FILE' to E-MESSAGE
5
Y
Subtract ACTION from BALANCE-DUE
Place record back into random file
4

Figure 201

20. HAL Corporation maintains a random file of its customer accounts. Every week the file is updated by adding records for new accounts and adjusting the record when a charge or credit is received.

Figure 200 shows a system flow chart for a program to update the file. All of the update data is contained in the randomly ordered card file CHANGES. The first column of each card (see library text TRANSACTIONS in the same figure) contains ACTION-CODE, which signifies the action to be taken for each record. A 1 indicates data from the card record is to be added to the file as a record; a 2 indicates that a charge is to be added to the balance in the corresponding record; and a 3 indicates that a credit is to be subtracted from the balance in the corresponding record. Any records in the card file which are in error, such as having an incorrect ACTION-CODE, are to be moved to BAD-DATA, and error messages printed as indicated in the flow charts.

Figure 201 shows the program flow chart for the same problem. Follow this flow chart and write the entire updating program FINAL-ONE.

*　　　*　　　*

```
0   0    1    1    2    2    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. FINAL-ONE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1130.
       OBJECT-COMPUTER. IBM-1130.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RANDOM-FILE
               ASSIGN TO DF-10-800-X
               ACTUAL KEY IS NOM-KEY
               ACCESS IS RANDOM.
           SELECT CHANGES
               ASSIGN TO RD-1442.
           SELECT ERRORS
               ASSIGN TO PR-1132-C.
       DATA DIVISION.
       FILE SECTION.
       FD  RANDOM-FILE
           LABEL RECORDS ARE STANDARD
       01  BASIC-RECORD COPY DATA-RECORD.
       FD  CHANGES
           LABEL RECORDS ARE OMITTED.
       01  CHANGE-RECORD COPY TRANSACTIONS.
       FD  ERRORS
           LABEL RECORDS ARE OMITTED.
       01  ERROR-LIST PIC X(121).
       WORKING-STORAGE SECTION.
       77  NOM-KEY PIC S99999 USAGE COMP.
       77  ACTION-COMP-3 PIC 9999V99
               USAGE IS COMP.
       01  LISTING.
           02  CARRIAGE PIC X.
           02  BAD-DATA PIC X(68).
           02  E-MESSAGE PIC X(52).
       PROCEDURE DIVISION.
       BEGIN.
           OPEN INPUT CHANGES
               OUTPUT ERRORS
               I-O RANDOM-FILE.
       CHECK-A-CARD.
           READ CHANGES
               AT END GO TO FINISH.
           MOVE CREDIT-CARD TO NOM-KEY.
           GO TO ADD-RECORD
               ADD-CHARGE SUBTRACT-CREDIT
               DEPENDING ON ACTION-CODE.
           MOVE 'WRONG CODE' TO E-MESSAGE.
       LIST-OF-ERRORS.
           MOVE CHANGE-RECORD TO BAD-DATA.
           WRITE ERROR-LIST FROM LISTING.
           GO TO CHECK-A-CARD.
```

```
ADD-RECORD.
    MOVE CREDIT-CARD TO CUSTOMER-NUMBER.
    MOVE NEW-NAME TO NAME.
    MOVE N-ADDRESS TO C-ADDRESS.
    MOVE YEAR TO YEAR-OPENED.
    MOVE MAX-CREDIT TO MAXIMUM-CREDIT.
    MOVE ZEROS TO MAXIMUM-BILL
        BALANCE-DUE PAYCODE.
    WRITE BASIC-RECORD INVALID KEY
        MOVE 'ERROR KEY'
            TO E-MESSAGE
        GO TO LIST-OF-ERRORS.
    GO TO CHECK-A-CARD.
ADD-CHARGE.
    READ RANDOM-FILE
        INVALID KEY
        MOVE 'NOT IN FILE' TO E-MESSAGE
        GO TO LIST-OF-ERRORS.
    MOVE ACTION TO ACTION-COMP-3.
    ADD ACTION-COMP-3 TO BLANCE-DUE.
    IF BALANCE-DUE IS GREATER THAN
        MAXIMUM-BILL
        MOVE BALANCE-DUE TO MAXIMUM-BILL
        IF MAXIMUM-BILL IS GREATER THAN
            MAXIMUM-CREDIT
            MOVE ' CHECK CREDIT'
                TO E-MESSAGE
            MOVE CHANGE-RECORD
                TO BAD-DATA
            WRITE ERROR-LIST
                FROM LISTING.
    WRITE BASIC-RECORD INVALID                      (17)
        MOVE 'NOT IN FILE' TO E-MESSAGE
        GO TO LIST-OF-ERRORS.
    GO TO CHECK-A-CARD.
SUBTRACT-CREDIT.
    READ RANDOM-FILE
        INVALID KEY
        MOVE ' NOT IN FILE' TO E-MESSAGE
        GO TO LIST-OF-ERRORS.
    MOVE ACTION TO ACTION-COMP-3.
    SUBTRACT ACTION-COMP-3
        FROM BALANCE-DUE.
    REWRITE BASIC-RECORD.                           (17)
    GO TO CHECK-A-CARD.
FINISH.
    DISPLAY 'NORMAL ENDING'
        UPON CONSOLE.
SHUT.
    CLOSE CHANGES ERRORS
        RANDOM-FILE.
    STOP RUN.
```

SUMMARY

You have now completed the lesson. This lesson concludes the sequence in which you have learned to code programs to create, process, and maintain random files.

END OF LESSON 37

LESSON 38

LESSON 38 - 1130 COBOL WITHIN THE MONITOR SYSTEM

INTRODUCTION

1130 COBOL operates as a systems program under the disk monitor. This section is designed to review the disk monitor as it applies to 1130 COBOL only.

The 1130 Monitor has associated with it a number of systems programs. In addition to COBOL, the core load builder and the disk utility programs provided by the monitor are of great importance. These are reviewed in this section. Since this review is incomplete, the reader is strongly urged to use the Programming and Operator's Guide for the 1130 Disk Monitor along with this manual.

1130 COBOL provides for a number of compile time options. Since the control cards used to specify these options are normally provided by the programmer, these are also reviewed in this section, along with certain other operational features of interest to the programmer. Since again this review is incomplete, the reader is strongly urged to use the 1130 COBOL Operations Manual and the 1130 COBOL Programmers Guide along with this manual.

This lesson will require approximately one and one quarter hour.

1. The 1130 Disk Monitor System provides for the continuous operation of the 1130 Computing System, with minimal set-up time and operator intervention, in a stacked job environment. The Monitor System consists of nine distinct but interdependent elements -- Supervisor, Disk Utility Program, COBOL Compiler, FORTRAN Compiler, RPG Compiler, Core Load Builder, Core Image Loader, Assembler, and System Library.

   The beginning programmer will not be directly concerned with all of these elements. Therefore, only certain parts will be discussed in this course.

   Complete this sentence:

   The 1130 Disk Monitor System provides for the .... operation of the 1130 Computing System with minimal .... time and .... intervention in a .... environment.

* * *

continuous
set-up
operator
stacked job

---

2. The Disk Utility Program (DUP) is a group of IBM-supplied programs that perform operations involving the disk such as storing, moving, deleting and dumping data and/or programs.

   The Core Load Builder constructs core image programs from mainline object programs. The mainline programs and all necessary subprograms are converted into Disk Core Image format from disk system format, and the resultant core load is built for immediate execution or for storing for future execution.

   The Core Image Loader serves as both a loader for core loads and as an interface for the Monitor programs.

   The System Library is a group of disk-resident programs that performs I/O, data conversion, arithmetic, disk initialization, and maintenance functions.

   1. What operation does the Disk Utility Program do?

   2. Name 3 functions performed by the System Library.

* * *

1. It performs operations involving the disk such as storing, moving, deleting, and dumping data and/or programs.

2. I/O, data conversion, arithmetic, disk initialization, and maintenance functions.

---

3. A job is a specified unit of work to be performed under control of the disk monitor system. A typical job might be the processing of a COBOL program -- compiling Job definition -- the process of specifying the work to be done during a single job -- allows the programmer considerable flexibility. A job can include as many or as few job steps as the programmer desires.

Match the items in the two columns:

1.	Job	a.	Unit of work within a job
2.	Example of a job	b.	Specified unit of work
3.	Job step	c.	A program loop
		d.	Processing a COBOL compilation

* * *

1. b
2. d
3. a

---

4. A job step is exactly what the name implies -- one step in the processing of a job. Thus, in the job mentioned above, one job step is the compilation of source statements; another is the building of a core load; another is the execution of a core load. In contrast to a job definition, the definition of a job step is fixed. Each job step involves the execution of a program, whether it be a program that is part of the Disk Monitor System or a program that is written by the user. A compilation requires the execution of the COBOL compiler. Similarly, core load building implies the execution of the core load builder. Finally, the execution of a core load is the execution of the problem program itself.

   1. A .... is one step in the processing of a job.

   2. Execution of a core load is execution of a problem program.

* * *

1. Job step
2. True

---

5. A typical job falls into one of the following categories: Compile Only, Build Core Load Only, Compile and Build Core Load, Execute Only, Build a Core Load and Execute, Compile, Build Core Load, and Execute.

Three elements found in typical jobs are:

........, ... and ....

* * *

Compilations
Building Core Loads
Executions

---

6. A Compile Only job involves only the execution of the COBOL compiler. It is useful when checking for errors in COBOL source statements.

The .... type of job is useful in checking for errors in COBOL source statements.

* * *

Compile Only

---

7. A Build Core Load Only job involves only the execution of the core load builder. It is used primarily to combine modules produced in previous compile only jobs, and to check that all cross references between modules have been resolved.

The .... type of job combines and cross checks program modules.

* * *

Build Core Load Only

---

8. A Compile and Build Core Load combines the functions of the compile-only and the core load-building-only jobs. It requires the execution of both the COBOL compiler and the Core Load Builder.

A Compile and Build Core Load requires the execution of:

a. the COBOL Compiler

b. the Core Load Builder

* * *

Both

---

9. An Execute Only job involves the execution of core load produced in a previous job. Once a COBOL program has been compiled and its core load has been built successfully, it can be retained as one or more core loads and executed whenever needed.

Once a COBOL program compiles and its core load has been successfully built it must be executed immediately.

a. True

b. False

* * *

b. Once the core load is successfully built, the object program may be executed whenever needed.

---

10. A Build Core Load and Execute job combines the functions of the Core Load Build Only and the Execute Only jobs. This type of job requires the execution of:

    a. the COBOL compiler

    b. the Core Load Builder

    c. the program built in core

* * *

b,c

---

11. The Compile, Build Core Load and Execute type of job combines the functions of the Compile Only, Build Core Load Only and Execute jobs. It calls for the execution of:

    a. the COBOL compiler

    b. the Core Load Builder

    c. the problem program

* * *

All of these. This type of job processes the COBOL program completely.

---

12. When considering the definition of his job, the programmer should be aware of the following: <u>if a job step is cancelled during execution, the entire job is terminated; any remaining job steps are skipped</u>. Thus, in a compile-core load building and execute job,a failure in compilation precludes the core loads of the module and core load execution. Similarly, a failure in core load precludes core load execution.

    Name two results of the cancellation of a job step during execution.

* * *

1. The entire job is terminated
2. Remaining job steps are skipped

---

13. Once the programmer has decided the work to be done within his job and how many job steps are required to perform the job, he can then define his job by writing monitor control records. Since these records are usually punched in cards, the set of monitor control records is referred to as a job deck. In addition to monitor control records, the job deck can include input data for a program that is executed during a job step. For example, input data for the COBOL compiler -- the COBOL program to be compiled -- can be placed in the job deck.

   1. A set of monitor control records is called a ....

   2. A job deck can include .... for a program executed during a job ....

   3. The .... defines a job.

* * *

1. job deck
2. input data, step
3. programmer

---

14. All of the following monitor control records need not appear in the job deck. The programmer will select those that he needs to accomplish his job only. The monitor control records are summarized briefly:

// JOB	Defines the start of a new job. Must be present for all jobs.
// COBOL	Causes the supervisor to read the COBOL compiler into core storage and transfer control to it.
// DUP	Causes the supervisor to read the control portion of the Disk Utility Program into core storage and transfer control to it.
// XEQ	Causes the supervisor to initialize for core load execution, and results in the execution of a job step.
// PAUS	Causes the supervisor to wait.
// *	Allows the user to print alphameric text on the listing.

Monitor control records perform the load and control functions of the Monitor System. All monitor control records begin with // in columns 1 and 2, and a blank in column 3.

When shown in the control record format, a blank position enclosed by text indicates that the column must be blank. Remarks may be punched in the card columns listed as "not used" in the control record formats.

After compilation, the object program will have been stored in Working Storage on disk in DSF format, in exactly the same format as would be produced for a program compiled using any other compiler on the 1130.

Of the card types listed above, identify which 3 would be necessary to compile and execute a program on the same run.

* * *

```
// JOB
// COBOL
// XEQ
```

---

15. A typical job stream for compilation of a COBOL program might appear as follows:

```
Card #1:    // JOB
            // COBOL
            *XXXX,XXXX (etc.)
               OPTIONAL CONTROL CARD
            SOURCE PROGRAM DECK
Last Card:  /*
```

If any compile time option cards are to be specified (control cards identified with an * symbol in cc 1), these must follow the // COBOL Control Card, and precede the actual COBOL source cards. After the last COBOL source card, a card identified with /* in cc 1-2 must be present.

Compile time option control cards are throroughly discussed in the Operation's Manual. Most installations will define a standard set of assumptions, eliminating the need for most compile time option control cards. For the first time user, however, note that unless the LIST option card has been provided for, no source listing will be produced. At a minimum, compilation of programs newly written (especially if written by programmers new to the COBOL language) should specify the LIST, DMAP, PMAP, DUMP, STNO and /1FE options.

1. Which card comes last in a job stream?

2. Which cards contain an asterisk in column 1?

3. Most installations define a ....

* * *

1. /* in cc 1-2.
2. compile time option cards
3. Standard set of assumptions

---

16. // JOB

The JOB control record defines the start of a new job. It causes the supervisor to perform the job initialization procedure.

If a T is punched in column 8, the disk output from the job to be performed will be only temporarily stored i.e; this output will be deleted from the disk automatically at the beginning of the next job.

Answer true or false:

1. The JOB control record defines the start of a new job.

2. A T in column 8 causes the job to be stored permanently.

* * *

1. True
2. False. The job disk output will be deleted at the start of the next job.

---

17. // COBOL

This control record causes the supervisor to load the COBOL compiler into core storage and transfer control to it. Any COBOL control records and the source statements to be compiled must follow this control record. Comments control records (//*) may not follow this control record.

The format of the COBOL control record is as follows:

Card Column	Contents	Notes
1-8	// COBOL	
9-72	Reserved	
73-80	Not used	

Match the items in the columns:

1. // COBOL control record
2. Source statements
3. Comments control records

a. May not follow the // COBOL card.

b. Activates the // COBOL compiler.

c. Must follow the // COBOL card.

d. End a job.

* * *

1. b
2. c
3. a

---

18. <u>// DUP</u>

This control record causes the supervisor to load the control portion of the Disk Utility program into Core Storage, and transfer control to it. The Disk Utility Program (DUP) provides the user with the ability to perform the following operations through the use of control records:

- Programs and Data Files stored on disk
- Print and Punch
- Remove programs and data files from disk
- Determine status of disk
- Modify the system

1. The Disk Utility Program stores .... and .... on disk.

2. It controls .... and punching of information stored on disk.

3. It determines .... of the disk.

* * *

1. programs, data files
2. printing
3. status

---

19. // XEQ

This control record causes the supervisor to initialize for core load execution. If the name of a program is omitted from cc 8-12 (leaving these columns blank), then the last program compiled but not yet stored from Working Storage is selected. By this means, a new program may be compiled and tested under execution without the necessity of storing it permanently, or of even giving it a name.

/*

End-of-file control card. Identifies to the compiler that the last COBOL source statement has been read. The COBOL language does not provide for an "END" card statement.

1. Which type of control record card identifies the end of a source deck?

2. Which type causes execution of a program?

* * *

1. /*, End-of-File
2. // XEQ, Execute

---

20. During the compilation of a COBOL program, the following types of output can occur on the 1130 line printer:

- Monitor Control card images
- Compiler Control card images
- [Forms ejection]
- [Program title in heading]
- [Source Program]
- [Forms ejection]
- [DATA DIVISION Map]
- [PROCEDURE DIVISION Map]
- [Forms ejection]
- Diagnostics, if any
- Program Size Message (If successful)
- End of Compilation Message

[ ] Indicates that this output is at the programmer's option, through the use of compile time option cards.

During the execution of a program compiled by COBOL, line printer output can only be produced by specific coding included in the object programs provided for by the source programmer.

1. What do you think a card image is?

2. What do you think a map is?

3. Will you always get a Program Size Message after compilation?

* * *

1. Card image: printed representation of the contents of a punched card.

2. Map: printed representation of the contents of a part of core storage, including data and/or program.

3. Program Size Message: printed only if the program is completely compiled.

---

21. All control statements read in card form will always print at the start of a compilation. If an *EJCT compile time option is in effect, a new page will be started before listing any further data.

If the *LIST option is in effect, a source listing will follow the listing of the control satements. All compiler options in effect at the time will be summarized at the top on an option line.

In the COBOL source statement format, page-line sequence numbers may be punched into card columns 1-6. When listing a source program, 1130 COBOL will print the page-line numbers punched into a card at the extreme right margin, and not at the left. This has been done to permit the more important compiler-generated statement numbers to print at the left.

Match items in the two columns:

1.	*EJCT card	a.	Causes ejection to a new page.
2.	*LIST card	b.	Ends a job
3.	Page-line numbers	c.	Causes printing of a source listing.
		d.	Print on the right of the source program listing.

*   *   *

1. a
2. c
3. d

---

22. As the COBOL source program is listed, the compiler generates statement numbers which it will subsequently use for all references back to an individual source statement. A statement in 1130 COBOL is defined as that portion of the source text which starts with:

- anything beginning in Area "A" in any Division
- the word SELECT in the Environment Division.
- A level number in the Data Division
- the word IF in the Procedure Division
- any verb in the Procedure Division except NOTE and ENTER

Statements terminate just prior to the start of the next statement, or the end of the program.

Why are statement numbers important?

*   *   *

All references back to individual source statements by the compiler are made via these numbers.

---

23. After the statement number is printed at the left of each line, each input card will then list. At the top, the print positions identifying the start of area A and area B, corresponding with data punched starting in card columns 8 and 12 is identified. At the end of the listing, the control card containing the /* identifying the end of the COBOL source statements is listed.

If the *EJCT option is in effect, and any diagnostics are to print, these will print starting on a new page following any MAPS or following the /* card if no MAPs were selected. If no diagnostics are produced, the message NO ERRORS IN THIS COMPILATION will be printed on the next line to be printed without starting a new page unnecessarily.

1. .... are printed messages showing errors in source program cards in regard to compilation.

2. .... and .... designations are needed for the delineation of card punching.

* * *

1. Diagnostics
2. Area A, Area B

---

24. The omission of a page line number will not cause a sequence error indication, either when first encountered following cards with page-line numbers, or when last encountered upon beginning to have numbers after a group of cards with none. When a card with a page-line number is subsequently followed by another card with a lower value number, even if cards intervene with no sequence numbers, an error will be indicated on that line only. If any such errors are observed when reading a source deck, a message indicating the presence of sequence errors are printed at the end of the listing, along with a count of how many such error line indicators were printed. The presence of sequence errors does not in any way restrict a compilation of COBOL, and is for warning purposes only.

