8 OUTPUT DESIGN AND FILE PROCESSING

8.1 Output Formatting

The print statement we have been using in the previous chapters is a list-directed output statement. In list-directed output, the output list determines the precise appearance of printed output. In other words, we have no control over the format of the output. To control the manner in which the output is printed or to produce an output in a more readable form, we use **FORMAT** statements. To use a **FORMAT** statement, we must modify the **PRINT** statement by replacing the '*' with a **FORMAT** statement label. The general form of a formatted **PRINT** statement is

```
      PRINT K, expression list
```

The **FORMAT** statement number **k** identifies a format to be used by the print statement. The statement number can be any positive **INTEGER** constant up to five digits. Recall that statement numbers are placed in columns 1 through 5. The *expression list* specifies the value(s) to be printed. The general form of the **FORMAT** statement is

```
K     FORMAT(specification list)
```

A **FORMAT** statement is a non-executable statement. It can appear anywhere in the program before or after the associated print statement. The *specification list* in the **FORMAT** statement specifies both the vertical spacing and the horizontal spacing to be used when printing an output. The first character of the specification list, called the *carriage control character*, is used to control the vertical spacing. The rest of the specification list consists of various format specifications and controls the horizontal spacing.

FORTRAN provides format specifications for blank spaces, integer, real, character and logical types. Commas are used to separate specifications in the specification list. Before printing the line, the computer constructs each output line internally in a memory area called the *output buffer*. The length of each line in the buffer is 133 characters. The first character is used to control the vertical spacing and the remaining 132 characters represent the line to be printed. The buffer is filled with blanks before it is used to construct an output line.

The following are some of the carriage control characters used to control the vertical spacing:

- `' '`: single spacing (start printing at the next line)
- `'0'`: double spacing (skip one line then start printing)

- `'-'`: triple spacing (skip 2 lines then start printing)
- `'1'`: new page (move to the top of the next page before printing)
- `'+'`: no vertical spacing (start printing at the beginning of the current line irrespective of what was printed before)

The six format specifications presented below allow the control of horizontal spacing. In the following sections we will use

```
....+....1....+....2....+....3....+....4.
```

as a header to the output to indicate the horizontal spacing, Notes that the above line is not part of the output.

8.1.1 I Specification

The **I** specification is used to print integer expressions. The general form of I specification is **{Iw}**, where **w** is a positive integer representing the number of positions to be used to print the integer value. To find the minimum number of positions necessary to print a number, we count the number of digits in the integer including the minus sign. For example, if we want to print -25, the value of **w** should be at least 3. In the case where the value of **w** is more than 3, the number -25 is printed right-justified. If the value of **w** is less than 3, the number -25 cannot be printed and asterisk (*) characters appear in the output. In this case, the number of asterisks is equal to **w**.

In other words, to print an integer number using I specification, we start filling the positions from right to left. The extra positions to the left of the integer (if any) will be filled with blanks. If the positions are not enough to represent the number, the positions are filled with asterisks indicating that the specification is not enough to print the integer number.

Example 1: What is the minimum I specification needed to print each of the following integers?

345, 67, -57, 1000, 123456

Solution:

Number	I specification
345	I3
67	I2
-57	I3
1000	I4
123456	I6

Example 2: *What will be printed by the following program?*

```
      INTEGER M
      M = -356
      PRINT 10, M
10    FORMAT(' ',  I4)
      END
```

Solution:

```
....+....1....+....2
-356
```

Notice that the carriage control character ' ' did not appear in the output. This characters indicates that the output line is single spacing.

Example 3: *If the **FORMAT** statement in the previous example is modified as follows:*

```
      FORMAT('1', I6)
```

What will be printed?

Solution:

The printed output in this case will start on a *new page*, because of the carriage control character '1':

(new page)

```
....+....1....+....2....+....3....+....4.
-356
```

Example 4: *If the **FORMAT** statement in the previous example is modified as follows:*

```
      FORMAT('-', I3)
```

What will be printed?

Solution:

```
....+....1....+....2....+....3....+....4.


***
```

Notice that the printed output in this case has two empty lines before the data. The reason is the carriage control character '-' which means triple spacing. Moreover, the data is printed as three asterisks because the format specification I3 is not enough for the number -356.