Answer true or false:

1. The presence of sequence errors in source card listing will cause compilation to end.

* * *

False. Compilation will continue.

---

25. The following are examples using the control cards discussed to this point. List the control cards needed to effect these job streams.

    a. Compile, Build the Core Image, and Execute a program reading a card file.

    b. Load a program stored previously under the name PAYRL, and Execute it.

* * *

a // JOB // COBOL (Source Program goes here, preceded by any compile time option cards) // XEQ (Data deck goes here, preceded by any *FILES cards needed, etc.) /*

b // JOB // XEQ PAYRL (Data deck goes here, preceded by any *FILES cards needed, etc.) /*

---

26. 1130 COBOL uses standard disk data files, as stored using the DUP programs provided with the Monitor. If the programmer provides the proper bridging of data types, formats, array sequencing, etc., then 1130 COBOL may process in a sequential or in a random mode any sequential disk files produced by programs coded using 1130 Assembler Language, FORTRAN, or RPG. 1130 COBOL does not implement directly Indexed Sequential files.

    1130 COBOL Disk Files are identified to the program by means of *FILES cards, just as used in FORTRAN. The *FILES card must follow the //XEQ control card, and precede any data cards to be read. Cc 16-17 in the XEQ card will contain a count of the number of * control cards that must follow.

    Answer true or false:

    1. 1130 COBOL may use standard disk data files produced by programs coded in other languages, except Index Sequential files.

    2. 1130 COBOL may process disk files in random or sequential mode.

* * *

1. True
2. True

---

27. COBOL provides for specification within the source program of the number of records to be processed by a program. Since a single compiled program may be executed successively against a number of files of differing lengths but similar contents, the size specified by the programmer may in such cases be the maximum size number.

If one or more Card Data Files are to be read, a /* in cc 1-2 control card must follow the last data card in each card data file. If a /* card is present but unneeded, it will be ignored.

Why are these statements wrong?

1. The programmer specifies varying record number sizes for each different use.

2. One /* card will suffice for mulltiple card data files.

* * *

1. Maximum record number size is specified, allowing multiple uses.

2. Each card data file must be followed by a /* card.

---

28. Compile time option specifications are normally provided by the programmer, along with the COBOL source program. Three categories of compile time options can be exercised using 1130 COBOL. The sources of these options are:

    Compiler-defined standard assumtions
    Installation-defined standard assumptions
    Compile-time options defined by Compiler Control Cards

Answer true or false:

1. The programmer is bound by compiler defined standard assumptions in regard to compile time options.

2. The programmer may over-rule any type of option assumption.

* * *

1. False. Many compiler defined standard assumptions may be replaced by installation defined standard assumptions.

2. True. This may be done by including compiler control cards selecting the desired options.

---

29. The categories were shown in the previous frame in order of priority. The lowest level is the compiler-defined standard assumptions. If nothing at all is done by the user, these will be in effect for all compilations. If a user installation creates a set of installation defined standard assumptions, these will completely override the compiler defined assumptions. The highest priority, however, are compile time defined options. Inclusion of compiler control cards at compilation time will override on an individual basis any option in either of the other two assumption sets, for that one compilation only.

   1. .... standard assumptions are in effect for all compilations if nothing is done by the user.

   2. .... assumptions or .... may override compiler-defined standard assumptions.

* * *

1. Compiler-defined
2. Installation defined, compiler control cards

---

30. Since standard option assumptions may be either positive (rarely) or negative (normally), most of the options have both a positive form and a negative form. The negative form, when permitted, is always formed by appending the letters NO ahead of the normal 4-character positive form of the option key word.

   The compiler itself, if nothing further is done, always makes a negative assumption whenever possible on all options. Unless the installation defines a local standard to the contrary, all options available at compile time having negative forms must be specifically called for on a positive basis if desired through the use of compiler control cards. The negative form of option key words are thus of use only in cancelling installation defined positive options, when such options are not desired for a specific compilation.

   Match the items in the two columns:

   1. How a negative form of an option is formed

   2. What the compiler assumes

   3. Function of negative form of option

   a. Negative form of options

   b. Formed by putting NO ahead of the option key word

   c. Cancels positive installation defined options

   d. Positive form of options

* * *

1. b
2. a
3. c

---

31. Each installation may individually define local standard assumptions for compile time options to be taken unless overriden by compiler control cards. This capability extends to both defining a standard title to print at the head of each page of source listing (e.g. Name of the Installation, School, Business, etc.), and to defining which of the various options are normally desired to be exercised.

1. Compile time options usually in force are:

    a. installation defined

    b. punched in control cards

2. Compiler control cards:

    a. affect printing only

    b. override the usual options

* * *

1. a
2. b

---

32. An installation defined default option set is communicated to the compiler through a set of constants stored on the system disk. The data defining the options appear on the disk as a "program" in Core Image format, but actually contain only constants defined in a fixed format, and no executable instructions. These constants are initially created by an assembler language program, a copy of which is supplied in card form with the compiler. This program, known as QDFLT (Q-default), as received along with the rest of the original program material, is punched to agree with the compiler standard assumptions. The program provides internal comments, containing all instructions necessary to modify the constants to form the standard options desired by an installation.

1. What is the function of the card program QDFLT?

2. When is QDFLT used?

* * *

1. QDFLT converts the standard assumptions into the options desired by a particular installation.

2. QDFLT is used during compilation.

---

33. When the compiler defined standard assumptions, or installation defined standard assumptions are unsatisfactory for a particular compilation, compiler control cards can be introduced along with the source program itself, to change any options selected on an individual basis for that one compilation only. Compiler control cards precede the source program, coming just after the // COBOL monitor control card that calls the compiler into execution.

All compiler control cards are identified with an * symbol in cc1. Columns 2-72 are normally available for use in specifying options. Columns 73-80 are always available to the user for any valid card codes. Normally, these would be punched with the identification of the program with which they must be used.

Compiler control cards list as they are read, immediately following the printing of the // COBOL card. Editing is accomplished as they are read, to insure that key words are spelled properly, etc. Cards which completely pass edit are listed exactly as punched. Cards which contain any errors are identified on the listing by changing the "*" symbol normally printing from cc 1, to a "-" symbol before printing.

1. .... are used to override options otherwise in force.

2. As these cards are read, an .... is performed to verify the accuracy of punched data.

* * *

1. Compiler control cards
2. Edit

---

34. A source listing title card is identified uniquely by a second * symbol punched into column 2, in addition to the * symbol in cc 1. The title itself is punched into cc 3-58. Columns 59-72 are reserved (should be left blank), and cc 73-80 are not used by the compiler (may be punched by the user with any valid card codes.)

What is the function of a source listing title card?

* * *

The card causes the title of the source program to print at the top of the compilation listing.

---

35. In the standard form of the compiler control cards (all except the source listing title card format), key words may be punched one to a control card, or multiple key words per card can be punched, separated only by commas, and containing no embedded blanks. In the same job stream, some cards may have one key word per card, and be intermixed with other cards with multiple key words. Any number of compiler control cards may be used.

All compiler control card key words in their positive form have a 4-character name. Most key words have a negative form as well, constructed by appending NO in front of the positive form of the key word, forming a composite 6-character name. These will have the form NOxxxx where xxxx is the positive form of the key word to be negated. Negative form key words are required in order to be able to reverse installation defined standard assumptions expressed in a positive form, but which are not desired for this compilation.

Match items in the two columns:

1. Key words a. Reverses positive standard assumptions

2. The word NO b. Branch of a program

c. Coded form of options

* * *

1. c
2. a

----

36. Examples of control card formats:

```
*xxxx       )
 (or)       )   Example of a single key word in a control card.
*NOxxxx     )

*NOxxxx,xxxx,xxxx,NOxxxx     )   Examples where multiple key
*xxxx,xxxx,NOxxxx            )   words are in the same control
                             )   cards.
*NOxxxx,NOxxxx,NOxxxx,xxxx   )   Key words permitted in any
                             )   sequence.
```

The first blank read in a compiler control card terminates any further search for additional key words. As a result, no embedded blanks can be included in a string of key words, and only commas can be used to separate selected options.

Why cannot blanks be embedded in a string of key words?

* * *

The first blank read in a compiler control card terminates the reading of key words.

----

37. Two types of map options are available, providing a listing of the symbolic assignment table of core addresses assigned (relative to zero to which a relocation factor must be added) to the names of data elements in the Data Division, and to each statement number assigned by the compiler in the Procedure Division. These are as follows (standard assumptions are underlined):

DMAP = Print a MAP of the Data Division.
XMAP = Print an extended MAP of the Data Division.
(or)
<u>NODMAP</u> = Omit the Data Division map.

PMAP = Print a MAP of the Procedure Division.
(or)
<u>NOPMAP</u> = Omit the Procedure Division map.

/XXX = Print all MAP addresses relative to a base
(or) address XXX (three hex digits) - normally /1FE.
<u>/000</u> = Print all MAP addresses relative to zero.

If a Base Address Option was not specified at compile time, nor established as an Installation Defined Assumption, then all addresses printed in either MAP output type will be displayed relative to zero as the assumed origin point of the program in core.

Answer true or false:

1. All statement addresses are exact core locations.

2. Data and/or statement addresses comprise maps.

* * *

1. False. Addresses are relative to a base address.
2. True.

---

38. Two types of system device compile time options are provided as alternates in 1130 COBOL. These are specified as follows (standard assumptions are underlined):

2501 = System Input Device is a 2501 when ACCEPT
(or) is coded and no FROM is specified.
<u>1442</u> = System Input Device is a 1442 when ACCEPT
is coded and no FROM is specified.

1403 = System Print Device is a 1403 For trace or
error termination output, and when DISPLAY
(or) is coded and no UPON is specified.
<u>1132</u> = System Print Device is a 1132 for trace or
error termination output, and when DISPLAY
is coded and no UPON is specified.

Note that these two options are alternate choices, and have no negative forms. Whenever possible, it is highly preferable to establish installation standards for these two options, since they are not likely to change.

1. Do system device compile time options have a negative form?

2. Are the 1442 and the 1132 standard assumptions?

* * *

1. No. An alternate choice is used.
2. Yes

SUMMARY

In this lesson you have been introduced to operation of 1130 COBOL as a component of the 1130 Disk Monitor System, Version 2. The lesson also covered the COBOL compile time options. Refer to the 1130 COBOL Operations Manual and the 1130 COBOL Programmer's Guide for complete information.

END OF LESSON 38

LESSON 39

LESSON 39 - COBOL ERROR MESSAGES AND DIAGNOSTIC AIDS

INTRODUCTION

The 1130 COBOL Compiler was designed specifically for use in educational institutions as a tool of instruction in the use of COBOL, as well as for use as a highly efficient compiler of programs to make effective use of the 1130 Computing System as a general purpose computer. As a result of the former, this compiler was designed with a very large number of diagnostic messages intended to assist the novice programmer in producing syntactically correct source programs, and in assisting in understanding the reasons for non-operability of object programs.

Also included in this lesson is a partial list of error codes produced by object programs compiled using 1130 COBOL. In order to minimize core requirements for non-productive code in the object program, 1130 COBOL identifies execution time errors by use of a two digit error code only, associating with this code the location of the procedural statement and the data being processed at the time of the error. This lesson discusses such messages and other diagnostic aids.

This lesson will require approximately three quarters of an hour.

1. Diagnostic messages produced by the compiler have identification numbers, but such numbers are for reference only, since the full text of the error message is always produced along with the number. By this means, in maximum communication with minimum opportunities for error in interpretation is provided to the programmer.

   1. Diagnostic messages assist the programmer to detect errors in both .... and .... programs.

   2. Diagnostic messages produced by the compiler are:

      a. complete

      b. references only

   3. Messages produced during execution are ....

*　　　*　　　*

1. Source, object
2. a
3. References to the messages listed fully in other COBOL documentation.

---

2. The messages supplied along with source diagnostics produced at compile time are of two general types. In one type, the content of the diagnostic message is fixed. By reference to the statement number supplied along with the diagnostic error message, the meaning of the message as it applies to the condition becomes clear. In the other type, the content of the diagnostic message is incomplete as described subsequently, with additional text appropriate to the condition being reported supplied at the time the message is produced on the printer. Variable content of diagnostic messages is always included within a pair of / symbols. It frequently appears in the form /XX...XX/ in the message as shown in this manual. The actual length of the text supplied, however, will vary with the content length, causing such messages to vary in length according to the data reported.

   1. Not all source diagnostic messages are complete in and of themselves.

      a. true

      b. false

*　　　*　　　*

a. Some messages refer to variable length text found in the 1130 COBOL Operations Manual.

---

3. This compiler attempts to provide full diagnostics of all source text in the program, despite the recognition of errors which make it impossible to produce a valid object code. Upon occasion, however, the compiler cannot continue on a given statement since no logic exists to place the remainder of the statement in a proper context. In such cases, the statement provided will state that the compiler cannot continue, and is flushing the rest of the statement identified. If this occurs, the programmer should examine the entire statement not yet analysed for syntax correctness by the compiler, in order to improve the opportunity of correct operation during the next compilation.

   Why will all levels of errors in program text syntax not be found in one compilation run?

* * *

In cases where no logic exists to place the remainder of a statement in proper context, the compiler does not finish analysis of the statement beyond the point of error.

---

4. 1130 COBOL provides for three levels of diagnostics. An E-level error always results in suppression of execution (and of storing the compiled program.) This type of diagnostic indicates an error judged to be severe to the point where an attempted execution would be pointless.

   The mildest type of diagnostic is the W-level, intended to be in the form of a warning. While execution and storing of the compiled module is always permitted, the programmer is provided with a warning for coding which might possibly be in error.

   Match the items in the two columns:

1. E-level error	a. Execution is discontinued
2. W-level error	b. Warning only
	c. Continued execution
	d. Most severe type of error

* * *

1. a,d
2. b,c

---

5. A middle level of diagnostic provided in 1130 COBOL is the C-level error, or conditional acceptable error. Here the situation detected is marginal, and could easily go in either direction. Since a decision must be made however as to what to do when C-level errors are detected, a compile time option is provided to establish a default action which is agreeable to the user. This option is specified as follows (the standard assumption is underlined):

SUPX (or)	=	Suppress execution and cataloging on C-type errors.
NOSUPX	=	Permit execution and storing of the object module on C-type errors, provided E-type errors are not present.

It is possible to establish a standard assumption for the installation as to which option should be normally selected in the absence of a compile time control card specification.

Match the items in the two columns.

1. C-level error	a. Suppress execution
2. SUPX	b. Moderate in severity
	c. Option refers to C-level errors only
	d. Permits execution or discontinuation at user's option.

* * *

1. b,d
2. a,c

---

6. Some examples of compilation error messages follow:

ERRNO	LEVEL	MESSAGE SUPPLIED
004	W	CONTINUATION LINE ALL BLANK. LINE IGNORED.
005	C	FIRST CHARACTER ON CONTINUATION LINE IN AREA A. ACCEPTED AS IF IN AREA B.
006	C	IN A CONTINUED NON-NUMERIC LITERAL, FIRST CHARACTER ON CONTINUATION LINE NOT QUOTE. ACCEPTED AS IF QUOTE PROCEDED.
007	C	UNFINISHED NON-NUMERIC LITERAL NOT CONTINUED. TREATED AS IF CONTINUED.
008	E	NON-NUMERIC LITERAL LONGER THAN 120 CHARACTERS, TRUNCATED TO 120 AND STATEMENTS BYPASSED TO NEXT NON-CONTINUATION LINE.
009	E	NON-NUMERIC LITERAL DEFINED WITH NO CHARACTERS BETWEEN QUOTES. LITERAL IGNORED.

---

7. Compilations for which no diagnostics were produced is no guarantee of the logical correctness of a COBOL program. For example, a misplaced GO TO statement may cause the program to enter an endless loop. Other type of errors not diagnosed in the source program may involve error data not apparent until execution time.

   While not all execution time errors will be identified by the error recognition coding included in object programs produced by 1130 COBOL, many will be caught, forcing error termination at execution time. If the compiler itself is operating properly, no error condition occuring during execution time should cause the computer to terminate due to loss of the program control, except device malfunctions not under control of the program.

   The normal error conditions during execution time merely terminate the program being executed, and proceed to the next job in the job stream. Upon occasion (as selected optionally during compile time) a core dump may be produced at such times.

   1. If a program compiles without diagnostics, it will execute properly.

   2. Device malfunction may cause computer termination.

   3. Most execution time errors cause flushing to a new job.

      a. true

      b. false

* * *

1. b. Errors in logic do not become apparent until execution time.
2. a
3. a

----

8. Execution errors have the following print format:

   STNO=XXXXERR=YY**LOC=ZZZZ**ITEM=WWWW.

   The execution error messages indicate the nature of the error (YY), the location of the error (ZZZZ), and the location of erroneous data (WWWW), if applicable. Optionally the number of the source statement in error will precede the message if selected at compile time. If the statement number option was not chosen, errors print shifted to the left so that **ERR will now appear in print positions 1 to 5. The operator should be aware that in some cases a core dump follows the printing of an error message.

   Match items in the two columns:

1.	YY	a.	Location of erroneous data
2.	ZZZZ	b.	Nature of error
3.	WWWW	c.	Location of error
		d.	Beginning disk location

* * *

1. b
2. c
3. a

---

9. During execution of a program, certain error conditions will cause the program to Wait. The operator may determine the cause of the temporary stop by reading the contents of the Accumulator and Extension which will be displayed in the Console lights.

   The operator can terminate a program by pressing the Interrupt Request Key on the Console Typewriter. By setting the Console Entry Switches 12 to 15 before pressing the Interrupt Request Key, the operator can cause any message between 90 and 9F to print. The installation must define the meaning of messages 91 to 99, 9A, 9B, 9C, 9D, 9E, and 9F.

   1. Certain error conditions cause the program to ....

   2. The cause of the stop is evidenced in the console ....

   3. Pressing the .... on the typewriter terminates the program.

* * *

1. Wait
2. lights
3. Interrupt Request Key

---

10. The following table identifies the console entry switch settings for each of the 16 possible message numbers:

Msgs	Switches	Msgs	Switches
90	12-15 off -	98	12 -
91	15 -	99	12,15 -
92	14 -	9A	12,14 -
93	14,15 -	9B	12,14,15 -
94	13 -	9C	12,13 -
95	13,15 -	9D	12,13,15 -
96	13,14 -	9E	12,13,14 -
97	13,14,15 -	9F	12,13,14,15 -

What is the purpose of the above chart?

* * *

To correlate switch settings with messages 90-99, 9A-9F.

---

11. Some examples of object diagnostic code messages follow:

YY	WWWW	MEANING
01	File number of the checkpoint file.	Checkpoint file defined in working storage. (A checkpoint file must always be pre-defined in space reserved in the User or the Fixed Area on the disk.)
07	Data Address in core storage. See the DMAP.	A data exception was taken due to an invalid data type being recognized for the use intended. Examine the data for validity.

1. The YY code designates ....

2. The WWWW designation shows ....

* * *

1. the nature of the error
2. the location of erroneous data

---

12. Two options provide for supplying assistance to the programmer when problems occur at execution time, even though the compilation was apparently successful.

One option provides for producing a core dump, when an execution run is terminated by the operator. A core dump is also produced when an execution time routine determines that execution cannot continue due to an error either in the program or in the data being processed, making further execution pointless. This option has two positive forms providing two alternative forms of a core dump, plus a single negative form when an installation defined standard specifies a dump is to be overriden. This option is as follows (standard assumption is underlined):

DUMP = Dump all of core following an execution time
(or) error termination.
DDMP = Dump the Data Division core of the mainline
(or) program following an execution time error termination.
NODUMP = Do not provide an automatic core dump at an execution time error termination.

Match the items in the two columns:

1. DUMP a. Dump all core

2. DDMP b. No automatic core dump desired

3. NODUMP c. Procedure Division core dump

d. Data Division core dump

* * *

1. a
2. d
3. b

---

13. During program execution, if one of the above positive options has been selected, a DUMP or DDMP (data dump) can be initiated by the operator. This can be done at any time during the execution of a program by cancelling the job with the usual "Interrupt Request" key depression on the typewriter console of the 1130. Even if the execution problem is so severe that the computer is in a non-processing "wait" state, if core itself has not been overwritten, the operator can initiate the dump with the Interrupt Request Key.

How can the operator initiate a core dump during execution?

* * *

By depressing the Interrupt Request Key.

---

14. Frequently, errors in the data being processed, or unexpected ranges in data, can result in the error termination of a program which may have executed successfully for some time prior to this point. One example of this is when input data provides the subscript for an operation upon an array, and the data value indicates a location in core outside of the range of the array. This condition cannot be checked for during compilation, since the actual values to be used as subscripts cannot be known until input data is read at execution time. When an error termination does occur, a DUMP or DDMP will be produced automatically provided one of the above positive options has been selected.

    1. Errors in the .... being processed can cause an error dump during execution.

    2. Such a core dump cannot occur unless a .... dump option was previously selected.

* * *

1. data or data range
2. positive

----

15. A second execution time option specified at compilation time provides for inclusion of source program statement numbers right in the object code. While additional core is required for the object program when this option is taken, the advantages during the debugging stage of a program are many. For example, when statement numbers are included as a result of this option, all execution time error messages include the source statement number of the statement being executed at the time of the error recognition. This option is specified as follows (standard assumption is underlined):

    STNO = Include statement numbers in with the object code.
    (or)
    NOSTNO = Do not include statement numbers in the object code.

    Name an advantage of having source program statement numbers included in object code. Name a disadvantage.

* * *

Advantage: debugging is easier because source statements involved in execution errors are more easily located.

Disadvantage: additional core is needed for the object program.

----

16. In one situation, the STNO specification is mandatory. This occurs when a READY TRACE has been included in the COBOL source text. The trace provided in COBOL is a transfer trace, printing statement numbers each time that a transfer to the start of a new paragraph occurs. Obviously this cannot be done unless the statement numbers are included in with the object code. At the time the READY TRACE is recognized during compilation, it is too late to automatically turn on the STNO option. At least some addresses will have already been computed without allowing for the inclusion of the statement number in core.

The inclusion of statement numbers in object code is mandatory when ....

* * *

READY TRACE is included in the COBOL source text.

---

17. 1130 COBOL statement numbers are assigned by the compiler, and are printed on the source statement listing. Recall that a statement in 1130 COBOL is defined as that portion of the source text which starts with:

- anything beginning in Area "A" in any Division
- the word SELECT in the Environment Division
- a level number in the Data Division
- the word IF in the Procedure Division
- any verb in the Procedure Division except NOTE and ENTER

Statements terminate just prior to the start of the next statement, or the end of the program.

Match items in the two columns:

1. Assigned by the compiler
2. Point of statement termination
3. Anything beginning in Area "A"

a. Beginning of a source text statement

b. Prior to start of a new statement

c. Level numbers

d. Statement numbers

* * *

1. d
2. b
3. a

---

18. During the execution of a COBOL program which includes statements providing for use of TRACE, TRACE output and error messages may appear, regardless of the settings of the console switches. If the programmer requests a TRACE, the statement number of a paragraph is printed whenever that paragraph is executed. The output format for TRACE is '=nnnn' in print positions 1 to 5 where 'nnnn' is the statement number of a paragraph. The 'STNO' compiler option must also have been in effect when a program using TRACE was originally compiled.

Answer true or false

1. TRACE output and error messages may print during execution regardless of console switch settings if TRACE statements are included in the program.

2. The "STNO" compiler option must have been previously selected.

* * *

1. true
2. true

----

19. Most COBOL programs are easily debugged at the source level, without resorting to the use of the TRACE feature. When it is required however, the amount of information provided by the TRACE can be augmented through judicious use of the DISPLAY capability to print intermediate results at appropriate times during the display.

Since TRACE takes considerable time on the printer, every attempt to reduce TRACE print time should be attempted. For example if a program failed only upon reading and processing of a particular record in a file, insert a routine to recognized the desired record before allowing READY TRACE to be executed, restoring the program to the non-tracing mode by a RESET TRACE before the next record is read.

Match items in the two columns:

1. DISPLAY

2. Selective initiation of READY TRACE

a. Reduces printing time for TRACE output.

b. Can be used if is not properly processed

c. Command which may be used to print intermediate results

* * *

1. c
2. a,b

----

20. To use the TRACE feature, code READY TRACE anywhere in the Procedure Division of a program. After the READY TRACE is executed, the compiler-generated statement numbers shown at the left of source listings will begin to print on the printer, one statement per line, intermixed with other line printer output produced by the program. This will continue until a RESET TRACE statement coded into the same program is subsequently executed.

    1. What command in the program causes a TRACE to occur during execution?

    2. What command causes the TRACE to stop?

    3. What will be printed?

* * *

1. READY TRACE: starts TRACE
2. RESET TRACE: stops TRACE
3. the number of each statement executed, plus all normal output directed to the systems printer device

---

21. At times it is convenient to code the TRACE features, including applicable DISPLAYs, and still not have them occur at execution time except as desired, without recompilation. By the use of the sense switch option, execution of the READY TRACE can be conditioned to occur only when a particular sense switch has been raised. For example, it would be possible to program TRACE so that it did not occur, unless, like the TRACE in FORTRAN, sense switch 15 were raised.

    How can TRACE be selected upon operator initiative?

* * *

By use of a sense switch option.

---

22. During execution of a COBOL program, punched card output may be produced. COBOL cannot read data from a card and punch information back into the same card. COBOL must always punch into cards which are entirely blank when placed in the punch device. If the punch device is a 1442 model 6 or 7, each card will be automatically read first to insure that it is truly entirely blank before punching.

    1. COBOL cannot read data from a card and .... data into the same card.

    2. .... cards must be used where punching is required.

* * *

1. punch
2. blank

---

23. Compile time stops resulting from problems which must be resolved by the programmer and presented only on the line printer, since they do not involve the operator. Samples of these messages numbered between Q01-Q09 will be found in the section of this course on Abnormal Compilation Terminations.

Compile time stops requiring action by the operator are presented either on both the line printer and the console typewriter, or in the lights of the console using the accumulator and its extension. Lights are used when routine operations are involved, such as requiring a device to be placed in a ready status. The typewriter is used for non-routine operations where, in addition to a message number, a full text message describing the event requiring action is always displayed. Messages numbered between Q10-Q29 fall in this category.

1. Compile time stops requiring action by the operator are presented on the ...., ..., or in the ....

2. The .... are used for routine operations and the .... for non-routine operations.

* * *

1. console typewriter, line printer, console lights
2. lights, typewriter

---

24. At times the compiler itself may fail to function according to specifications being checked during operation. Messages numbered between Q30-Q99 are presented at such times, with a common text as follows:

```
Qnn INTERNAL COMPILER ERROR WHILE PROCESSING STATEMENT nnnn.
    COMPILER REQUIRES MAINTENANCE.
```

When a message in this category is received, the operator should immediately refer the situation to local management. This situation could result from not using the latest compiler release. If local personnel responsible for the 1130 installation determine that the latest release is in fact installed, the appropriate assistance must be obtained to bypass or circumvent the problem, and report the situation to those responsible for central maintenance of the compiler.

Answer true of false:

1. The operator should handle a message numbered Q30-Q99 himself.

2. A message in the Q30-Q99 range could occur because the latest compiler release is not being used.

* * *

1. False. He should notify local management.
2. True

---

25. Unexplained stops may sometimes occur. Whenever possible, an attempt should be made to explain these in terms of the physical effects observed, and refer these to a programmer for determination as to the cause. When these occur, the Interrupt Request Key on the console may be depressed to go to the next job.

If an unexplained error occurs, the .... should be consulted. The .... may be depressed to skip to the next job.

*   *   *

programmer
Interrupt Request Key

---

26. The operator should be aware that many errors encountered in the compilation of a program may result in an abnormal compilation termination. Regardless of the cause, reasons for abnormal compilation termination are always provided on the main printer, even though certain ones also appear on the console typewriter. No stop is involved if possible. By this means, full communication is provided to the programmer, while maintaining maximum efficiency of operations.

Abnormal compilation termination causes a message of the following format to appear:

Qnn COMPILATION DISCONTINUED BECAUSE**** reason for termination.

1. When does the message "Qnn COMPILATION DISCONTINUED BECAUSE**** reason for termination print?

2. Where may such a message print?

*   *   *

1. When an abnormal compilation termination occurs.
2. On the main printer always; on the typewriter sometimes.

---

27. Stops occurring during execution of a program written in COBOL will generally result in identifying codes being displayed in lights on the console using the accumulator and its extension. A few conditions are displayed on the console typewriter.

Two types of temporary stops are produced during the execution time of a COBOL program. Each type is immediately recognizable by reason of the contents of the extension which either has "all lights on", or not. Whenever a stop occurs as a direct result of a programmed request for a stop, all lights will be on in the extension, to in effect 'underline' the contents of the accumulator itself, in which a unique code assigned by the programmer identifies the reason for the stop. All stops not intentionally caused by the program itself will will never cause all sixteen lights to be on in the extension.

1. Whenever a stop occurs because of a programmed request, ... lights of the Accumulator Extension will be on.

2. When all lights are on subsequent to a programmer stop, a .... identifies the stop condition.

* * *

1. all
2. unique code displayed in the Accumulator

28.

EXECUTION TIME PROGRAMMED STOPS (Extension is /FFFF)

Accumulator	Condition and Action Required
<0000	Request for keyboard entry by the operator. Keyboard select light will be on. Operator must key in appropriate message.
<FFFF	A STOP followed by an 'alpha literal' displayed on the console typewriter has been executed by the program. Operator must perform actions requested (if any), and press Program Start to continue, or Interrupt Request on console to abort the job.
Other	A STOP followed by a 'numeric literal' displayed in the accumulator has been executed by the program. Operator must perform actions (if any) required by the unique program operating instructions prepared by the programmer, or by established installation conventions, and press Program Start to continue, or Interrupt Request on console to abort the job.

1. A STOP may be followed by an .... or a .... literal.

2. Literals may appear on the .... or in the ....

* * *

1. alpha, numeric
2. typewriter, accumulator lights

29.

EXAMPLES OF EXECUTION TIME OTHER STOPS (Extension is not /FFFF)

Accumulator	Condition and Action Required
/1000	1442 reader requires attention. (1) Press reader START to make ready. (2) Press PROGRAM START on console.
/100B	1442 PUNCH REQUIRES ATTENTION. (1) NPRO run out all cards. (2) Next to last card in stacker was not blank as required for punching. (3) Discard any non-blank cards, and reload hopper with all-blank cards. (4) Press START on the 1442. (5) Press PROGRAM START on console.

Some execution time stops are caused when an .... requires attention.

* * *

Input/Output unit

---

30. Most execution time stops identified by a display on the typewriter are produced by the Disk Monitor System, brought into effect by the // XEQ control card used to execute a program. One stop condition however, not handled by the monitor, is handled through code supplied by the COBOL compiler, and included in the object code. This condition occurs when a disk file spans across two packs, only one of which is mounted at the moment. While multi-cartridge files are permitted by COBOL, the monitor as such does not provide direct support, thus requiring control to be provided by the COBOL compiler.

.... files are permitted by COBOL but control must be supplied by the ....

* * *

multi-cartridge, COBOL compiler

SUMMARY

In this lesson you have been introduced to some of the diagnostic messages and other aids. In addition, some of the codes have been provided for recognition of conditions causing a termination of processing. Refer to the 1130 COBOL Operations Manual and the 1130 COBOL Programmer's Guide for complete information.

END OF LESSON 39

THIS PAGE INTENTIONALLY LEFT BLANK

LESSON 40

LESSON 40 - 1130 COBOL COMPILER EXTENSIONS

INTRODUCTION

COBOL is a "standardized" language, with responsibility for cross-industry standards assigned by all implementors of COBOL to the American National Standards Institute. Yet each implementor of a COBOL compiler is free to "extend" the language in a manner not in conflict with previously adopted standards. Eventually some of these extensions may become part of the standard language due to customer acceptance of the features provided.

1130 COBOL implements a number of "IBM Extensions" to the language and additionally provides a number of utility type programs and sub-programs designed to make easy the installation and use of this compiler. The 1130 computing system is a unique piece of hardware, which by its nature makes certain extensions easy, while others are more difficult. No attempt has been made to provide a true library of utility type programs and/or sub-programs. Instead only those minimum necessary and highly generalized routines have been included which are likely to prove to be of the greatest benefit to the user of this compiler.

Some of the extensions and utility type programs included with this compiler are described within this lesson.

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

COPY statement
QLIBR library routine
CALL statement
USING option of the CALL statement

This lesson will require approximately one hour

1. The 1130 COBOL compiler contains the capability to copy a source module from a source language library into a COBOL program being compiled. Since other than through the use of COPY all COBOL source statements must be read by the card reader on the 1130 system, a total COBOL source program may be composed of a combination of uniquely coded source statements punched into card form, plus standard library source routines incorporated into a program at the time of compilation from a source language library. A routine incorporated via the COPY verb will be incorporated unchanged from the form in which it was originally stored. No REPLACING capability is included in the 1130 COBOL subset.

   A COBOL source program may consist of .... and ....

* * *

Individually coded source statements
Standard library source routines.

---

2. On the 1130 computing system, the Disk Monitor incorporates both data and program content into a single disk library system. No source language library as such is provided by the Monitor system. The user may establish in the User Area of his System Cartridge, a Source Library for the COBOL Compiler. This library appears to the Monitor as an ordinary data file whose name is QLIB. During compilation, source modules may be incorporated into a program by means of a COPY statement.

   The Disk Monitor furnishes the source language library. True or false?

* * *

False. The Source Library is added to the main program at compilation time through use of the COPY statement.

---

3. A program to maintain the Source Library is provided as part of the COBOL program product. Named QLIBR, it is included among the library routines distributed with 1130 COBOL. This main line program provides for allocation of a special directory within the stored area, and for the storage of source language modules in a compressed manner in the remainder of the space. Each module will be cataloged into the directory at the time it is stored originally, so that it may be subsequently located.

   What is the function of the library routine QLIBR?

* * *

QLIBR maintains the Source Library.

---

4. The method of reserving space for the Source Library is very simple. These monitor cards fulfill the function:

```
//JOB
//DUP
*STOREDATA  WS   UA  QSLIB XXXX
 (Sectors for the size of Source Library)
// XEQ    QLIBR
*INITIALIZE XXX(size of directory in sectors)
/*
```

Space for the Source Library is reserved by use of a few ....

* * *

Monitor Control Cards

---

5. The QLIBR routine is a service program provided with the COBOL compiler that accomplishes the following:

a) Adds a source module to QSLIB library.
b) Deletes a source module from QSLIB library.
c) Replaces a source module in QSLIB library (deletes old and adds new).
d) Lists the directory or a source module.
e) Punches a source module into cards.
f) Compresses the non-directory portion.
g) Separately compresses the directory portion.
h) Initializes the library.

The Source Library is maintained by the library routine called ....

* * *

QLIBR

---

6. If a programmer wishes to incorporate a source module from QSLIB into a program, he must use the COPY statement as in the following example:

```
// JOB
// COBOL
*(COBOL Control Cards)
┌──────────────────────────┐   ──┐
| IDENTIFICATION DIVISION. |     |
|         .                |     |
|         .                |     |
|         .                |     |    Source module added
| FD SOME-FILE ...         |     |    by use of the  COPY
| 01 SOME-REC COPY ...     |     |    statement
|         .                |     |
|         .                |     |
|         .                |     |
└──────────────────────────┘   ──┘
/*
```

Judging from the example above, what are some kinds of source module which might be COPYed to advantage?

*   *   *

Data or file description routines, which are somewhat repetitious or catalog-like in nature.

---

7. Eight function are provided by QLIBR in support of the COBCL Source Language Library. These are summarized as follows:

*INITIALIZE XXX	(XXX is the number of sectors to be allocated for the QSLIB Directory.)
*ADD name	Adds a source module to QSLIB.
*DELETE name	Deletes a source module from QSLIB.
*REPLACE name	Replaces a source module in QSLIB.
*COMPRESS	Reorganizes and compresses the library.
*LIST	Lists the directory.
*LIST name	Lists a source module from QSLIB.
*PUNCH name	Punches a source module from QSLIB into cards.

The main function of QLIBR is to ....

*   *   *

Maintain the Source Library

---

8. The Source Librarian Control Cards are punched as follows:

```
*function (module name (options)
```

For example: *ADD PROG1 (PROG1 is the user created source module name.)

The above control cards are placed immediately after the // XEQ QLIBR card. The job stream to add a routine in QSLIB is as follows:

```
// JOB
// XEQ QLIBR
*ADD module-name
(Source Cards)
/*
```

The /* card is only needed at the end of all input to QLIBR. The next * control card will identify the end of each source deck that is followed by a new function.

Source Library control cards are part of the .... at .... time.

* * *

Job stream, execution of QLIBR

---

9. You will recall that 1130 COBOL implements the CALL statement as an IBM extension to the American National Standard COBOL language. This feature permits a program written using COBOL to cause the inclusion of an independently written and compiled subprogram along with the main line program during the core load build operation. By this means, on an 1130 computing system a main line program may also cause execution of separately compiled DSF format object modules. These may be generated using 1130 assembly language, or as a subroutine written in COBOL or FORTRAN.

All subprograms brought to execution by the CALL statement must be written in COBOL. True or False?

* * *

False. The subroutines are compiled separately. They may be written in Assembler, FORTRAN, or COBOL.

---

10. The COPY statement causes source routines to become part of a program being compiled. The subprogram called by a CALL statement is not actually part of the main program in that it is compiled separately and always remains physically separate in the Source Library.

What do the COPY and CALL statements have in common?

* * *

They both cause segments of programs written separately to function as part of the main program at execution time.

---

11. A subprogram CALLed by a main line program or by another subprogram may have up to fifteen parameters in the calling sequence. Each parameter must be the name of a data item that can be resolved by the compiler into an address in core, or the name of a file which has been identically defined in both the main line program, and in the subroutine. It must not be a condition name, a special name, a procedure name, the name of a device, switch status, etc. Each subprogram has its own defined number of parameters associated with the subprogram, and exactly this number of parameters must be accounted for when calling the sub-program. No edit of this number can be performed by the compiler, and failure to conform to the precise requirements of a parameter list when calling a subprogram may be obvious only during execution time.

Match each item with the appropriate factor:

1. Limit of parameters in calling sequence
2. Permissible parameter names
3. Control of subprogram parameters

a. Data item
b. 15
c. File
d. Programmer
e. 8
f. Condition name
g. Compiler edit function

* * *

1. b
2. a,c
3. d

---

12. A file of any type (disk or unit-record) may be defined within a COBOL language subroutine compiled as a subprogram using the option provided with the 1130 COBOL compiler. All input-output operations are permitted, via READS and WRITES within the subprogram, as in a main line program. A single restriction applies to all file usage within a subprogram. This restriction requires that a file to be used within a subprogram must be defined together with its record definition both in the subroutine, and in the main line program, with these two names included in the parameter list. This restriction is not required for S/360 ANSI COBOL.