Example 5: *Assume K = -244 and M = 12. The following **PRINT** statements will produce the shown outputs.*

```
a.    PRINT 10,  K
10    FORMAT(' ', I4)
```

```
....+....1....+....2....+....3....+....4.
-244
```

```
b.    PRINT 20,  K,  M
20    FORMAT(' ', I5, I6)
```

```
....+....1....+....2....+....3....+....4.
 -244     12
```

```
c.    PRINT 30,  K
      PRINT 35,  M
30    FORMAT(' ', I3)
35    FORMAT('0', I2)
```

```
....+....1....+....2....+....3....+....4.
***

12
```

```
d.    PRINT 40,  K + M
40    FORMAT(' ', I5)
```

```
....+....1....+....2....+....3....+....4.
 -232
```

```
e.    PRINT 50,  K / M
50    FORMAT(' ', I3)
```

```
....+....1....+....2....+....3....+....4.
-20
```

```
f.    PRINT 60,  M + 1.0
60    FORMAT(' ', I3)
```

```
ERROR MESSAGE: TYPE MISMATCH
```

```
g.    PRINT 70,  -345
70    FORMAT(' ', I7)
```

```
....+....1....+....2....+....3....+....4.
   -345
```

```
h.    PRINT 80,  -39 / 3 * 2
80    FORMAT(' ', I3)
```

```
....+....1....+....2....+....3....+....4.
-26
```

```
i.    PRINT 90, K
      PRINT 95, M
90    FORMAT(' ', I4)
95    FORMAT('+', I8)
```

```
....+....1....+....2....+....3....+....4.
-244  12
```

```
j.    PRINT 98, K
      PRINT 98, M
98    FORMAT(' ', I4)
```

```
....+....1....+....2....+....3....+....4.
-244
  12
```

8.1.2 F Specification

The **F** specification is used to print real values. The general form of the **F** specification is **{Fw.d}**, where **w** is a positive integer representing the total number of positions to be used to print the real number and **d** represents the number of positions to be used to print the fractional part of the real number. Note that **w** must satisfy the relation $w \geq d + 1$.

To find the number of positions needed to print a real number, we count the number of significant digits in the real number including the decimal point and the minus sign. For example, if we want to print -91.35, we need a total of six positions, two of them to the right of the decimal point, so the specification should be at least F6.2. To print the real number, we count from right to left **d** positions and place the decimal point at position **d+1**. We start placing the integer part of the real number from right to left and the fractional part of the real number from left to right. The extra positions to the left of the decimal point (if any) are filled with blanks, while the extra positions to the right of the decimal point (if any) are filled with zeros. If the number of positions to the left of the decimal point is not enough to represent the integer part of the real number, all **w** positions are filled with asterisks. If the number of positions to the right of the decimal

point is not enough to represent the fractional part of the real number, the number will be rounded to just fill the specified number of decimal positions.

Example 1: *What is the minimum F specification needed to print the following real numbers?:*

823.67509, 0.002, .05, -.05, -0.0008

Solution:

Number	F specification
823.67509	F9.5
0.002	F5.3
.05	F3.2
-.05	F4.2
98.	F3.0
98.0	F4.1
-0.0008	F7.4

Example 2: *What will be printed by the following program?*

```
      REAL X
      X = 31.286
      PRINT 10, X
10    FORMAT('1',  F6.3)
      END
```

Solution:

The printed output on *a new page* is as follows:

```
....+....1....+....2....+....3....+....4.
31.286
```

Example 3: *If the* ***FORMAT*** *statement in the previous example is modified as follows:*

```
      FORMAT(' ', F8.3)
```

What will be printed?

Solution:

```
....+....1....+....2....+....3....+....4.
  31.286
```

Example 4: *If the* ***FORMAT*** *statement in the previous example is modified as follows:*

```
      FORMAT(' ', F8.4)
```

What will be printed?

Solution:

```
....+....1....+....2....+....3....+....4.
 31.2860
```

Example 5: *If the* ***FORMAT*** *statement in the previous example is modified as follows:*

```
      FORMAT('   ', F5.3)
```

What will be printed?