There are no restrictions in file usage within subprograms compiled with the 1130 COBOL compiler option. True of False?

* * *

False. Both file and records must be defined in subroutine as well as main program.

---

13. It is strongly recommended that file and record definitions be cataloged in the COBOL source language library, and COPYed into a program when required. Since few files are defined and used only once, this procedure insures that all uses of a file are defined identically. This technique is especially important for use with files to be used within subprograms. Since all files MUST be defined absolutely identical in both the main line program and in the subroutine, the use of COPY is especially recommended in this instance.

The chief advantage of cataloging file and record definitions in the source library is ....

* * *

The assurance that files and records will always be defined indentically

---

14. Due to an 1130 COBOL requirement, both disk files and unit record files must be defined in the main line program if they are to occur in a subroutine.

COBOL language subroutines used by an COBOL program must be written when using any type of files, with the name of each file and its record area named in the parameter list of the subroutine. Information from these specifications will be used in the compilation. During execution, the file definition and the record area of each file compiled into the main line program will be used for any file I/O operations coded in the subprogram.

Answer true or false:

1. Both disk files and unit record files must be defined in the main line program if they are to occur in a subroutine.

2. The 1130 Disk Monitor automatically assigns file and record storage areas in a subroutine.

* * *

1. True
2. False: the programmer must relate subroutine and main line program file and record references through the use of parameters.

---

15. Since only subprograms may be overlayed in core (via LOCALs) during execution, the result of the above requirements is that file definitions and I/O areas may never be overlayed. I/O areas, however, may be common for more than one file, by means of the SAME AREA clause specified in the main-line routine for unit record files, or by omitting the -x specification for disk files. The actual core generated in defining a file requires but 15 words of core. It can be seen therefore, that no great penalty is paid for this requirement.

1. File definitions and I/O areas may never be .... in core.

2. I/O areas may be .... for more than one file.

* * *

1. overlayed
2. common

---

16. Some non-compatibility with COBOL as implemented on S/360 may result from the duplication of file and record definitions within both a subprogram and its main line program. On S/360, a file may be defined only within a subprogram if desired, or it may be defined both within the main line and within the subprogram, as long as both files are not in an OPEN status at the same time (since both files would have been assigned to the same I/O hardware unit). For these reasons, to insure that upwards compatability to S/360 COBOL is maintained as completely as possible, the following procedures are required both in 1130 and in S/360 COBOL:

    1. In the main line program, CLOSE a file before CALLing a subprogram which will use the same file. After returning again to the main line routine, OPEN the file again if the file must be used once more.

    2. In the subprogram, OPEN, process, and CLOSE a file before returning to the main line.

    In S/360 and 1130 COBOL, when file and record definitions are duplicated, the file may not be .... in the main line program and subroutine at the same time.

* * *

OPEN

---

17. In 1130 COBOL, only a single File Definition System Table and record area is actually in core for a given file, even though defined both in the main line and in the subroutine. For this reason, it is required in 1130 COBOL only that the file name and record name be in the argument list of the CALL and in the parameter list of the subroutine, following the PROCEDURE DIVISION USING... The inclusion of these two elements represent the only true incompatibility between 1130 and its S/360 counterpart. To simplify conversion of programs to S/360, it is recommended that these parameters be punched into a separate card, for easy removal if compilation using S/360 COBOL is desired.

    Match each COBOL-related item with the proper associated fact:

    1. File name and record name, in CALL statement of 1130 COBOL
    2. File name and record name, in subroutine USING ... of 1130 COBOL
    3. File name in CALL or subroutine of S/360 COBOL
    4. Record name in CALL or subroutine of S/360 COBOL

    a. Not a valid COBOL usage
    b. Must be in argument list
    c. Must be in parameter list
    d. May be present if defined in subroutine LINKAGE SECTION and not in FILE SECTION

* * *

1. b
2. c
3. a
4. d

---

18. A subroutine written in COBOL may have no parameters required, or may specify the use of up to fifteen parameters. These appear if present in the USING clause of the PROCEDURES DIVISION header. Their sequence in that clause must match the sequence of arguments in the CALL statement that invokes the subroutine.

Each parameter must be further defined within the subroutine as one of the following:

- A level 77 item in the LINKAGE SECTION
- A level 01 item in the LINKAGE SECTION
- An FD in the FILE SECTION
- A record name (Level 01) in the FILE SECTION

1. Parameter .... must be the same in the USING clause and CALL statement arguments.

2. Each parameter must be .... within the subroutine.

*　　　*　　　*

1. sequence
2. further defined

---

19. The order in which these appear need bear no relation to their order in the USING clause, but:

- Every parameter must be defined in one of the above four ways.

- Every FD in the FILE SECTION and every Level 01 in the FILE or LINKAGE SECTION must appear as a parameter in the USING clause.

Any items defined under a level 01 in the FILE or LINKAGE SECTION should be identically defined in terms of their structure in both the CALLing program and in the subroutine, and these may be referenced in the subroutine even though only the level 01 name is specified in the parameter list in the USING clause. Parameter names defined in a subroutine need not be identical with names of the matching arguments in the CALLing program, but the attributes of each parameter and its associated argument must be identical.

Which statement is false?

1. Parameter names defined in a subroutine must be identical with names of the matching arguments in the CALL program.

2. The attributes of each parameter and its associated argument must be identical.

*　　　*　　　*

Statement #1 is false.

---

20. In the COBOL subroutine, the Linkage Section must be the last in the Data Division, as in the following example:

```
LINKAGE SECTION.

77  NAME1   PIC (etc)

01  NAME2.
    05  NAME3   PIC   (etc)
    05  NAME4   PIC   (etc)

01  NAME5.
         (etc)
```

Structure of a typical LINKAGE SECTION.

The LINKAGE SECTION must be .... in the Data Division of a COBOL subroutine.

*   *   *

Last

---

21. The following is an example of the calling sequence in a program calling a subprogram: CALL "NAME" USING A B C D (up to a maximum of 15 arguments)

If a structure is to be referenced, the highest level to be used by the subprogram must be passed in the argument list, and the level of the name passed and that by the matching subprogram parameter will be assumed to be equal.

If in the example shown in frame 20 both NAME3 and NAME4 are needed by the subroutine, what single name could appear as a parameter to obtain this result?

*   *   *

NAME2

---

22. If data descriptive code has been cataloged into the COBOL source language library, these may be COPYd into both the subroutine and into the main line program, to assure identical descriptions, levels, etc., and simplify subsequent changes to the data descriptions.

    The Procedure Division header must be of the following form if any parameters are to be passed:

    PROCEDURE DIVISION USING A B C D (up to a maximum of 15 parameters)

    Each name following the USING must be specified in the FILE or LINKAGE SECTION. The argument list between the subprogram and the calling program must be the same size. While the same names need not be used, good practice would require that precisely the same names be used as arguments in the calling program and as parameters in the called program for non-generalized subroutines.

    If a subroutine does not require any parameters, then the LINKAGE SECTION in the Data Division must be omitted, and only PROCEDURE DIVISION specified at the start of the Procedure Division.

    Write a Procedure Division header statement showing parameters A, C, and E in use in both subroutine and CALLing program.

* * *

PROCEDURE DIVISION USING A C E

---

23. COBOL subroutines must be set up for compilation as a separate job. The *SUBR control card must precede the COBOL deck. After the compilation the subprogram must be stored into the User Area (UA).

    Example:

```
// JOB T
// COBOL
*SUBR
 (COBOL deck)
/*
// DUP
*STORE       WS  UA  SUBRT
// COBOL
   (Main COBOL Program)
/*
// XEQ
```

    Answer these questions pertaining to the above example:

    1. What is the name of the subprogram to be stored in the User Area?

    2. Will the compiled program be executed?

* * *

1. SUBRT
2. Yes: the // XEQ card triggers execution

---

24. WRITING AN ASSEMBLER LANGUAGE SUBROUTINE TO BE USED WITH COBOL

(Optional section for student follows)

Subroutine linkage generated in a COBOL language subprogram is exactly the same as that generated in FORTRAN. The equivalent assembler language instruction generated by the COBOL compiler when a CALL is encountered is as follows:

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

                                        CALL XXXXX
```

where XXXXX is the name of the subprogram entry point which will be resolved by the Core Load Builder to an address in core.

1. .... and COBOL subroutine linkages are the same

2. The assembler language linkage instruction is CALL XXXXX, wherein XXXXX is the name of ....

* * *

1. FORTRAN
2. The subprogram entry point

---

25. Following the generation of the CALL, DC's will be generated for each argument coded in the CALL in the source program as follows:

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

                                         DC       XXXXX    (where XXXXX=addr of 1st arg)
                                         . .      . . .
                                         . .      . . .
                                         . .      . . .
                                         DC       XXXXX    (where XXXXX=addr of Nth arg)
```

with a maximum of 15 DC's generated when the maximum of 15 arguments are coded in the CALL statement.

Name two assembler language-connected items generated in COBOL subroutine linkage procedures.

* * *

1. The CALL XXXXX statement
2. A DC for each CALL argument

---

26. A subroutine must in turn be coded with the proper linkage to operate when called by the equivalent of above assembler language coding. In assembler language, this will appear as follows:

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

                                      ENTRY DC    *-* (the address of the CALLing
                                                      argument list will be stored
                                                      here when entering the sub-
                                                      program)
```

Here follows the assembler language subroutine coding beginning with the first instruciton to be executed at entry to the subprogram.

Match each argument with its corresponding factor:

1. Assembler subroutine
2. ENTRY DC *-*

a. Contains address of the CALLing argument list

b. Requires operable linkage

c. COBOL instruction

d. Follows ENTRY DC *-*

* * *

1. b,d
2. a

SUMMARY

In this lesson you learned that you can avoid repetitious subroutine writing by using the COPY and CALL statements. You also found that reliance on library routines allows you to assume compatibility with other programmers making use of the same formats or routines, as in data definition, for example. The 1130 COBOL language offers such features as these to facilitate installation in a multi-job environment.

END OF LESSON 40

LESSON 41

LESSON 41 - CALLABLE SUBPROGRAMS

INTRODUCTION

A number of highly generalized CALLable subroutines are distributed together with the 1130 COBOL compiler. Each of these subprograms may be CALLed in a COBOL program. Coding takes the form:

```
       CALL    "XXXXX"     USING        ARG01 ARG02 ARG03 etc, up to 15
                                      arguments.
```

Field names provided in argument lists coded in a main line routine are known by size and data types only to the main line routine, and not to the subprogram. For this reason, field lengths are always included as one of the parameters when coding generalized subroutines to be used with data varying in size. In the CALLable subprograms provided with the compiler, where required, field lengths have been provided for in argument lists. These subprograms will always operate upon the precise number of positions provided at execution time, regardless of whether these positions are correct for the data being operated upon or not. Care must be taken, therefore, that this count is correctly initialized, else sending fields may be unexpectedly truncated, or receiving fields left with prior contents still remaining. Worse yet, non-associated fields located in contigous core, may be overrun, and contents destroyed.

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

QDUMP   CALLable subroutine (as are all others listings below)
QLINK
QCV12
QCV21
QCV13
QCV31
QCVBP
QCVPB
QCVBC
QCVCB
QTEST
QCBRn
QCBPC
QCTBL
QCORE: introduction only
QFIND: introduction only

This lesson will require approximately three quarters of an hour.

1.

"QDUMP" USING Name1 Name2 — Where Name1 is the name of a data element or procedure whose address in core designates from where a core dump is desired upon execution, and Name2 is the name of another data element or procedure whose address specifies where the dump may terminate.

QDUMP

Provides a core dump during execution of main storage between the bounds specified by its two arguments.

This subprogram permits the programmer to examine a limited segment of coding, thus enabling him to concentrate on trouble spots. Write a statement CALLing a subprogram intended to cause a core dump of data element A, when data element B follows next.

* * *

CALL "QDUMP" USING A B

2.

"QLINK" USING Name — Where Name is the name of a field which contains the 5-character name in Alpha-Numeric DISPLAY format of a program to be brought into core in substitution for the present program.

QLINK

Converts the 5-character argument specified (program name) to namecode, and links via the Resident Monitor, to the program specified by the name. The contents of the field may be read into core during execution of the calling program, and thus be variable between executions of the main line program. The QLINK is an alternate exit from the program, and will cause a new main line program to come into core, completely replacing that previously present.

Write a CALL statement which will cause main line program execution to switch to a program named EXIT2.

* * *

CALL "QLINK" USING EXIT2.

---

3. The following four subroutines are character code conversion routines, converting variable-length fields between COBOL DISPLAY formats and another format:

QCV12

Converts from COBOL DISPLAY to two-EBCDIC-character-per-word form.

QCV21

Converts from two-EBCDIC-charact er-per-word form to COBOL DISPLAY form.

QCV13

Converts (40-character set) from COBOL DISPLAY form to three-character-per-word form. A conversion table -- QCTBL -- is used.

QCV31

Converts from three-character-per-word form to COBOL DISPLAY form. A conversion table -- QCTBL -- is used.

What are the two advantages of converting data in the manner described above?

*   *   *

Storage space is saved on the disk through compression.
Data restored for use in program executions.

4.

```
"QCV21"  USING   Expanded-field Compressed-field Name-of-field-
and                containing-numbers-of-characters-to-convert.
"QCV12"
```

This subprogram converts between character fields in COBOL display format and character fields with characters packed two characters per 1130 word of core. QCV12 converts COBOL display formatted data to two-characters-per-word format, and QCV21 provides the reverse conversion. The third parameter is the name of a field defined in the Data Division as a 1 to 4 digit computational field containing the number of characters to be converted. The value stored here should probably be initialized to the proper value at compilation time, but any value could be moved into the field prior to execution of the CALL.

Write a CALL statement which will activate a subprogram intended to convert a field called A to a more condensed field, B. Twelve characters are involved.

*   *   *

CALL "QCV12" USING A B C (C contains the value 12).

5.

```
"QCV31"  USING    Expanded-field Compressed-field Name-of-field-
and                 containing-numbers-of-characters-to-convert.
"QCV13"
```

QCV31

Converts from 3-character/word format to COBOL DISPLAY format.

QCV13

Converts from COBOL DISPLAY format to 3-character/word format.

Conversion is accomplished by using a table supplied with 1130 COBOL called QCTBL. The character set and the particular coding in this table as distributed with the compiler contains the characters "blank", "period", "comma", "hyphen", the 26 letters and the ten digits arranged in collating sequence. The generated codes will thus also be in collating sequence. By changing the values in this table (fully commented in the source program) the user can choose a different character set and/or different encoding for these characters, including the blank.

If a character is presented to the routine other than one of those present in the conversion table, a code zero is produced in the converted text, which normally converts to a blank.

In what way could the subprogram described above be used to compress or expand multiple fields on a single execution of the subprogram?

*   *   *

The user can specify the total length in characters of a number of different fields located in adjacent core positions.

6.

```
"QCVPB"  USING  Binary-field Packed-Decimal-field Name-of-field-
and             containing-numbers-of-decimal-digits-to-convert.
"QCVBP"
```

QCVBP

Converts COBOL binary item (1, 2, or 4 words, depending on number of digits) to Packed Decimal form (4 digits per word, except that rightmost word contains sign in bits 12-15).

QCVPB

Converts Packed Decimal field to COBOL binary item.

By the use of this subroutine, numerical data stored in sequential files produced by an RPG program can be converted to computational form for processing by COBOL, or sequential files can be created containing numerical data capable of being processed by an RPG program. Note that "Indexed Sequential" files created by RPG cannot be directly processed by 1130 COBOL.

When defining working storage or record area fields which will contain data in packed decimal format, data types should be defines as alphanumeric DISPLAY, since this is the most forgiving format in terms of permissible data types.

Sequential files produced by RPG programs can be used during execution of 1130 COBOL program.

a. True

b. False

* * *

a With the help of the QCVPB subprogram when necessary.

7.

```
"QCVBC"  USING    Character-string-field Bit-string-field
and                 Name-of-field-containing-bit-string-length
"QCVCB"             Name-of-field-containing-starting-bit-
                    position-in-bit-string
```

QCVBC

Converts a bit string to a variable length character string, with each on-bit becoming a "1", and each off-bit becoming a "0" in the character string.

QCVCB

Converts a variable length character string, with each "1" becoming an on-bit and each "0" becoming an off-bit.

Converts between character strings exclusively containing 1's and 0's and "bit-strings" (bits contained within 1130 hardware words of core memory). By the use of this subroutine, together with QTEST (described later) to test the content of bit-strings, non-numeric/non-character type data can be processed on the 1130 system, using COBOL as the nominal processing language.

Match each item with the correct factor:

1. Reason for converting bit string to 1's and 0's

2. Reason for converting 1's and 0's to bit string

3. Function of QTEST

a. Tests content of bit-strings

b. Makes data ready for 1130 COBOL processing

c. Conserves storage space

* * *

1. b
2. c
3. a

---

8. Examples of a use for this facility is as follows:

Pattern-type data read from mark-sensed cards using all potentially possible punching positions, and read into the computer by use of the appropriate column-binary read program (QCBR1 or QCBR2).

Analysis of test score responses read into the computer by a column-binary read program from mark-sensed cards or from input produced on an IBM optical mark reader.

Processing of a large sized array of logical switches, densely packed in bit form for data compression purposes, rather than carried as 1's or 0's in character form.

Bit-pattern data can be read into the 1130 via .... or ....

* * *

Mark-sensed cards
Optical mark reader input

9.

"QTEST" USING (Nine parameters required -- See Programmer's Guide)

QTEST

Provides the facility for testing a bit string against a bit mask. Permits branching to separate routine when:

1. The bit string and the mask are identical

2. When all bits in the tested string corresponding to the on-bits in the mask are themselves on

3. When some but not all bits in the tested string corresponding to on-bits in the mask are on

4. When none of the bits in the tested string corresponding to on-bits in the mask are on.

This subprogram provides two distinct types of comparisons at the bit level, which together with QCVBC and QCVCB provide 1130 COBOL with an extremely powerful bit manipulation and examination capability equivalent to assembler language.

How might a mask logically be used in conjunction with test scoring bit string input?

* * *

To test a valid pattern of test answers e.g., answers on a mark-sensed card.

---

10. The first type of bit comparison for QTEST is in effect a character compare, but performed at the bit level instead of at the character level. All bits included within the two strings to participate in the comparison are compared to determine if they are precisely equal (including both the 1's and the 0's in the strings). This test is performed first, and if an exact equal comparison is obtained, a transfer to the routine named as the 6th parameter will occur.

The second type of comparison in effect treates the controlled field as a mask, with only bit positions set to "1" in the mask participating in the comparisons. Any bit position for which a "0" is found in the controlled field will be ignored in the test. Three types of tests are made, after which a transfer will occur if the test is successful. These are as follows:

1. "ALL" -- Each of the 1 bits found in the controlled field is precisely matched by a 1 bit in the tested field.

2. "MIXED" -- Some but not all of the 1 bits found in the controlled field are matched by a 1 bit in the tested field.

3. "NONE" -- There is no 1 bit found in the controlled field which is matched by a 1 in the tested field.

The configuration of bits in the tested field to be checked by QTEST could be changed by altering the .... used in QTEST.

*   *   *

Mask or controlled field bit contents

---

11. If any impossible specifications are present during the execution of QTEST then the program will fall through to the next sequential instruction following the test. For example, if the length of the field to be tested was zero, then the instruction becomes effectively a no-operation and an error return to the next sequential instruction will occur. As a result, this CALL should be followed by an exception routine to be taken if the bit string length can ever be zero.

The QTEST .... statement should be followed by an .... to cover the possibility that a length zero is specified in the routine.

*   *   *

CALL
Exception routine

12.

```
"QCBRn"  USING    ARG01 ARG02 ARG03

n = 1 if the card read device is assigned to 1442
    2 if the card read device is assigned to 2501
```

ARG01 = Name of reader file defined in the program. Must have alternate area.

ARG02 = Name of a work area defined in Working Storage of equal length to the reader input area assigned to the file named ARG01.

ARG03 = A second work area identical in size to that named in ARG02.

QCBR1

Permits the subsequent reading of column binary cards on a 1442 on the next READ command. Punching rows 12-3 and 4-9 of each column are stored as bits 10-15 of two words of storage. Validity checking is suppressed and /* and // cards are not recognized as delimiting cards. Used in conjunction with RD14Q or RP14Q. Normal EBCDIC representation is also stored in the standard Input File Data area.

QCBR2

Permits the subsequent reading of column binary cards on a 2501 on the next READ command. Same facilities and method of operation as in QCBR1. Used in conjunction with RD25Q.

Match each item with the correct factor:

1. Recognition suppression of Monitor Control Cards	a. Reader File Standard Input area also filled
2. Suppression of validity checking	b. Two 6-bit patterns in each card column
3. Separation of card rows, 12-3, 4-9	c. Standard punch-code not applicable
4. Mixed Binary-EBCDIC data in the same card	d. Any punch combination accepted

* * *

1. d
2. c
3. b
4. a

13. A reader file must be established, just as if a normal read were to occur, and this file may be shared between normal reads and column binary reads. The reader file must have had an alternate area assigned if it is to be used with column binary reads. Additionally, two work areas of identical size with the input area assigned to the card input file must be defined in the WORKING-STORAGE SECTION or in the record area of another file. These may be the same work areas also used for column binary punching. The reader file must be OPENed before use.

   1. A reader file may be shared between .... reads and .... reads.

   2. Two .... areas are needed if types of reads are mixed.

   3. The reader file must be .... before use.

* * *

1. Normal, column binary
2. Work
3. OPENed

---

14. When the QCBRn subprogram is CALLed, a binary image of the upper half of the card comprising punching levels 12-11-0-1-2-3 will be placed in the first work area named ARG01, with the lower card half comprising punching levels 4-5-6-7-8-9 placed in the second work area named ARG02, also in binary image form. The six bit levels established by holes in the card are set into the 1130 core word using bits 10-15, with bits 0-9 being reset to zero bits. The 12 or 4 level punch will set bit 10 continuing until the 3 or 9 level punch sets bit 15.

   1. A .... image of the upper half of the card is placed in the first work area.

   2. ARG02 is .... (Complete the sentence)

   3. The normal EBCDIC equivalent of all valid characters read is also placed in the ....

* * *

1. Binary
2. The second work area where the lower half of the card is stored.
3. Standard record area defined in the reader file.

---

15. Since invalid Hollerith card codes are expected with the above type of read, no validity check will be performed, and the execution time stop with Accumulator lights of /100E or /400E will not be provided. In the normal read input area, if a valid EBCDIC character cannot be derived from the punches read, a hex 00 will be established instead.

Following the CALL "QCBRn", a normal READ must be issued, at which time the normal EBCDIC character equivalents of the Hollerith card codes read will be produced, and placed into the normal read areas as defined in the card input file.

Match the items with their related terms:

1. Absence of read validity check
2. Statement needed after CALL QCBRn
3. Mixed EBCDIC-Binary data in input card

a. READ

b. EBCDIC placed in normal read area

c. Programmer choice

d. Invalid Hollerith card codes expected

e. Decimal replaces binary

* * *

1. d
2. a
3. b

---

16. Since a card read in column binary may have any combination of punches possible, a /* or // card will not be recognized for the meaning normally associated with such cards. Therefore, the user must provide his own test to recognize the logical end of a column binary card file. Note, however, that due to the alternate area specification for the file, the card following the end-of-file marker card will have been physically read by the READ statement which made the previous card (end-of-file marker) available to the user. The user must therefore provide one additional card (which will be ignored) following the end-of-file marker card.

A /* or // card will not be recognized as file end code because the QCBRn subprogram does not recognize these symbols as such.

a. True

b. False

* * *

a. Since a card read in column binary may have any possible combination of punches, the usual ending symbols might be accidentally produced in binary punches, and are therefore not allowed to effect their normal meaning in identifying the end of a file.

17.

```
"QCBPC" USING      ARG01 ARG02 ARG03

    ARG01 = Name of punch file defined in the program.
            Must have alternate area.

    ARG02 = Name  of  work  area  defined  in  Working
            Storage  of  equal  length  to  the  punch
            output area assigned to the file named  in
            ARG01.

    ARG03 = A  second work area identical  in  size to
            that named in ARG02.
```

QCBPC

Permits the subsequent punching of column binary cards on a 1442 on the next WRITE command. Punching rows 12-3 and 4-9 for each column are taken from bits 10-15 of two separate words. Checking for the punching of Extended Hollerith characters is suppressed. Used in conjunction with PU14Q, RP14Q, or PO14Q.

What is the chief reason for punching column binary codes into cards?

*   *   *

To get more data per card, or to store data in pure binary formats.

---

18. A punch file must be established just as if normal punch were to occur, and this file may be shared between normal punching and column binary punching. Additionally, two work areas of identical size with the output area assigned to the card output file must be defined in the Working Storage Section, or in the record area of another file. These may be the same work areas also used for column binary reads. The punch file must be OPENed before use.

    1. The punch file may be shared between .... punching and .... punching.

    2. Two .... must be assigned

    3. The punch file must be .... before use.

*   *   *

1. Normal, column binary
2. Work areas
3. OPENed

---

19. When the QCBPC subprogram is CALLed, the normal Hollerith card codes of the EBCDIC character equivalents established in the normal punch output area defined in the punch file will be punched where present, with one exception, where a hex 00 character is discovered.

Note that as in the column binary READ, the CALL "QCBPC" must be followed by a WRITE statement referencing the normal punch record name. No actual punching occurs until this statement is issued.

Match items with corresponding facts:

1. Punched output, QCBPC subprogram
2. After CALL "QCBPC"
3. Punch occurrence

a. Hollerith

b. WRITE statement must be issued

c. Column Binary

d. After WRITE statement

e. After CALL "QCBPC"

* * *

1. a,c
2. b
3. d

---

20. QCTBL

A conversion table stored on the disk in the form of a subprogram specifying a 40-character set similar to that suggested for Commercial Subroutine Package, but which generates codes retaining full collating sequence. The user may replace this with his own table if a different character set is desired.

Why is QCTBL not an end-purpose or stand-alone subprogram?

* * *

The conversion table brought into core is passive in and of itself, but used by other subprograms.

---

21. Supplied with the 1130 COBOL compiler are two subprograms - QCORE and QFIND. These subprograms provide support for creating and using an in-core indexed access to a file which has been organized according to conventions explained hereafter. This file is basically sequential in nature, with a non-sequential additions area. It may be accessed randomly via the record key contents -- not merely by relative record numbers.

QCORE - ESTABLISH AN IN-CORE INDEX TO A FILE,
USING ALL AVAILABLE CORE

QFIND - LOCATE A RECORD RANDOMLY, BASED ON
CONTENT OF A RECORD KEY

1. .... established an in-core index to a ....

2. .... locates a .... through use of a key.

3. A .... contains the record embedded data being searched for.

* * *

1. QCORE, file
2. QFIND, record
3. record key

---

22. QCORE

Dynamically allocates all otherwise unused core for use as an index to be used by QFIND. Establishes necessary tables required by QFIND.

QFIND

Locates a record within a sequentially organized file. Returns the relative record number of a record in the file whose nominal key is equal to the record key presented as an argument to the subroutine.

Match items with their appropriate factors:

1. Dynamic core allocation

2. Relative record number

3. Search Arguments

a. Data used as a nominal key for locating a desired record

b. Shows disk location of record in terms of position in the file

c. Dependent on program size and size of core memory

d. Example: EMPNO

* * *

1. c
2. b
3. a,d

SUMMARY

In this lesson you learned to use or were introduced to some of the CALLable subprograms supplied with 1130 COBOL. These subprograms are intended to fill a variety of needs - data manipulation, storage conservation, program debugging, and disk file accessing. You have also seen that certain conventions must be observed, such as the definition of CALLable subprogram arguments.

The next lesson will provide additional information on QCORE and QFIND presented briefly here in the last two frames.

END OF LESSON 41

LESSON 42

LESSON 42 - FILE ACCESSING TECHNIQUE

INTRODUCTION

This lesson presents in detail the method of using QCORE and QFIND covered briefly in the preceding lesson. These two CALLable subprograms implement a method of locating records in a file based on the content of the data in a control field embedded in the record. It has been included with 1130 COBOL as a substitute for an Indexed Sequential Access Method, which was not possible to implement. A summary of the features and restrictions of this access method follows:

FEATURES

- A file may be established which permits logical additions and deletions to occur, without rewriting the entire file each time.
- Records may be located in this file, based on the content of a nominal key in core matching a record key on a record in the file.
- A new index is created in core with each new use, eliminating a file index and maintenance to the same resulting from updates.
- Construction of the core index does not require a special reading of the file -- it is constructed dynamically as the file is used.
- No fixed space in core is required for the index -- it uses all otherwise unused core remaining after the program is loaded.
- Basic search technique is a binary file search. As each midpoint is located, the record key found is stored in the index if missing.
- If the desired record is not located in the sequential portion of the file, the additions area is searched sequentially.
- The file may be easily reorganized each time a purely sequential processing is required or when access time becomes excessive.

RESTRICTIONS

- The user must always specify ACCESS IS RANDOM when defining a file to be used with these subprograms, except when first creating the file.

- Only a single file may be accessed randomly using these subprograms during the operation of a single main line program, although many other files using standard access techniques may be used in the same program.

- A small minimum available core must be present to use this program at all. When the index core is small, processing accesses will be relatively slow; when much core is available, relatively fast.

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

QCORE	CALLable subprogram
QFIND	CALLable subprogram

This lesson will require approximately one hour.

1. Record Key

A field defined in a record stored in a file on disk containing data by which a record can be located. The records in the sequential portion of the file must be in logical ascending sorted order based on the contents of the record keys. For example, a payroll file organized by employee number has a record key which is the field in which the employee number is stored. A record key must always be defined as a separate unique field, contained exclusively within a given number of full words in the record. For example, a record key can be compressed by any of the supporting techniques, as long as the record key is compressed as an individual item, and not part of a group item.

1. The disk-located .... contains data by which a disk record can be located.

2. The record in the main part of the file must be in .... order.

3. The search argument data used as a record key must always be kept in a .... field.

* * *

1. Record key
2. Sequential
3. Unique

---

2. Nominal Key

A field defined in core, which is initialized with the data contents to be compared against the contents of the record key field for each new record to be located in the file. The data stored in the Nominal Key must be in the identical format as the data stored in the record key -- no data conversions will occur at the time a comparison is made with a record key while searching for the desired record.

Match items with related factor:

1. Record key	a. Core located search data field
2. Nominal key	b. ACCESS IS RANDOM
3. Required main file organization	c. Sequential
	d. Disk located data identifying record
4. Required Access Statement in program	e. Punched in card

* * *

1. d
2. a
3. c
4. b

---

3. File Label Record(s)

One or more records at the physical beginning of the file space allocated, containing variables required by the program to identify the current ending location of the sequential portion of the file and of the additions area.

The File Label Record contains the labels of most disk stored records.

a. True

b. False

* * *

b. It contains variables needed to identify the present ending location of the sequential part of the file and of the additions area.

---

4. Sequential Portion of the file

That area of the file immediately following the last header record, containing records initially created in the file in sorted sequence by record key contents.

Additions Area of the File

That area of the file immediately following the last data record in the sequential portion of the file through and including the last addition to the file. The EOF Marker record (if any) will be the first record in this area. This area does not include any as yet unused space allocated to the file.

Possible Expansion Area

That area of the file space originally allocated beyond the last addition to the file up to the last position of the file in which a record may be added.

Match the items with the relevant factors:

1. Sequential portion of the file
2. Additions area of the file
3. Possible expansion area

a. File space allocated for future records

b. Follows the last file addition

c. Contains EOF Marker record

d. Non-sequential organization

e. Contains the initially created file records

* * *

1. e
2. a,c,d
3. a,b,d

---

5. File Definition Table

A structure in Working Storage defined with a level 01 name and containing five elements defined with level 02 names. These elements define the constants and variables associated with the location and length of the record key in each record, the current beginning and ending of the sequential portion of the file, and the end of the additions area of the file.

The File Definition Table defines these elements:

1. Location and length of .... in each record.

2. Current start and ending of the .... of the file.

3. End of the .... of the file.

* * *

1. Record key
2. Sequential portion
3. Additions area

---

6. Deleted Record

A record with a hex /8000 character written in the first word of the record, anywhere in the file space allocated to actual data records, but excluding the potential expansion area. The EOF Marker (if any), identified by the same code, is initially located in the first record position of the potential expansion area, and will be later replaced with the first addition to the file.

As a record is deleted, its place in the allocated file space is immediately erased.

a. True

b. False

* * *

False. The record is coded as being deleted and left in place pending a future file reorganization.

---

7. Two types of file organization conventions may be used within these subprograms. In one, a strictly conventional sequential disk file may be searched to locate records based on the contents of a record key. This method does not support deletions or additions to such a file. A second file organization is provided for users who desire the capability of deleting records, or adding new records to an existing file, without rewriting the entire file. In order to establish a common standard among users of this latter method, certain conventions are provided.

Of the two file organization just described, which would be better for a volatile file?

* * *

The second, because provision is made for change, whereas the first method is fixed until a major reorganization is made.

---

8. Some of the conventions regarding the second method are:

1. A total space must be allocated on disk sufficient to contain the label records at the start of the file, all sequentially sorted records to be placed in the file initially, plus a potential expansion area containing space for additions to be added later.

2. The first record or records must be a label record(s) containing at a minimum a total of four words. If each record is of a size of 4 words or larger, only one header record will be needed. If each record is but 2 or 3 words in length, two records are needed. If each record is but one word in length, then four records are needed. If any space remains in the header after allocating the four words, it is suggested that at a minimum the date the original file was created be recorded, plus the last date when additions were added. Conventions for such dates are left to each user.

In the second method which permits file additions:

a. Disk space must be allocated for .... separate areas of the file. (how many?)

b. Header record(s) with a minimum of .... words must be provided.

* * *

a. Three: label records, initial sequential records, exapansion area.
b. Four

---

9. Some more conventions are:

   1. The label records will be followed by the sequential portion of the file. Records contained here must have been sorted into logically ascending sequence, based on the contents of the record key in each record.

   2. Following the sequential portion of the file, an EOF Marker record may be present, but need not be. This is followed by unused but allocated record space sufficient for any planned additions to the file.

   3. Additions to the file may be added in occurrence sequence to the potential expansion area, in an unsorted sequence. The first addition must overlay the EOF Marker record, if any.

   Match items with their corresponding factors:

   1. Record keys

   2. Additions

   3. EOF Marker Record

   a. Follow sequential part of file

   b. Used to sort initial records before creating the file

   c. In occurrence sequence

   d. Compared against nominal keys

   e. Replaced by first addition to file

* * *

1. b,d
2. a,c
3. a,e

10. The last of the conventions are:

    1. Deletions to the file must be tagged by overwriting a hex /8000 character in the first word of the deleted record. If a deleted record may be logically reopened by the program prior to reorganization, then a special word must be defined first in the record to prevent destruction of vital data when the record is initially deleted. The record key may not begin with the first word in the record for any file for which deletions are permitted, since this would destroy the sort sequence of the file.

    2. When new records are added to the file, the fifth element in the File Definition Table defining the relative record number of the last record in the additions area must be updated. Since it is possible to intermix additions with searches for records -- including a search for the latest record added, if this were not done, the search would terminate prematurely.

    True or false:

    1. Deleted records may be restored in place on disk.

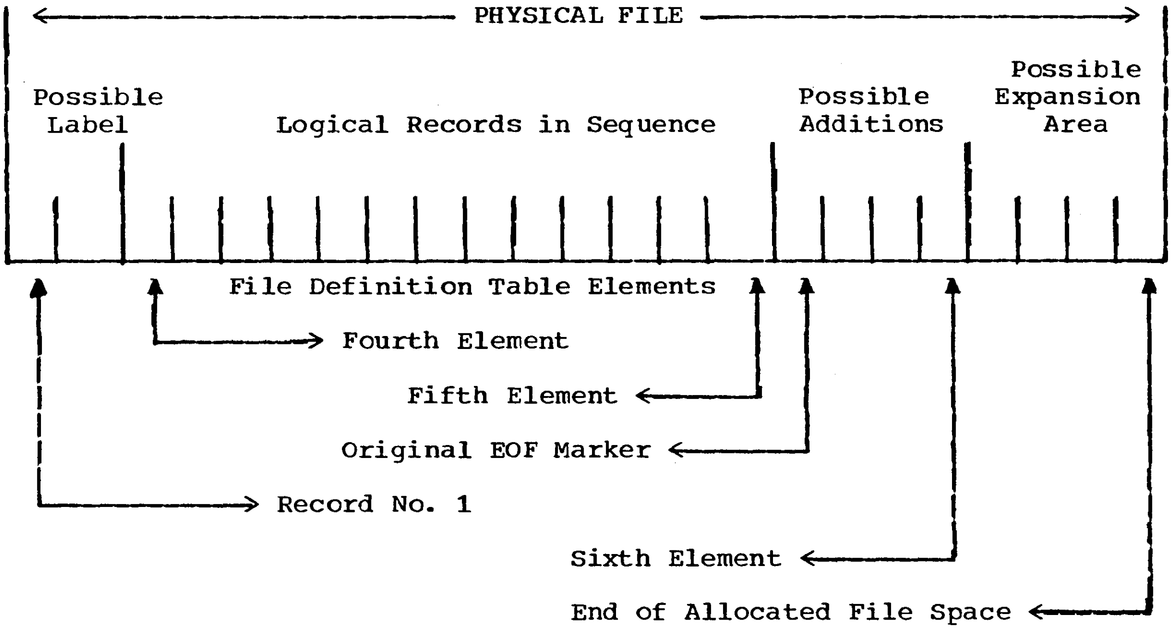
    2. The File Definition Table must be updated for each group of additions to the file.

* * *

1. True provided the first word of the record is reserved.
2. True

---

11. The following diagram illustrates the conventional organization of a file to be used with these subprograms:



Unscramble the numbers of major parts of the Physical Disk File arranging them in the proper sequence. Put the first number on the left:

(1)	(2)	(3)	(4)
Possible Expansion Area	Possible Label	Logical Records in Sequence	Possible Additions

* * *

2-3-4-1

---

12. The label record or records contains the variables of record ending locations, which may change during the life of the file due to updates. The header records contain only two fields of interest to these routines -- these are the relative record number of the last record placed in the sequential portion of the file and the relative record number of the last addition placed in the additions area (but this field will contain the same as the first field if no additions have yet occurred). Both fields must be defined to the program as PICTURE 9(5) COMPUTATIONAL.

What step do you think would be necessary when the additions area is full?

* * *

A major file reorganization would be necessary.

---

13. The first two words in the label record(s) must contain the last relative record number containing data in the sequential portion of the file -- this is the record that would be just before the EOF Marker record if such is present. This field must be loaded but once -- at the end of the initial creation of the sequential file.

The first two words of the label record(s) do not change during processing because ....

* * *

The sequential file area does not change.

---

14. The second two words in the label record(s) must initially contain the same value as the first field, since at completion of loading of the file, no additions would yet have been added. Thereafter it must be updated at the end of each run which makes additions to the file to contain the last relative record number containing data in the additions area of the file.

At times a record may be very small, such as a tag file. If a record is less than four words, then one header record cannot contain the minimum required contents to initialize the File Definition Table required by the routine. The convention in this case is to assign multiple header records until all space required has been accommodated.

The second two words in the label record(s) may change because ....

* * *

The end of the additions area changes with each run which adds new records to the file.

---

15. Normally, records will be larger than the four words required by these routines. If so, the use of such additional space, if used at all, is strictly up to the user. It is recommended, however, that the user establishes his own conventions for retaining the date that the file was created, plus the date that the last addition was added to the file. When backup records are retained for some period of time, the careful use of such dates can prevent processing with an older version of the file from occuring accidentally.

The user should develop his own method of using the .... as a control for file change iteration

* * *

Date

---

16. Prior to the first use of the file, the program must have a file definition table available in Working Storage of the main line program. This table contains five elements at level 02, defined by a single level 01 name. The first three of these elements are constants, based on the fixed details of the particular file to be used. The last two elements contain variables, defining the current ending locations of the sequential portion and the additions area of the file.

    1. The program requires a .... in Working Storage before using the file.

    2. The last two of the five table elements define current ending location of the .... and the .... of the file.

* * *

1. File Definition Table
2. Sequential portion, additions area.

---

17. All elements are in COMPUTATIONAL format and for this reason this specification must appear together with the level 01 name of the table. The first two elements must each comprise one word of storage, which will be assigned by reason of a PICTURE 9(4) specification with each. The last three elements must each comprise two words of storage, which will be assigned by reason of a PICTURE 9(5) specification with each.

    True or false?

    1. All table elements are COMPUTATIONAL in format.

    2. The format designation appears with the table name which is level 02.

* * *

1. True
2. False. The table name is a level 01.

---

18. The file definition table must be defined in Working Storage as follows:

    01 Name of the File Definition Table.
       COMPUTATIONAL must be specified.

        02 Number of the first word in each record at which the record key begins.

        02 Number of whole words in the record key.

        02 Number of the first relative record at which the file is to be searched.

        02 Number of the last relative record which is located in the sequential portion of the file.

        02 Number of the last relative record which is located in the additions area of the file.

    See how many File Definition Table elements you can recall.

* * *

1. Record key beginning word.
2. Number of whole words in the record key.
3. First relative record for the file search.
4. Last relative record in sequential portion of the file.
5. Last relative record in additions area of the file.

---

19. As mentioned earlier, one possible file organization is that of a purely sequential file created by purely conventional means. If this is so, then no label records will be present, and no additions area will be provided for. In this example, the third element of the File Definition Table must be initialized with a value which is equal to the relative record number of the last record in the file, excluding the EOF Marker record. This number must be determined at the time the file is created, and by some means caused to be placed into the last two elements in the table.

    Which item is not relevant to the conventional sequential file organization?

    a. No File Definition Table elements

    b. No label records

    c. Last record designation

    d. No additions area

* * *

a. (Last record designation, for example, is a table element.)

---

20. Sample Definition Table in Working Storage