Solution:

```
....+....1....+....2....+....3....+....4.
*****
```

Example 6: *If the **FORMAT** statement in the previous example is modified as follows:*

```
      FORMAT('   ', F6.2)
```

What will be printed?

Solution:

```
....+....1....+....2....+....3....+....4.
31.29
```

Example 7: *Assume X = -366.126, Y = 6.0 and Z = 20.97. The following **PRINT** statements will produce the shown outputs.*

```
a.    PRINT 10, X
10    FORMAT(' ', F11.5)
```

```
....+....1....+....2....+....3....+....4.
 -366.12600
```

```
b.    PRINT 20, X
20    FORMAT(' ', F8.3)
```

```
....+....1....+....2....+....3....+....4.
-366.126
```

```
c.    PRINT 30, Z
      PRINT 35, Y
30    FORMAT(' ', F4.1)
35    FORMAT('0', F4.2)
```

```
....+....1....+....2....+....3....+....4.
21.0

6.00
```

```
d.    PRINT 40,  X / Y
40    FORMAT(' ', F7.3)
```

```
....+....1....+....2....+....3....+....4.
-61.210
```

```
e.    PRINT 50,  Y + 0.00001
50    FORMAT(' ', F7.5)
```

```
....+....1....+....2....+....3....+....4.
6.00001
```

```
f.    PRINT 60,  Z - 5
60    FORMAT(' ', F5.2)
```

```
....+....1....+....2....+....3....+....4.
15.97
```

```
g.    PRINT 70,  Z
70    FORMAT('+', I5)
```

```
ERROR MESSAGE: TYPE MISMATCH
```

```
h.    PRINT 80,  -144 / 24 + 35.2
80    FORMAT(' ', F4.1)
```

```
....+....1....+....2....+....3....+....4.
```

```
29.2
```

```
i.     PRINT 85,  Y
       PRINT 85,  Z
85     FORMAT(' ', F6.2)
```

```
....+....1....+....2....+....3....+....4.
  6.00
 20.97
```

```
j.     PRINT 90,  Y
       PRINT 95,  Z
90     FORMAT(' ', F6.2)
95     FORMAT('-', F6.2)
```

```
....+....1....+....2....+....3....+....4.
  6.00


 20.97
```

8.1.3 X Specification

The **X** specification is used to insert blanks between the values we intend to print. The general form of this specification is **nX**, where **n** is a positive integer representing the number of blanks.

Example 1: *The following program:*

```
       REAL A, B
       A = -3.62
       B = 12.5
       PRINT 5, A,  B
5      FORMAT(' ',  F5.2, F4.1)
       END
```

prints the following output:

```
....+....1....+....2....+....3....+....4.
-3.6212.5
```

The output is not readable because the two printed values are not separated by blanks. If we modify the format statement using **X** specification as follows:

```
       FORMAT(' ', F5.2, 3X, F4.1)
```

the output becomes:

```
....+....1....+....2....+....3....+....4.
-3.62 12.5
```

The **X** specification can be used as a carriage control character. The following pairs of **FORMAT** statements print the same output.

```
10     FORMAT(' ', I2)
```

is equivalent to

```
10     FORMAT(1X, I2)
```

and

```
20     FORMAT(' ', 2X, F4.1)
```

is equivalent to

```
20    FORMAT(3X, F4.1)
```

8.1.4 Literal Specification

The literal specification is used to place character strings in a **FORMAT** statement as part of the specification list. The character string must be enclosed between two single quotation marks.

Example 1: *What will be printed by the following program?*

```
      REAL AVG
      AVG = 65.2
      PRINT 5, AVG
5     FORMAT(' ' ,'THE AVERAGE IS = ', F4.1)
      END
```

Solution:

```
....+....1....+....2....+....3....+....4.
THE AVERAGE IS = 65.2
```

Example 2: *The following program prints the message FORTRAN77 on top of a new page.*

```
      PRINT 30
30    FORMAT('1', 'FORTRAN77')
      END
```

The output printed at the *a new page* is:

```
....+....1....+....2....+....3....+....4.
FORTRAN77
```

8.1.5 A Specification

The **A** specification is used to print character expressions. The general form of the **A** specification is **Aw**, where **w** represents the length of the character string. If the string has more than **w** characters, only the left-most **w** characters will appear in the output line. On the other hand, if the string has fewer than **w** characters, its characters are right-justified in the output line with blanks to the left. The integer **w** may be omitted. If **w** is omitted, the number of characters is determined by the length of the character string.