```
0   0    1    1    2    2    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  TABLENAME COMPUTATIONAL.
             05  REC-KEY-START       PICTURE 9(4) VALUE IS 10.
             05  REC-KEY-LENGTH      PICTURE 9(4) VALUE IS 5.
             05  FIRST-SEQ-REC-NO    PICTURE 9(5) VALUE IS 2.
             05  LAST-SEQ-REC-NO     PICTURE 9(5).
             05  LAST-ADDN-REC-NO    PICTURE 9(5).
```

Fill in the blanks with proper name segments: COMPUTATIONAL, FIRST, LENGTH, LAST START. Ignore PICTURE and VALUE data.

a. 01 TABLENAME ....

b. 05 REC-KEY-........

c. 05 REC-KEY-........

d. 05 ....-SEQ-REC-NO

e. 05 ....-SEQ-REC-NO

f. 05 LAST-........-REC-NO

* * *

a. COMPUTATIONAL
b. START
c. LENGTH
d. FIRST
e. LAST
f. ADDITIONS

---

21. User Responsibilities

In order to provide maximum flexibility, the QCORE-QFIND routines provide the minimum processing necessary to support location of records in a file defined according to the conventions provided here. The user is therefore required to provide additional support through coding in his program to perform the following:

1. The user is required to construct the header records according to the required formats, at the time the file is created initially.

2. The user is required to create the sequential portion of the file initially. It is permitted to use the standard sequential file creation support provided by 1130 COBOL. The EOF Marker record written by the sequential file create support will not be a problem during subsequent processing.

3. The user is required to add any additional records to the additions area of the file, using a WRITE with ACCESS IS RANDOM, defining the ACTUAL KEY through operation of his program.

4. The user is required to initialize the fourth and fifth elements in the File Definition Table with the contents of the label records, prior to CALLING QCORE, keeping the fifth element updated as additions are added.

5. The user is required to update the header records at the end of processing additional records which have been added to the file.

6. The user is required to recognize the deletion code through operation of his program if he does not desire these records to be presented for processing when located by QFIND.

7. The user is required to write the deletion code (hex /8000) into the first word of any record which he desires to be logically deleted from the file.

8. The user is required to reorganize the file, whenever purely sequential access is required, or when processing time is excessive. The program to reorganize the file is a user responsibility.

The user is not required to do two of the functions below. Which two?

a. Initially create sequential part of file.

b. Add additonal records.

c. Move the EOF Marker during processing.

d. Erase deleted records.

e. Reorganize the file.

* * *

c: achieved by standard sequential file support.
d: only the deletion code is needed.

---

22. The QCORE routine does not itself generate an index -- this is left to QFIND as previously unindexed accesses are required to locate a record. QCORE does establish the area of core to be used by the index, and clears this area to hex /0000 at such time. Finally, entries in the File Definition Table are used to establish processing contants required for the proper operation of QFIND. These constants are stored in core in the area normally used by FORTRAN as a floating point accumulator, but never used in a COBOL program.

Match the items with the appropriate factors:

1. QCORE	a. Generates an index
2. QFIND	b. Establishes core area
	c. Clears core area
	d. Creates sequential file

* * *

1. b,c
2. a

The file is created by conventional 1130 COBOL sequential support.

---

23. QCORE uses all otherwise unused core not required by the main line program or any of the subroutines, etc. A minimum of core is required for the routine to be used at all -- when this minimum is not available, an execution time error message will be presented (error number 82). It is usually possible to free up enough core through the use of LOCALs, etc., although obviously execution speed of disk accesses will suffer when little core is available for the index.

1. .... of disk accesses suffers when little core is left.

2. If minimum core is not available, an execution time .... occurs.

3. .... often makes available enough core for an index.

* * *

1. Execution speed
2. Error message
3. Use of LOCALs

---

24. The basic record location technique used by this routine is a binary search. In the normal binary search, when no index is available, the middle record in the file is the first accessed, and the record key of this record examined to determine whether the nominal key stored in core is higher or lower. The half of the file in which the desired record must be located would itself then be bisected, and a new comparison made. Identification of the appropriate quarter file may then be made, and the process continued. This might be continued until the record is finally located, or at a certain point a purely sequential search might be begun.

True or false?

1. In normal binary search (when an index is not used), every record key is examined sequentially.

2. Sequential search is partially used in the method just described.

*   *   *

1. False. A process of progressive halving of the file is used.
2. True

----

25. The difference between the present routine and a straight binary search as just described, is that as each midpoint record is examined, the record key found there is retained in a core index. In this index the position of each cell in the index array defines its position in terms of the file search algorithm sequence. Thus having once been actually accessed, the record key contents are saved to eliminate having to make any further accesses of at least those records. As new paths are taken from time to time, previously untried accesses will occur, further filling out the index. If by the end of processing all records have been retrieved, then the index will at such time be complete. This will normally never occur.

1. In the present method, the .... is saved each time a previously untested midpoint record is accessed.

2. Such a retention avoids ....

3. Usually the .... is never built for all records in the file.

*   *   *

1. Record Key
2. Further accesses of records previously searched
3. Index

----

26. Binary searches apply only to the sequential portion of the file. When a record cannot be located in this area, then the additions area of the file is searched sequentially, looking for the record. When a record still cannot be found, a "not-found" condition is signaled by placing a record number of all-zeros or a negative value in the ACTUAL KEY area. When the READ occurs, the INVALID KEY action will then be performed. Since the only INVALID KEY condition should be that of a "not-found", the action programmed to occur upon INVALID KEY can perform the desired processing.

    1. The additions file area is searched ....

    2. If the record is not found, a non-positive value is placed in the ....

    3. If the record is not found, the action occurs when READ occurs.

*　　　*　　　*

1. Sequentially
2. ACTUAL KEY
3. INVALID KEY

---

27. The first cell in the index array will contain the record key contents found in the middle record of the entire sequential portion of the file. The next two cells contain the equivalent data from the middle records of each file-half. The next four cells contain the equivalent data from the middle records of each file-quarter. The next eight cells contain the equivalent data from the middle records of each file-eighth. This continues until there is insufficient core remaining in the index area to contain the next binary exapansion of the index array, although to the extent that such core is in fact available, cells will be allocated.

    Name three conditions or events which will stop the binary search.

*　　　*　　　*

1. The record is found
2. The index area in core lacks space.
3. The record is missing, and the file is completely searched.

---

28. The final cells in the array are temporary cells, and will be repetitively written as actual accesses occur outside the index, until the desired record is located. The temporary cell will assist in locating a record only if the next access taking this path is to a location in the file near to the last access of that path. This is frequently true, but if not, then actual accesses will replace it until the newly desired record is found. Note that table size determines whether many accesses will be required to locate records, or few.

Match items with their corresponding factors:

1.	Final array cells	a.	Related to number of accesses
2.	Table size	b.	Every time
3.	Temporary cell is benefitial in search	c.	Temporary
		d.	If next search is near to last search path

* * *

1. c
2. a
3. d

---

29. As mentioned previously, QCORE makes use of the floating point accumulator in core (actually located in the transfer vector area) to store constants required by QFIND. This is permissable, since COBOL does not support floating point data formats, and thus does not use this accumulator. Since a FORTRAN subprogram may be CALLed within a COBOL program, however, it is possible that a floating point operation may actually occur, and thus destroy the contents of this accumulator. To insure recognition of this condition, a check-sum is computed on the contents stored along with the data. QFIND then verifies that the check-sum is correct with each use. If it is not, an execution time error will occur (error number 81).

1. QCORE stores constants needed by ....

2. A FORTAN subprogram floating-point operation might destroy the contents of the ....

3. If the contents are destroyed, an .... occurs.

* * *

1. QFIND
2. Floating point accumulator
3. Execution time error

---

30. It is possible to execute QCORE repetitively, since with each execution it actually clears the core area allocated to the index enabling a restart. If a FORTRAN subprogram using floating-point data or the COMMON area is necessary to a program, QCORE must be CALLed after each such use. If done, however, the processing speed of QFIND is temporarily decreased greatly, since the index must be rebuilt from the beginning.

True or false?

1. QCORE may be executed repetitively.

2. Special core-clearing cards are needed with QCORE reiterations.

3. If QCORE is CALLed after floating-point data has been used, QFIND processing speed decreases temporarily.

* * *

1. True
2. False. QCORE clears core.
3. True

---

31. Two abnormal program execution termination conditions may occur through the use of QCORE and QFIND. These produce the usual message to appear on the principal print device, as shown in the Operations Manual. In this message, **ERR=YY** will have substituted for the "YY" one of the following two codes:

81 = Floating-point accumulator contents destroyed. Eliminate the use of floating-point arithmetic in a FORTRAN subprogram used with this program, or else re-execute QCORE following each subprogram.

82 = Insufficient core available for the index. Consider LOCALing some subprograms to free up core, or else reduce program functions.

Name one condition involving QCORE and QFIND wherein an abnormal program execution termination may occur.

* * *

a. Floating-point accumulator contents destroyed.
b. Insufficient core available for the index.

32.

" QCORE" USING FILENAME, TABLENAME

Two arguments are required by this subroutine. The first is the filename of the disk file to be processed. Note that this file must have been defined in the program with an ACCESS IS RANDOM clause. Also this must be the same name assigned to the file in the FD of the FILE SECTION in the COBOL program.

The second argument is the name of an area in Working Storage containing the File Definition Table. The level 01 name of this core table must be specified as the name in the argument list. This table must be defined in the main line of the program , and remain in core for the full time that the file is processed -- not just during the time that QCORE is CALLed. The contents of the File Definition Table have been fully described previously.

What two arguments are required by QCORE?

* * *

1. File name of the disk file to be processed.
2. Name of Working Storage area containing the File Definition Table.

33.

" QFIND" USING NOMINAL-KEY.

This subprogram has but one argument -- the name of a field defined in a storage area other than that used to contain the record contents of the file being searched. The field name must contain the Nominal Key contents to be used in selecting a record from the file.

The file named in this argument must have been defined identically in data format and length with the record key field appearing in the disk file record to be located. No data conversion will take place.

True or false?

1. QFIND has one argument, the name of a field containing Nominal Key contents.

2. The file named in the QFIND argument must be defined identically with the disk file record key field.

* * *

1. True
2. True

---

34. Note that only a single file can be processed in a given main line program using these subprograms (although many other files not using these subprograms can be processed within the same program). As a result, the single specifications of the FILENAME in the QCORE subprogram serves to identify which file is to be used with QFIND, without having to repeat this as an argument in QFIND.

1. .... file(s) can be used in a main line program using QCORE and QFIND.

2. QCORE file specification also serves ....

* * *

1. Only one
2. QFIND

---

35. The Nominal Key may be a field in core, or in the input record area of another file (but not of the file being processed), and is in no way changed by the operation of QFIND. Thus a record can be obtained from one file (either card or disk) which will contain the key of a record in another file. This key does not have to be in the same location for each use of QFIND within a single program, and for that reason was not included as an argument to QCORE.

1. QFIND (changes, does not change) the Nominal Key contents.

2. The record key (does, does not) have to be in the same location for each use of QFIND within a single program.

* * *

1. Does not change
2. Does not

---

36. During the operations of QFIND, the desired record will be located, and the relative record number placed in the area of core named as the ACTUAL KEY when defining the file being processed. The QFIND subprogram must be followed by a READ to make the record actually available to the program. If QFIND finds the record key within its index, it will not actually access the record, leaving this task for the READ to perform. If it does not find the record key in the index, it will have actually accessed the record, leaving it in the record area. Since the READ will discover that the desired record is already in core, no additional physical read will occur in this case, although the READ is still necessary to assure that the proper record within the block is made available to the program.

1. When the record is found, its .... is placed in the ....

2. QFIND must be followed by a ....

3. The .... actually accesses the record.

* * *

1. Relative record number, ACTUAL KEY
2. READ statement
3. READ statement

---

37. If the record located is to be updated, the READ can be followed by a WRITE, after appropriate changes to the record content have been made. Another QFIND, however, must not be CALLed before the WRITE, since this might bring in another copy of the original record, removing the changes desired to be made in the update. If the contents of the records other than the record key are unimportant, it is possible to follow a QFIND with a WRITE instead of a READ statement, but the contents of the record area must be established after the QFIND and before the WRITE, since QFIND may bring in a copy of the record, erasing the data desired to be written.

1. To update the located record, a .... statement must follow the READ.

2. Another .... must not precede the WRITE if an update has been made to the record contents.

3. A .... statement may be used rather than READ if the original record contents are not required.

* * *

1. WRITE
2. QFIND
3. WRITE

SUMMARY

In this lesson you learned to use a special 1130 COBOL disk file construction and accessing technique. Reminiscent of the System/360 Index Sequential Technique, the 1130 counterpart is not supported in the formal COBOL language itself. However, the CALLable subprograms QCORE and QFIND are seen to enable automatic establishment of a core area for a disk file index and the subsequent creation of this index, and accessing of records via data stored in the record, rather than its relative record number.

This technique permits both easy accessing of records, and additions of new records to the disk files.

END OF LESSON 42

LESSON 43

LESSON 43 - PROGRAM OVERLAYS

INTRODUCTION

1130 COBOL contains excellent provision for handling programs too large to be contained in core in one segment. The user may create subprograms known as LOCAL's, or Load On Call program segments. One common core area receives all LOCAL's in a series of overlays, enabling the program to be processed regardless of its size.

The Core Load Builder, if it discerns that a main line program is too large for core, breaks up some of the systems subroutines of the program into manageable segments called SOCAL's, or System Calls, without the user's stipulation. When LOCAL's are included, they take priority over SOCAL's and often remove the need for the latter.

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

LOCAL control record
NOCAL control record
SOCAL system call

This lesson will require approximately one hour.

1. You will recall that there are several monitor control records that are used for each job set up. A brief description of all monitor control records is given below:

// JOB — Defines the start of a new job.

// COBOL — Causes the supervisor to read the COBOL compiler into core storage and transfers control to it.

// DUP — Causes the supervisor to read the control portion of the Disk Utility program into core storage and transfer control to it.

// XEQ — Causes the supervisor to initialize for core load execution and transfers control to it.

// PAUS — Causes the supervisor to wait.

// * — Allows the user to print alphameric text on the listing.

Monitor control records cause the performance of the load and control functions of the Monitor System. This lesson will concentrate on special functions initiated by the // XEQ card, in particular those involving the LOCAL.

The .... card may cause the LOCAL special function to be initialized.

*   *   *

// XEQ

---

2. // XEQ

This control record causes the supervisor to initialize for core load execution. If the name specified in this control record (columns 8 through 12) is that of a main line program stored in Disk System format, the Supervisor reads the Supervisor control records (LOCAL, NOCAL, FILES or G2250), if any, from the input stream and writes them in the Supervisor Control Record Area (SCRA). The Core Load Builder is then called to build a core load image program from the main line program.

Match each item with its corresponding factor:

1. LOCAL
2. Core Load Builder
3. // XEQ

a. Causes reading of Monitor control records

b. Makes core load executable program

c. Provides for subprogram overlays

d. Part of input stream

*   *   *

1. c,d
2. b
3. a,d

---

3. If no name is specified on the // XEQ control card, a main line program in Disk System format is assumed to be stored in the Working Storage of the disk cartridge whose I.D. is specified in columns 21-24. The Supervisor then processes the Supervisor control records and calls the Core Load Builder via the LINK entry point in the Resident Monitor.

After the core image program has been built, or if the name in the control record is that of a program already stored on disk in DCI format, the Core Image Loader is called to read the core load into core storage and transfer control to it.

True or false?

1. The main line program name must always be present in the // XEQ card.

2. The Core Image Loader hands over control to the program.

* * *

1. False. If it is not, the main line program is assumed to have been stored in Working Storage, normally having just been compiled.
2. True

---

4. If the name specified in the // XEQ control card (columns 8-12) is that of a main line program stored in Disk System format, or is omitted, the Supervisor accepts the number of Supervisor Control Cards (LOCAL, NOCAL, FILES) specified in cc 16-17 if any, and honors them in the building of the core load for immediate execution.

The LOCAL and NOCAL Supervisor Control Cards are described below.

*LOCAL Load on Call	Causes a common core area to be established the size of the largest LOCAL subprogram. Each LOCAL identified subprogram will then share the common area at execution time, coming into core when required by the program. Since one LOCAL cannot CALL another LOCAL, only a single level, normally the lowest level in the overlay hierarchy, can be LOCALed.
*NOCAL Load Uncalled	Causes a subprogram to be loaded into core although no CALL to the routine was coded.

1. The *LOCAL card reserves a core area as large as ....

2. Each LOCAL subprogram shares ....

3. LOCAL means ....

* * *

1. The largest LOCAL subprogram
2. A common core area
3. Load on Call (i.e., a subprogram)

---

5. The following function does not require a Supervisor Control Card:

   (SOCAL) System Call Similar to LOCAL, but applies to system routines only, is entirely automatic, and no control card is required or exists. An area separate from the LOCAL area will be used for overlays which are SOCALed.

   1. SOCAL means ....

   2. SOCAL's apply to ....

   3. Both SOCAL's and LOCAL's are subprograms which are ....

* * *

1. System Call
2. Automatically called system routines
3. Overlayed in core

---

6. The control records described below (LOCAL, NOCAL) are used by the Core Load Builder to:

   - Provide for subprogram overlays during execution (LOCAL).

   - Include subprograms not called in the core load (NOCAL).

   1. A LOCAL is identified with a subprogram overlayed during (compilation, execution).

   2. NOCAL's (are, are not) CALLed in the main line program.

   3. LOCAL's and NOCAL's are used to control the (object program, Core Load Builder)

* * *

1. Execution
2. Are not
3. Core Load Builder

---

7. The control records described previously are placed in the input stream following a // XEQ Monitor control record that names a main line program stored in Disk System format or following a STORECI control record. In either case, the control records are written on disk in the Supervisor Control Record Area (SCRA), from which the Core Load Builder reads them for processing.

Match each item with its corresponding factor:

1. Control records
2. // XEQ
3. Supervisor Control Record Area
4. Core Load Builder

a. Area on disk

b. Monitor control card

c. Processes LOCAL's and NOCAL's

d. LOCAL's and NOCAL's

e. Punched in a card

*   *   *

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

---

8. LOCAL (load-on-call) subprograms are subprograms specified by the user to be read, one at a time, as they are called during the execution into a LOCAL overlay area. The LOCAL subprograms are specified on the LOCAL control record as follows:

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

*LOCALMAIN1,SUB1,SUB2,...,SUBn
```

where

MAIN1 is the name of the main line program already stored on disk. SUB1 through SUBn are the names of the LOCAL subprograms used with that main line program which must share a common overlay area in core.

True or false?

1. It is not necessary to have a separate LOCAL card for each subprogram to be overlayed.
2. LOCAL subprograms are read into adjacent core areas simultaneously.

*   *   *

1. True. Many subprograms may be named in the same card.
2. False. One subprogram at a time is called into a common core area.

---

9. In the case illustrated below, all the LOCAL control records except the last end with a comma (continuation character) and the main line program name appears on the first LOCAL control record only.

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

*LOCALMAIN1,SUB1,SUB2,
*LOCALSUB3,
.
.
.
*LOCALSUBn
```

Only the first LOCAL card contains both the .... and ....

* * *

Main line program name, called subprogram name(s).

---

10. The same results as in the above case would have been obtained, however, if the records had been:

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

*LOCALMAIN1,SUB1
*LOCALMAIN1,SUB2
.
.
.
*LOCALMAIN1,SUBn
```

As can be seen, only if the final comma (continuation character) is omitted, can a subsequent LOCAL control card contain the name of the main line program.

All the LOCAL subprograms for each main line program in an execution must be specified on the LOCAL control records that follow the // XEQ Monitor control record initiating the execution.

True or false?

1. Multiple main line programs may be associated with multiple LOCALed subprograms in a card.

2. The // XEQ Monitor control record initiates the execution.

* * *

1. False. Only one main line program is normally allowed.***
2. True

*** An exception is when QLINK is used to CALL a suicide overlay of another main line program to follow an earlier program. Here each main line program to be executed as a result of a single // XEQ control card may be separately named with their own sets of LOCALed subprograms.

---

11. Separate LOCAL control records must be used for each main line program in the execution that calls LOCAL subprograms. For 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..

*LOCALMAIN1,SUB1,SUB2,SUB5,...,SUBn
*LOCALMAIN2,SUB3,SUB4,...,SUBz
```

where

MAIN2 is CALLed by MAIN1 via the QLINK subprogram supplied with the compiler.

True or false?

1. Multiple main line programs may each have their own LOCAL cards.

2. Subprograms may be identified with one main line program.

*   *   *

1. True
2. True

---

12. If the main line program is to be executed from Working Storage, the main line program name must be omitted from the LOCAL control record. For 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..

*LOCAL,SUB1,SUB2,...,SUBn
```

This LOCAL control record format must be used if LOCALs are to be specified with the DUP operation STORECI.

No embedded blanks are allowed in the LOCAL control record.

If a main line program is to be executed from .... it must not be named in the ....

*   *   *

Working Storage
LOCAL card

---

13. NOCAL (load uncalled) subprograms are subprograms specified by the user to be included in the core load, even though they are not CALLed. They are specified on NOCAL control records using the same format that applies to LOCAL control records except that *NOCAL is used in place of *LOCAL.

   1. NOCAL means ....

   2. NOCAL's are to be included in ....

   3. NOCAL card formats are identical to .... formats.

   4. NOCAL subprograms are always core ....

* * *

1. Load uncalled
2. The core load
3. LOCAL
4. Resident

---

14. Core loads that utilize LOCALs will not necessarily run the same way with LOCALs as they would on a larger machine without LOCALs. This is due to the fact that every time a LOCAL is fetched, it is fetched in its initial state from the disk. Thus, unless it comprises read-only code, it will execute differently the second and subsequent times it is fetched. This same rationale applies equally well to SOCALs, at least in theory, but the IBM-supplied subroutines that are stored with SOCAL subtype codes are read-only code.

   True or false?

   1. LOCALed subprograms are always the same throughout program execution.

   2. SOCALed subprograms are comprised of read-only code.

* * *

1. False. The LOCAL is called in its initial state but embedded data elements may change as the main line program proceeds.
2. True

---

15. The Monitor system library contains a flipper subroutine (FLIPR), which is used to call LOCAL (load-on-call) and SOCAL (system-load-on-call) subroutines into core storage.

When a LOCAL subroutine is called, control is passed to the flipper, which reads the LOCAL into core storage (if it is not already in core) and transfers control to it. All LOCALs in a given core load are executed from the same core storage locations; each LOCAL overlays the previous one. FLIPR fetches SOCALs in the same manner as LOCALs, but into a different core area.

Match each item with its corresponding factor.

1. FLIPR (Flipper)
2. LOCALs
3. SOCALs

a. Load-on-call

b. System-load-on-call

c. Reads subprogram into core storage if required

d. Transfers control to subprogram

e. Common core area used

* * *

1. c,d
2. a,e
3. b,e

---

16. The LOCAL/SOCAL flipper is included in each core load in which LOCAL subprograms have been specified and/or in which SOCALs have been employed. If execution of the core load immediately follows the building of the core image program, this subroutine read a LOCAL/SOCAL from Working Storage into the LOCAL or SOCAL area as it is called during execution. If the core image program was stored in the User or Fixed Area in Disk Core Image format prior to execution, the flipper reads each LOCAL/SOCAL as it is called during execution from the User or Fixed Area (where it was stored following the core load) into the LOCAL or SOCAL area.

The flipper is entered via the special LOCAL/SOCAL linkage. A check is made to determine if the required LOCAL/SOCAL is already in core. If it is not in core, the flipper reads the required LOCAL/SOCAL into the appropriate area, and transfers the LOCAL/SOCAL subprogram via the special linkage.

1. The flipper reads subprograms into the ....

2. The subprogram may already be in .... or on ....

3. Flipper hands over control to ....

* * *

1. LOCAL or SOCAL core storage area
2. Core storage, disk
3. The subprogram

---

17. The LOCAL, NOCAL, and other control records are read from the Supervisor Control Record Area (SCRA) on disk and analyzed. Tables are built from the information obtained from the respective control record types. These tables are used in later phases of the construction of the core image program.

    1. LOCAL or NOCAL control records are read from the (core, disk) Supervisor Control Record Area.

    2. Tables are used in the construction of a ....

*   *   *

1. Disk
2. Core image program

----

18. When a Core Image program is being built, the main line program is first converted from Disk System format to Disk Core Image format. The main line is always converted before any other part of the core load.

Distribution of a Core Image Program Being Built

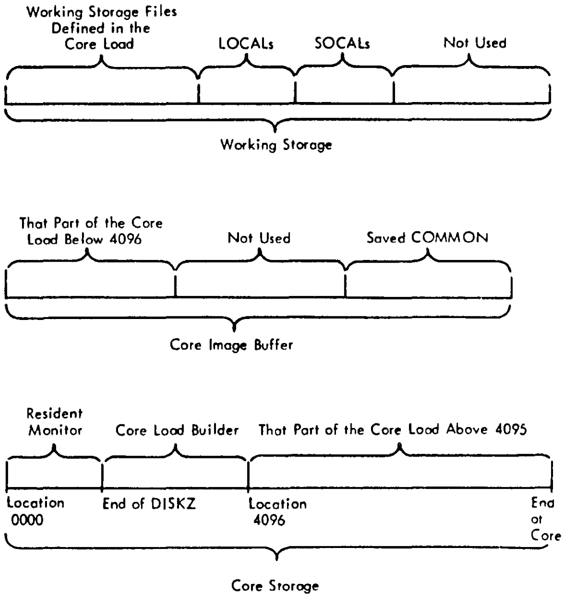


Figure 202

1. The main line program is converted from .... format to .... format.

2. The .... is the first part of the core load to be converted.

*   *   *

1. Disk System, Disk Core Image
2. Main line program

---

19. Figure 202 shows the layout of a core load loaded into core, ready for execution.

Layout of a Core Load Loaded for Execution

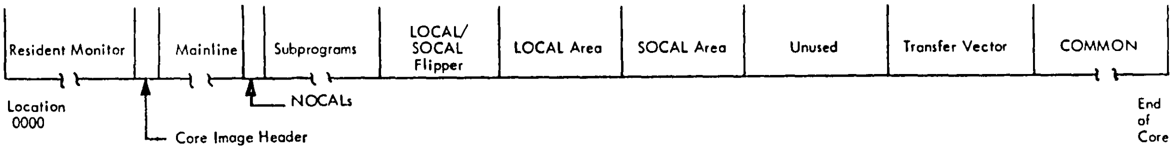


Figure 203

True or false?

1. NOCALs occupy the same core area as LOCALs.

2. All non-core-resident subprograms use the LOCAL core area.

* * *

1. False. NOCALs are always core resident.
2. False. SOCALs have their own separate area.

---

20. All the subprograms called by the main line program and by other subprograms are included in the core load, except for (1) the disk I/O subroutine, (2) any LOCAL subprograms specified, and (3) SOCALs.

If LOCALS have been specified, or if SOCALs are employed by the Core Load Builder, the LOCAL/SOCAL flipper (FLIPR) is included in the core load. The order of conversion is generally NOCALs, followed by the subprograms in the order they are called. The order of processing when flipper (FLIPR) is included is more complicated, and will not be discussed here.

Match each item with its corresponding factor.

1. LOCALs	a. Present in core load
2. Flipper	b. Not present in core load
3. Main line program	c. Sometimes present in core load
	d. Similar to SOCALs

* * *

1. b,d
2. c
3. a

---

21. If LOCALs have been specified, a LOCAL Area as large as the largest LOCAL is reserved in the core load, into which the LOCAL subprograms are read by the LOCAL/SOCAL flipper. In addition, the subprograms specified on the LOCAL control records are written in Working Storage following any files defined in Working Storage. If the core load is executed immediately, each LOCAL is read as it is called from Working Storage into the LOCAL area by the LOCAL/SOCAL flipper. If the core load is stored in Disk Core Image format before it is executed, the LOCALs are stored following the core load. During execution, the LOCAL/SOCAL flipper fetches them from the User/Fixed Area.

1. LOCAL subprograms during a core load build are written into .... following ....

2. LOCAL subprograms may also be stored eventually in the .... or in the .... area on disk.

* * *

1. Working Storage, any files defined
2. User, fixed

---

22. A subprogram cannot be specified as a LOCAL subprogram if it causes another subprogram, also specified as a LOCAL subprogram in the same main line program, to be called. For example, if A calls B and B calls C, and B is a LOCAL subprogram, neither A nor C can be specified as a LOCAL subprogram for the same main line program. If A were a LOCAL, then neither B or C could be LOCALed. Even though B were not a LOCAL, C could not be, since the return path to the main line program is through A, which will have been replaced by C.

True or false?

1. Under no circumstances can a LOCAL subprogram call another LOCAL subprogram in the same main line program.

2. There are subprograms other than the "CAL's".

* * *

1. True, since the return path to the main line program will have been destroyed
2. True, namely core resident subprograms

---

23. If a subprogram is specified as a LOCAL subprogram, and system overlays (SOCALs) are employed, the subprogram is made a LOCAL subprogram, even if it would otherwise have been inlcuded in one of the SOCALs.

If a subprogram is specified as a LOCAL subprogram, it is included as a LOCAL subprogram in the core image program even if it is not otherwise called.

1. A subprogram which would otherwise be made into a SOCAL sometimes is made into a ....

2. A .... does not have to be CALLed to become part of the core image program.

* * *

1. LOCAL
2. SOCAL or a NOCAL

---

24. When using LOCAL control records, care must be taken to adhere to the following formula:

$3M + 2S \leq 640$

where

M is the total number of main line program specified in the LOCAL records and

S is the total number of subroutines specified in the LOCAL record.

If execution is from Working Storage, the main line program already in Working Storage as a result of having just been compiled is counted as one, although it is not specified in the LOCAL records. The restriction also applies to NOCAL control records.

The restrictive formula regarding the number of LOCALs permitted (would, would not) impact the average 1130 COBOL program

* * *

Would not

---

25. Certain hardware considerations are worth noting in refernce to LOCALs. It would be very difficult, but not impossible, to successfully employ multi-cartridge files on a single drive system. The Core Load Builder constructs a working copy of any overlay phases (LOCALs and SOCALs) required by a program in a special area on disk. Were this cartridge replaced by a cartridge containing the continuation of a file begun on an original cartridge, the working copy of the overlay phases would not be present. It would be possible to overcome this difficulty by requiring that all routines be entirely core resident. This would be ensured by reference to a core map produced by the Core Load Builder.

1. Care must be taken not to lose the .... when multi-cartridge files are used on a single drive system.

2. This limitation can be avoided if .... is used instead of disk for all subprogram residence.

* * *

1. Working copy of overlay phases
2. Core storage

---

26. If SOCALs are employed by the Core Load Builder, a SOCAL area as large as the largest SOCAL is reserved in the core load, into which the SOCALs are read by the LOCAL/SOCAL flipper. In addition, the subprograms comprising the SOCALs are written in Working Storage following any files defined in Working Storage and any LOCALs stored there. If the core load is executed immediately, each SOCAL is read from Working Storage into the SOCAL area by the LOCAL/SOCAL flipper as it is called. If the core load is stored in Disk Core Image format before it is executed, the SOCALs are stored following the core load and the LOCALs, if any, in the User/Fixed area.

During execution, the LOCAL/SOCAL flipper fetches the SOCALs from the User/Fixed Area.

Match each item with its corresponding factor:

1. SOCAL   a. Reserved area with the size of the largest

2. Flipper   b. Is always core resident

3. LOCAL   c. May reside in Working Storage

4. NOCAL   d. Reads LOCALs and SOCALs into core, and transfers control to them

* * *

1. a,c
2. d
3. a,c
4. b

---

27. Enough Working Storage is reserved by the Core Load Builder to contain any Data Files assigned by the Core Load Builder to Working Storage. All the LOCAL subprograms and SOCALs, respectively, are stored in Working Storage following any files defined there. Figure 201 shows the distribution of a core image program between core storage, the CBI, and Working Storage. These diagrams depict a core image program just after it has been built but before it has been stored (STORECI).

   1. Working Storage may contain ...., ..., and ....

   2. STORECI is used to store the ....

* * *

1. LOCALs, SOCALs, Data Files
2. Core Image Program in the User/Fixed area

---

28. The Transfer Vector is a table included in each core load that provides the linkage to the subprograms. It is composed of the LIBF TV, the Transfer Vector for subprograms referenced by LIBF statements, and the CALL TV, the Transfer Vector for subprograms referenced by CALL statements. LIBFs are used only by programs written in Assembler Language.

   1. The .... is a linkage table in the core load, providing linkage to the ....

   2. Subprograms in COBOL may be referenced by .... statements.

* * *

1. Transfer vector, subprogram
2. CALL

---

29. SOCALs (system-overlays-to-be-loaded-on-call) are subprogram groups (by type and subtype) that are made into overlays by the Core Load Builder. They make it possible for many COBOL core loads that would otherwise not fit into core to be loaded and executed.

If, in constructing a core image program from a main line program, the Core Load Builder determines that the core load will not fit into core, SOCALs are created by the Core Load Builder for the core load. In addition, the LOCAL/SOCAL flipper, which fetches the SOCALs when they are required during execution, is included in the core load along with the area into which the SOCALs are loaded (the SOCAL area).

1. SOCALs are made into overlays by the ....

2. The .... fetches SOCALs when they are needed, and transfers control to them.

3. (LOCAL, SOCAL) creation takes highest priority.

* * *

1. Core Load Builder
2. Flipper
3. LOCAL

---

30. Stated differently, main line program functions are assigned a priority code. If the Core Load Builder determines that the main line program cannot fit into core, two attempts at system overlays are made. If the second attempt still does not make it possible for the core load to fit into core, an error message is printed.

True or false?

1. SOCALs are categorized by priority.

2. Two levels of prioritized overlaying are attempted.

* * *

1. True
2. True

---

31. Each SOCAL group does not contain all the available SOCALable subprograms of the specified types and subtypes. Only those subprograms of the specified types and subtypes required by the core load are contained in the SOCAL.

If a subprogram that would otherwise be included in a SOCAL is specified as a LOCAL subprogram, that subprogram is made a LOCAL and is not included in the SOCAL in which it would ordinarily be found.

SOCALs are never built for core loads in which the main line program is written in Assembler or RPG language.

1. Each SOCAL (contains, does not contain) all possible SOCALable functions.

2. (LOCAL, SOCAL) creation takes precedence.

* * *

1. Does not contain
2. LOCAL

----

32. A rule of prime importance regarding subroutines in the SOCAL scheme is that none must cut across SOCALs. That is, a given subroutine that is in one SOCAL may not call a subroutine that is in another SOCAL or cause another SOCAL to be brought into core before the execution of the given subroutine is completed. It should also be noted that disk I/O is used every time a SOCAL (or a LOCAL) is brought into core. This means that disk I/O will sometimes be entered without the user's direct knowledge.

When writing or modifying a program that is known to require SOCALs, planning is required to minimize the flipping of the various SOCALs in and out of core during execution. Ideally the program should be written in sections, each of which employs a single SOCAL, e.g., input, computation, and output. Even input and output should be carefully planned so as to separate disk and non-disk operations whenever possible.

Match each item with its corresponding factor.

1. SOCALs
2. Disk I/O
3. Program sections
4. SOCALS and LOCALs

a. One must not call another
b. Completely intermixable, since separate areas are used
c. Hidden SOCAL activity is possible
d. Avoid mixing SOCALs

* * *

1. a
2. c
3. d

SUMMARY

In this lesson, you have learned to create program overlays by means of LOCALs - Load on Call program segments. You also learned about the SOCAL, or System Calls, which enable over-sized program segmentation without programmer intervention. The capabilities described herein greatly expand the power of the 1130 Computing System.

This has been the last lesson in the 1130 COBOL Programmed Instruction Manual. As you use the language, you should actively refer to all manuals available for reference, until you are completely familiar with their contents.