Example 1: *What will be printed by the following program?*

```
      PRINT 55, 'ICS-101'
55    FORMAT(' ',A7)
      END
```

Solution:

```
....+....1....+....2....+....3....+....4.
ICS-101
```

Example 2: *What will be printed by the following program?*

```
      CHARACTER TEXT*5
      TEXT = 'KFUPM'
      PRINT 55, TEXT, TEXT, TEXT
55    FORMAT(' ', A, 3X, A3, 3X, A9)
      END
```

Solution:

```
....+....1....+....2....+....3....+....4.
KFUPM    KFU        KFUPM
```

8.1.6 L Specification

The **L** specification is used to print logical expressions. The general form of **L** specification is **Lw**. The letter T or F is printed if the logical expression is true or false respectively. The printed letter is right-justified.

Example 1: *What will be printed by the following program?*

```
      PRINT 5, .TRUE.
5     FORMAT(' ',L1)
      END
```

Solution:

```
....+....1....+....2....+....3....+....4.
T
```

Example 2: *What will be printed by the following program?*

```
      LOGICAL X,  Y
      X = .TRUE.
      Y = .FALSE.
      PRINT 15, X,  X
15    FORMAT(' ', L1, 2X,  L5)
      PRINT 20, Y,  Y
20    FORMAT(' ', L1, 2X,  L7)
      END
```

Solution:

```
....+....1....+....2....+....3....+....4.
T       T
F         F
```

8.2 Specification Repetition: Another Format Feature

If we have consecutive identical specifications, we can replace them by an integer constant followed by the identical specification(s) to indicate repetition. For example, the specifications: I4, I4, I4 can be replaced by 3I4. Also, the specifications: I2, 3X, I2, 3X, I2, 3X, I2, 3X can be replaced by 4(I2, 3X). The following pairs of **FORMAT** statements illustrate the use of repetition constants:

```
10    FORMAT('0', 3X, I2, 3X, I2)
```

is equivalent to

```
10    FORMAT('0', 2(3X, I2))
```

and

```
20    FORMAT(' ',F5.1, F5.1, F5.1, 5X, I3, 5X, I3, 5X, I3, 5X, I3)
```

is equivalent to

```
20    FORMAT(' ',3F5.1,4(5X, I3))
```

8.3 Carriage Control Specification

The carriage control character is normally specified as the first character in the format specification list. It can be specified as a blank or the characters 0,1,-, +. But in the case where it is not specified as part of the specification list, the first character in the buffer output is taken as the carriage control character. If the first character of the buffer output is one of the carriage control characters (a blank, 0, 1, +, -), then the proper action is taken. If the first character is not among the carriage control characters, then the output is system dependent. The following example illustrates a specification list where carriage control character is missing:

Example:

```
      PRINT 10
10    FORMAT('1995')
      END
```

The output, on *a new page*, would be as follows:

```
....+....1....+....2....+....3....+....4.
995
```

Notice that the first character '1' was considered as a new page carriage control character.

8.4 File Processing

In many applications, the amount of data read and/ or produced is huge. Providing data interactively is not efficient, thus a different way to handle data is needed, namely, files. Another reason for using files comes from the repetitive use of the same data every time the program is run; making the data entry task very tedious. The third reason is that data in many real applications is taken or recorded by instruments or devices then used for analysis and computations.

8.4.1 Opening Files

Before using a file for input or output, it must be prepared for that operation. Files that are used for input must exist prior to their usage. To prepare a file for input, the following **OPEN** statement must precede any read statement from that file:

```
      OPEN(UNIT = INTEGER EXPR, FILE = FILENAME, STATUS = 'OLD')
```

where UNIT equals an integer expression in the range of 0 to 99. Avoid using 5 and 6 as unit numbers since they are already assigned for the keyboard and the screen. The filename is a character string containing the actual name of the file followed by the file extension. In the IBM mainframe, the file name is separated from the file extension by a space and if the extension is omitted, it is assumed to be FILE. Upon opening a file for reading, the reading will take place from the beginning of the file.