END OF LESSON 43

EXAMINATION

EXAMINATION

1. A record name is specified in:

   1. an OPEN statement.

   2. a READ statement.

   3. a PERFORM statement.

   4. a SELECT statement.

   5. a WRITE statement.

---

2. A file name is specified in:

   1. a MOVE statement.

   2. the Working-Storage Section.

   3. a WRITE statement.

   4. a READ statement.

   5. a COMPUTE statement.

---

3. An FD entry is a part of the:

   1. File Section.

   2. FILE-CONTROL paragraph.

   3. Environment Division.

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..

       A   B
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-1130.
       OBJECT-COMPUTER.  IBM-1130.
       SPECIAL-NAMES.  C04 IS TO-COMMENT.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT INCOME-FILE
               ASSIGN TO RD-1442.
```

In the Environment Division segment above:

1. SELECT should begin in Area A.

2. the SPECIAL-NAMES paragraph should be in another division.

3. FILE SECTION should replace INPUT-OUTPUT SECTION.

4. None of these

---

5. A PICTURE clause that specifies a numeric variable is:

   1. PIC $$9.99.
   2. PIC 99V99.
   3. PIC 9.9.
   4. PIC 9,999V9.

---

6. Alignment of a decimal point is caused by:

   1. only the picture character V
   2. only the picture character.
   3. both the configuration of the picture characters and V
   4. neither the picture character nor V

---

7. The picture that would allow printing of the variable $11,200.05 is: (Choose the picture most likely to be used in a real-life situation.)

   1. $99,999.00
   2. $$$,$$$V.$$
   3. $$$,999.99
   4. $$$,$$$.99

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..

           IF MARITAL-STATUS NOT EQUAL TO 'S' GO TO MARRIED.
           IF DEPENDENTS NOT EQUAL TO 1
               GO TO SPECIAL-CASE.
```

The coding above represents a:

   1. series of two simple IF statements.
   2. compound condition.
   3. a single IF sentence.

---

9. In order to find the average sales per month for the first three months of the year, you could write:

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..

           COMPUTE AVERAGE =
               JAN + FEB + MARCH / 3.
```

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..

       COMPUTE (JAN + FEB + MAR) / 3
           = AVERAGE.
```

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..

       ADD JAN FEB MAR GIVING TOTAL.
       DIVIDE 3 INTO TOTAL GIVING
           AVERAGE.
```

---

10. One of two input files contains a record of data relating to each item stocked. The other file contains a record of data for each item stocked that has increased in price. In order to incorporate the price changes into the complete file, you would use the technique of:

1. matching records.

2. channel skipping.

3. horizontal spacing.

4. sorting file.

---

11. To transfer control to a paragraph in a program and then return control to the statement directly following the point of transfer, you would write:

1. a GO TO statement.

2. two PERFORM statements.

3. a relational condition.

4. a PERFORM statement.

---

12. Before writing record variables called PRINT-RECORD into a file called PRINTER, you must be sure that the computer has executed the 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..

           OPEN PRINTER.
```

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..

           OPEN OUTPUT PRINTER.
```

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..

           OPEN OUTPUT PRINT-RECORD.
```

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..

           OPEN PRINTER OUTPUT.
```

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..

           PERFORM TOTAL.
```

In the statement above a restriction on TOTAL is that it must not contain a:

1. An IF statement.

2. PERFORM statement.

3. GO TO statement that refers to a procedure-name outside TOTAL.

4. None of these.

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..

           GO TO TOTAL.
```

In this statement a restriction on TOTAL is that it must not contain a:

1. An IF statement.

2. PERFORM statement.

3. GO TO statement.

4. None of 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..

101     IDENTIFICATION DIVISION.
102     PROGRAM-ID. TEST-PROGRAM.
103     ENVIRONMENT DIVISION.
104     CONFIGURATION SECTION.
105     SOURCE-COMPUTER.  IBM-1130.
106     OBJECT-COMPUTER.  IBM-1130.
107     SPECIAL-NAMES.  C02 IS TO-TAX.
108     INPUT-OUTPUT SECTION.
109     FILE-CONTROL.
110         SELECT MASTER-FILE
111             ASSIGN TO DF-10-600-X.
112         SELECT OUT-FILE
113             ASSIGN TO PR-1403-C.
114         SELECT PAY-FILE
115             ASSIGN TO PR-1403-G.
116             RESERVE NO ALTERNATE AREA.
117     DATA DIVISION.
118     FILE SELECTION.
119     FD  MASTER-FILE.
120         LABEL RECORDS ARE STANDARD.
201         BLOCK CONTAINS 6 RECORDS.
202     01  MASTER-RECORD.
203         02 ID-NUMBER PIC X(4).
204            88 LOCAL VALUES ARE 0000
205                 THRU 2000.
206            88 OUT-OF-TOWN VALUES ARE
207                 2001 THRU 4000.
208         02 TEST-VALUE PIC 999.
        .
        .
        .
301     WORKING-STORAGE SECTION.
302     77  SAVE-CONTROL PIC X(4).
303     01  HEADING-RECORD.
        .
        .
        .
```

The next six questions refer to the program segment above.

---

15. The line that states how records are grouped on disk is:

1. 111

2. 201

3. 208

---

16. The line that makes it possible to use a mnemonic name in the AFTER ADVANCING option is:

1. 107

2. 116

3. 115

---

17. The line that links a file to a disk device is:

1. 119

2. 113

3. 111

---

18. The line that associates a record with a file is:

1. 201

2. 303

3. 202

---

19. The line that specifies an independent elementary variable is:

1. 302

2. 203

3. 206

---

20. The line that specifies a condition name is:

1. 302

2. 208

3. 206

---

21. Which of the statements below can change the normal flow of control within a COBOL program:

1. an ALTER statement.
2. a PERFORM statement.
3. an EXIT statement.
4. a STOP statement.
5. a GO TO statement.
6. all of the above.

---

22. A file of customer records is in ascending sequence by identification number. The file does not contain a record for every possible identification number because some customers have moved or have closed their accounts. In preparing a listing of customers who have had active accounts since 1966, the paragraph LIST-THE-CUSTOMER is to be executed for each record with an identification number smaller than 721. The number of customers who fall into this category is already stored in the variable NUMBER-OF-CUSTOMERS. Which of the following statements could be used to produce the desired number of executions of LIST-OF-CUSTOMERS?

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 LIST-THE-CUSTOMER
                NUMBER-OF-CUSTOMERS TIMES.
```

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..

            PERFORM LIST-THE-CUSTOMER
                UNTIL IDENTIFICATION-NUMBER
                IS GREATER THAN 720.
```

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..

            PERFORM LIST-THE-CUSTOMER
                VARYING COUNTER FROM 1
                BY 1 UNTIL COUNTER
                IS GREATER THAN
                NUMBER-OF-CUSTOMERS.
```

4. All of these
5. None of these

---

23. a GO TO statement that will be changed within a program with an ALTER statement:

    1. may contain the DEPENDING ON option.

    2. must be the only statement in a paragraph.

    3. must not include a paragraph name.

    4. All of these

    5. None of these

---

24. In a normal alphanumeric move, the data is aligned at the:

    1. right of the receiving variable.

    2. left of the receiving variable.

    3. decimal point position of the receiving variable.

    4. None of these

---

25. The USAGE clause:

    1. Determines the manner in which a data item is represented in core storage.

    2. May be omitted from a Program.

    3. Neither a nor b.

    4. Both a and b.

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..

        01 PERCENT-TABLE USAGE IS COMP.
            02 PERCENT OCCURS 3 TIMES PIC V99.
```

    1. PERCENT-TABLE will have two elements.

    2. PERCENT-TABLE will have three elements.

    3. Each element of 01 PERCENT-TABLE will have 3 digits.

---

27. Which picture could produce the printed results ***15.21- and 1,928.43+ from source values -1521 and +192843?

    1. *,***.99-

    2. ***99.99+

    3. *,*99.99+

    4. *,*99.99-

---

28. Which picture could produce the printed results ƀƀƀ9ƀ7ƀ6 and ƀƀ5ƀ4ƀ3ƀ2 from source values 00976 and 05432?

    1. ZBB9B9B9

    2. 9B9B9B9B9

    3. ZZZB9B9B9

    4. ZB9B9B9B9

---

29. Which picture could produce the printed result 1700ƀCR if the source value, 17, were negative?

    1. 9999OCR

    2. **00BCR

    3. 99000CR

    4. 9999BCR

---

30. A subscript for a table element may be:

    1. an integer.

    2. a variable that has an integer value.

    3. an index.

    4. Any of these

---

31. The value 3 could be assigned to the index QUEST by execution of the 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..

        MOVE 3 TO QUEST.
```

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..

        COMPUTE QUEST = 3.
```

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..

        SET QUEST TO 3.
```

----

32. The index YEAR could be associated with the table STATISTICS by:

1. an IF statement.

2. a PERFORM statement with the VARYING option.

3. an OCCURS clause with the INDEXED BY option.

4. including the table in a random file.

----

33. For creation, a sequential file must have:

1. random access and be opened OUTPUT.

2. sequential access and be opened OUTPUT.

3. random access and be opened I-O.

4. sequential access and be opened I-O.

----

34. The system-name DF-10-600-X is required for:

1. Card reading.

2. Card punching.

3. Printing.

4. Mass-storage files.

----

35. The statement form that would be used to write a record to a sequential disk file is:

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
```

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
               INVALID KEY imperative-statement.
```

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 record-name
               AT END IMPERATIVE 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..

           WRITE record-name
               AT EOP imperative-statement.
```

----

36. The statement that would be used to place an updated record back into a random file is:

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.
```

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
               AT END imperative-statement.
```

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 record-name
               AT EOP imperative-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..

           WRITE record-name
               INVALID KEY imperative-statement.
```

----

37. The ACCESS clause of the FILE-CONTROL paragraph must be used in the SELECT whenever:

1. sequential access will be needed from a mass-storage device.

2. sequential access will be needed from a card-reader.

3. the file will be opened as OUTPUT.

4. the file will be randomly accessed.

----

38. The ACTUAL KEY clause of the FILE-CONTROL paragraph must be used in the SELECT entry for:

1. card files.

2. all sequentially accessed files.

3. some sequentially accessed files but no randomly accessed indexed files.

4. all randomly accessed files.

----

39. The variable specified in the ACTUAL KEY clause of a random file must be:

1. set to the value of the record sequence number to be accessed.

2. a numeric variable.

3. a working-storage variable.

4. defined within the record associated with the random file.

40.

```
0   0   1   1   2   2   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 PAYROLL
                    ASSIGN TO DF-10-900-X FOR MULTIPLE UNIT.
```

The above entry will be:

1. Entered for random files.

2. Entered for card files.

3. Entered for random files where the file extends to more than one disk.

4. Entered for sequential files where the file extends to more than one disk.

END OF EXAMINATION

ADVISOR GUIDE

IBM 1130 COBOL PROGRAM PRODUCT - PROGRAM INSTRUCTION MANUAL

ADVISOR GUIDE

To the Advisor

Each lesson has been kept to a minimum length, in order that it may be completed within one sitting requiring usually not more than one hour. The largest part of this manual has been devoted to a thorough coverage of ANSI COBOL language features. A major portion of that has been dedicated to the disk, since on the 1130, disk usuage is of utmost importance.

In addition to pure language features and disk usuage, a number of lessons at the end have been provided specifically oriented to the 1130 itself. Minimum instruction in the use of the 1130 Disk Monitor, and the control cards necessary for the compilation and execution of the COBOL programs has been given. Special attention has been paid to the use of CALLable subprograms, including the use of file I/O within COBOL language subprograms.

This manual is not intended to replace the standard manuals provided with the COBOL compiler. It is intended to insure that language features and computer operational fundamentals are presented to the user in a controlled sequence, and at a rate which can be easily learned. Once basic instruction in the language and machine characteristics has been learned, it is expected that the normal manual would be of assistance in providing the remaining details which may not be included in the lessons, and which are of benefit normally only after considerable hands-on experience has been obtained.

Following completion of this course, the student is advised to participate in an 1130 Workshop of approximate five day length, dedicated to providing COBOL coding experience. Should such a workshop not be available locally, the student at a minimum should review the workshop handouts, and work the problems, compiling and executing these until fully debuged. Only through actual experience on the computer, with problems not too complex for his present capabilities, can a student expect to proceed to full competence in the COBOL language.

How is the compiler obtained?

The 1130 COBOL compiler is a program product, leased to IBM customers at a charge of $75.00 per month. It may be ordered from any Data Processing Sales Representative, under order number 5711-CB1.

What features are included in the language?

1130 COBOL is a subset of the full COBOL defined by the American National Standards Institute, formally known as United States of America Standards Institute (USASI). ANSI COBOL is composed of standard features, and extensions to the language. Only standard features are normally implemented identical accross compilers produced for various computer manufacturers. In the 1130 COBOL Language Specifications Manual (Form No. SH20-0816), the precise language features included in the 1130 COBOL subset are defined. In addition, IBM extensions to the standard language are identified through halftone shading.

What materials are required to be used?

The Programmed Instruction Manual is comprised of 43 lessons. Only a single manual is required, since all illustrations are integrated fully into each lesson. In addition to the PI materials, a full set of 1130 COBOL manuals should be available, including the language specifications manual mentioned above, plus the 1130 COBOL Operations Manual (Form No. SH20-0927), and the 1130 COBOL Programmers Guide (Form No. SH20-0928). Only the language specifications manual will normallly be required until the final six lessons, when specific 1130 oriented subjects are addressed apart from the COBOL language.

Are these lessons suitable for programmers who have previously used COBOL for other computer systems?

If a programmer has experience in another version of COBOL, he should review the solutions for problem assignments at the end of the lessons. The student should also attempt to code solutions for some problems before reviewing the text solutions. If unfamiliar with too many statements and rules, the student should be advised to begin the manual at Lesson 1 and proceed at a comfortable pace.

How are students graded?

A multiple-choice test has been constructed to assist in student evaluation. At best a multiple-choice quiz can help you determine what a student knows about COBOL. It is difficult, if not impossible, to determine whether a student can code an ANSI COBOL program through interpretation of results from a multiple-choice test. Convenience in administration and a minimum criterion accomplishment are the main reasons such a test is provided. The primary criterion for judging a student's success is his ability to code solutions to the problem assignments given in the course. The student is advised to contact his advisor if, after attempting to code his solution and reviewing the text solution, he still does not understand the concepts, rules, or techniques involved. Your primary role as advisor should be to resolve any problems of this nature while the student is taking the course. If a student successfully codes the problems assigned in this course, he should function effectively in an ANSI COBOL installation.

In what time period should the student be expected to finish?

Study time for the lessons will require about 34 and one half hours for most students. A minimum of two weeks and a maximum of five weeks should be used as a gauge to establish a schedule for prospective students. A convenient method to plan a student's schedule is to determine the number of lessons to be completed each day, with six per day maximum. Programmed Instruction is not intended to be studied eight hours a day, and students should be advised of this before beginning the course. Gaps are likely to occur in the student's schedule, but interruptions of a week or more should not be allowed in the study program. A student cannot be expected to retain previously acquired skills without some immediate use through further study; this is especially true in the beginning. Each lesson has an estimated required study time at the beginning, but these timings will vary greatly with different people. Given correct information and advice before beginning the course, the student should have no problems maintaining a well-planned study schedule.

What is the main strategy of the manual?

The structure of the lessons is based on a set of commercially oriented problems. The student begins by coding COBOL solutions to transmit data to and from the console typewriter, progressing to problems requiring card input to produce printer listings. He then progresses to disk-oriented solutions that incorporate logic functions of matching records, control breaks on group levels, and the calculation capabilities required to solve each problem. Study topics are introduced as required for the student to proceed successfully through the problem set. Some lessons are topical in nature, covering efficient data formats, data manipulation, conditional statements, and the structure and searching of data tables. Later lessons require the student to code COBOL solutions to create, maintain, and access Sequential and Random Disk files. Finally, the student learns those elements of the 1130 Monitor System relevant to COBOL usage.

The primary intent of the manual is to place the student in a problem-solving environment with American National Standard Institute COBOL as his major tool. It attempts to simulate the daily working environment of a COBOL programmer.

Who should study these materials?

People who have a need to write ANSI COBOL programs or who will play a role in planning, implementing, and reviewing applications in a COBOL installation should study this manual.

Do the lessons teach beginners how to solve computer problems?

No, they are "language" lessons rather than "programming" lessons. Before a beginner starts on this manual, he must be trained in programming techniques -- how to define and analyze jobs, system and program flowcharting, and the nature of programming systems, to name some of the most important topics. In addition, students are expected to know a little about the IBM 1130. If a student does not posses the preceding talents, he may wish to take the Computing System Fundamentals PI. The beginner may also find the classroom course Introduction to Small Binary Computers helpful. In certain sections of the COBOL course, the student is instructed to seek advisor assistance if he has a problem performing an assumed skill. Examples of this are flowchart construction or sign notation in hexadecimal.

What types of students may be studying this manual?

This programmed instruction manual is designed for students with a variety of backgrounds and aptitudes. Familiarity with the lesson structure and strategy and an understanding of a student's prerequisite skills and experience will enable you to best advise him on how to study the manual. Someone who possesses only minimum prerequisites and has no coding experience in another language should follow all instructions in the course and perform all required coding, whereas a student with experience in one or more programming languages may be able to proceed through the manual very rapidly.

All students should code the problem assignments at the end of the lessons to be sure they can do the thing being taught. If, during the lessons, a student indicates a lack of prerequisite skill (e.g., cannot read or construct a flowchart) and this hinders progress, the prerequisite should be acquired through appropriate education before continuing the COBOL PI. Successful administration will be greatly enhanced through your understanding of a student's entering behavior in relation to the structure and strategy of the manual.

What is the next step in a student's education after completion of these lessons?

Upon completion of the final lesson, the student should have many coding skills in American National Standard Institute COBOL. However, there are no instructions provided on how to debug a source program using compiler diagnostics or to execution test any programs. Also, since the ANSI COBOL compiler is run under an operating system, some knowledge of the monitor control language may be required, depending upon an installation's procedures. These skill requirements are introduced in the last six lessons in the course.

ANSWERS TO THE STUDENT EXAMINATION

1 - 5
2 - 4
3 - 1
4 - 4
5 - 2
6 - 3
7 - 4
8 - 1
9 - 3
10 - 1
11 - 4
12 - 2
13 - 3
14 - 4
15 - 2
16 - 1
17 - 3
18 - 3
19 - 1
20 - 3
21 - 5
22 - 4
23 - 2
24 - 2
25 - 4
26 - 2
27 - 3
28 - 3
29 - 2
30 - 4
31 - 3
32 - 3
33 - 2
34 - 4
35 - 2
36 - 4
37 - 4
38 - 4
39 - 4
40 - 4

END OF ADVISOR GUIDE

READER'S COMMENT FORM

1130 COBOL Text Volume II SH20-0930-0

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM.

COMMENTS

Reply Requested

Yes □

No □

Name ______________________________

Job Title ______________________________

Address ______________________________

______________________ Zip __________

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

FOLD ON TWO LINES, STAPLE AND MAIL.

YOUR COMMENTS PLEASE

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

LD FOLD

FIRST CLASS
PERMIT NO. 142
POUGHKEEPSIE, NEW YORK

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY:

IBM Corporation
Education Center, Bldg. 005
South Road
Poughkeepsie, New York 12602

TENTION: Education Development - Publications Services, Dept. 78L

LD FOLD

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 II Printed in U. S. A. SH 20-0930-0