Files that are used for output may not exist before being used. If the file does not exist, it will be created whereas if it exists its contents will be erased. To prepare a file for output, the following statement must precede any write statement to that file:

```
OPEN(UNIT = INTEGER EXPR, FILE = FILENAME, STATUS = 'NEW')
```

or

```
OPEN(UNIT = INTEGER EXPR, FILE = FILENAME, STATUS ='UNKNOWN')
```

The second statement is preferred in our system because the first one assumes that the file does not exist and, therefore, if it exists an error occurs.

Example 1: *Assume that you want to use file POINTS DATA as an input file. The following statement will then appear before any read statement from the file:*

```
OPEN(UNIT = 1, FILE = 'POINTS DATA', STATUS = 'OLD')
```

Example 2: *Assume that you want to use file RESULT DATA as an output file. The following statement will then appear before any write statement to the file:*

```
OPEN(UNIT = 1, FILE = 'RESULT DATA', STATUS = 'UNKNOWN')
```

8.4.2 Reading from Files

To read from a file, the file must have been opened. The **READ** statement will be in the following form:

```
READ(UNIT, *) VARIABLE LIST
```

where UNIT is the same value that is used in the open statement. The rules of reading are exactly the same as the ones you have already seen, the only difference being that data is taken from the file.

Example 1: *Find the sum of three exam grades taken from file EXAM DATA.*

Solution:

```
INTEGER EXAM1, EXAM2, EXAM3, SUM
OPEN(UNIT = 10, FILE = 'EXAM DATA', STATUS = 'OLD')
READ(10, *) EXAM1, EXAM2, EXAM3
SUM = EXAM1 + EXAM2 + EXAM3
PRINT*, SUM
END
```

In many cases, the number of data values in a file is not known and we would like to do some calculations on the data values the file contains. For these cases, the read statement will look as follows:

```
READ(UNIT, *, END = NUMBER) VARIABLE LIST
```

where number is the label of the statement where control will be transferred after all the data from the file is read.

Example 2: *Find the average of real numbers that are stored in file NUMS DATA. Assume that we do not know how many values are in the file and that every value is stored on a separate line.*

Solution:

```
      REAL NUM, SUM, AVG
      INTEGER COUNT
      OPEN(UNIT = 12, FILE = 'NUMS DATA', STATUS = 'OLD')
      SUM = 0.0
      COUNT = 0
333   READ(12, *, END = 999) NUM
          SUM = SUM + NUM
          COUNT = COUNT + 1
          GOTO 333
999   AVG = SUM / COUNT
      PRINT*, AVG
      END
```

8.4.3 Writing to Files

To write to a file, the file must have been opened using an **OPEN** statement and the **WRITE** statement must be used in the following form:

```
      WRITE(UNIT, *) EXPRESSION LIST
```

where **UNIT** is the same value that is used in the **OPEN** statement. The rules of writing to a file are exactly the same as those of the print statement. The * in the **WRITE** statement indicates that the output is free formatted. If format is needed, the format statement number is used instead.

Example: *Create an output file CUBES DATA that contains the table of the cubes of integers from 1 to 20 inclusive.*

Solution:

```
      INTEGER NUM
      OPEN(UNIT = 20, FILE = 'CUBES DATA', STATUS = 'UNKNOWN')
      DO 22 NUM = 1, 20
          WRITE(20, *) NUM, NUM**3
22    CONTINUE
      END
```

Format statement could be used with the write statement in the same way it is used with the print statement. The * in the write statement is replaced with the format statement number.

8.4.4 Working with Multiple Files

In any program, more than one file may be open at the same time for either reading or writing. The same unit number that is used in one file should not be used with any other file in the same program. The number of the files that can be open at the same time is limited by the number of units, which is dependent on the computer you are using.

Example: *Create an output file THIRD that contains the values in file FIRST followed by the values in file SECOND. Assume that every line contains one integer number and we do not know how many values are stored in files FIRST and SECOND.*

Solution:

```
      INTEGER NUM
      OPEN(UNIT = 15, FILE = 'FIRST', STATUS = 'OLD')
      OPEN(UNIT = 17, FILE = 'SECOND', STATUS = 'OLD')
      OPEN(UNIT = 19, FILE = 'THIRD', STATUS = 'UNKNOWN')
123   READ(15, *, END = 456) NUM
          WRITE(19, *) NUM
          GOTO 123
456   READ(17, *, END = 789) NUM
          WRITE(19, *) NUM
          GOTO 456
789   STOP
      END
```

8.4.5 Closing Files

After using a file in our program, that file must be closed. The operating system of the computer we are using normally closes all the files that are open at the end of the program execution. But in some cases, we may need to read the data in the file more than one time. This can be done by closing the file after we finish reading from it and then re-opening the file to read the same data again. We may also need to read from files that were created by our program. This is achieved by closing the file as an output file then re-opening it as an input file. The **CLOSE** statement looks as follows:

```
      CLOSE(UNIT)
```

where unit is the same value that is used in the open statement. You can only close files that are already open.

8.4.6 Rewinding Files

After reading from the file the reading head moves forward towards the end of the file. In certain situations, we may need to restart reading from the beginning of the file which is done by closing the file then re-opening it again. Another method of doing the same thing is through the **REWIND** statement:

```
      REWIND(UNIT)
```

where unit is the same value that is used in the open statement. You can rewind files that are open for reading only.

8.5 Exercises

8.5.1 Exercises on Output Design

1. What will be printed by each of the following programs?

```
1.    REAL X
      X = 123.8367
      PRINT 10, X,  X,  X
10    FORMAT(' ', F7.2, 2X, F6.2, F9.5)
      END
```

```
2.    INTEGER J, K, N
      K = 123
      J = 456
      N = 789
      PRINT 10, K
      PRINT 11, J
      PRINT 12, N
10    FORMAT(' ', I3)
11    FORMAT('+', 3X, I3)
12    FORMAT('+', 6X, I3)
      END
```

```
3.    REAL X1,  X2
      INTEGER N1,  N2
      READ*, X1,  X2
      READ*, N1,  N2
      PRINT 10, X1,  X2
      PRINT 11, N1,  N2
      PRINT 12, X1/X2
10    FORMAT('1',F5.2,  2X,  F3.1)
11    FORMAT('0', I3,  2X,  I2)
12    FORMAT('+', 12X,  F6.2)
      END
```

Assume the input for the above program is:

```
81.6   9.2
-125   48
```

```
4.    PRINT 20, -35,  0.0,  12 * 10.0,  125 / 5
20    FORMAT(1X, I3, '+', F3.1, 'IS NOT EQUAL', F6.1,'-',I2)
      END
```

```
5.    LOGICAL FLAG,  P,  Q
      READ*, P,  Q
      FLAG = .NOT. P .AND. .NOT. Q
      PRINT 33, P,  'AND', Q
      PRINT 44, P .OR. Q,  FLAG
33    FORMAT(' ', L2, 2X,  A,  L3)
44    FORMAT('-', L1, 2X,  L1)
      END
```

Assume the input for the above program is:

```
T   F
```

```
6.    REAL X, Y
      INTEGER N
      X = 25.0
      Y = -35.0
      N = -35
      PRINT 40, X,  SQRT(X)
      PRINT 50, Y,  ABS(Y)
      PRINT 60, N,  ABS(N)
40    FORMAT(' ', 'X=', 2X, F4.1, 2X, 'SQUARE ROOT = ', F4.1)
50    FORMAT(' ', 'Y=', 2X, F5.1, 2X, 'ABSOLUTE VALUE = ',F5.1)
60    FORMAT(' ', 'N=', 2X, I3, 2X, 'ABSOLUTE VALUE = ', I2)
      END
```

```
7.    CHARACTER*6 CITY
      CITY = 'RIYADH'
      PRINT 1, 'THE CAPITAL IS', 2X, CITY
1     FORMAT(' ', A,  2X,  A4)
      END
```

```
8.      INTEGER ARR(5), K
        READ*, ( ARR(K), K = 1,  5)
        DO 70 K = 1,  5
        PRINT 10, ARR(K)
70      CONTINUE
10      FORMAT(' ', I4)
        END
```

Assume the input for the above program is:

```
10   20   30   40   50
```

```
9.      INTEGER ARR(5), K
        READ*, ( ARR(K), K = 1,  5)
        PRINT 10, ( ARR(K), K = 1,  5)
10      FORMAT(' ', 5I2)
        END
```

Assume the input for the program is:

```
10   20   30   40   50
```

```
10.     INTEGER ARR(5), K
        READ*, ( ARR(K), K = 1,  5)
        PRINT 10, ( ARR(K), K = 1,  5)
10      FORMAT(' ', 5(I2,2X))
        END
```

Assume the input for the program is:

```
10   20   30   40   50
```

```
11.     REAL MAT(2,3), I, J
        READ*,(( MAT(I, J), I=1,2),J=1,3)
        DO 10 I= 1,  2
            PRINT 55, (MAT(I, J), J=1,3)
10      CONTINUE
55      FORMAT(' ', 3( F4.1, 2X))
        END
```

Assume the input for the program is:

```
10   20   30   40   50   60
```

```
12.     REAL A(30), B(30), DOT, Z
        INTEGER K, N
        READ*, N,  (A(K), B(K), K=1,  N)
        Z = DOT(N, A, B)
        PRINT 10, Z
10      FORMAT('1', 'DOT PRODUCT = ', F5.1)
        END
        REAL FUNCTION DOT(M, X, Y)
        INTEGER M, I
        REAL X(M),Y(M), SUM
        SUM = 0.0
        DO 123 I = 1,  M
            SUM = SUM + X(I)* Y(I)
123     CONTINUE
        DOT = SUM
        RETURN
        END
```

Assume the input for the program is:

```
4   1   2   3   4   5   6   7   8
```

```
13.    INTEGER N1, N2
       REAL S1, S2
       READ*, N1, N2
       READ*, S1
       READ*, S2
       READ*, N1
1      FORMAT('0', I4, '+', I2, 2X, '=', I4)
2      FORMAT(' ', A, 3X, F5.2)
3      FORMAT('+', 7X, F10.2)
       PRINT 1, N1, N2, N1+N2
       PRINT 2, 'S1', S1
       PRINT 3, S2
       END
```

Assume the input for the program is:

```
37
101   4113   25.0
-30.459   210.0
427.5   48
23
```

2. Indicate the validity of the following statements:
   1. The **FORMAT** statement can be placed anywhere between the declaration statements and the **END** statement of a FORTRAN77 program.
   2. Two or more **PRINT** statements can refer to the same format statement. For example, if X and Y are real variables then the following program segment:

```
       PRINT 5, X
       PRINT 5, Y
5      FORMAT(4X, F5.2)
```

   is correct.

3. Complete the following programs in order to get the required outputs:

```
1.     REAL X
       X = 5.98
       PRINT 1, X
       PRINT 2, X
1      FORMAT(                          )
2      FORMAT(                          )
       END
```

The required output is:

```
....+....1....+....2....+....3....+....4.
    X=5.980   X=6.0
```

```
2.     INTEGER B
       REAL A, C
       A = 3.1
       B = 12.5
       C = 127.66
       PRINT 1520, A, B, C
1520   FORMAT(                              )
       END
```

The required output is:

```
....+....1....+....2....+....3....+....4.
  3.10  12 127.7
```

```
3.    REAL A,
      INTEGER J
      A = -5.62705
      J = 23
      PRINT 5, A, J
5     FORMAT(                          )
      END
```

The required output is:

```
....+....1....+....2....+....3....+....4.
         -5.63     23
```

```
4.    INTEGER Z
      REAL X, Y
      X = 5.00
      Y = 59.996
      Z = 3125
      PRINT 5, X,  Y, Z
5     FORMAT(                                  )
      END
```

The required output is:

```
....+....1....+....2....+....3....+....4.
X= 5.00 Y=  60.00  Z= ***
```

```
5.    PRINT 1, 'FORTRAN'
      PRINT 2, 'I LIKE'
1     FORMAT(                                 )
2     FORMAT(                                 )
      END
```

```
THE REQUIRED OUTPUT IS:
....+....1....+....2....+....3....+....4.
 I LIKE FORTRAN
```

```
6.    INTEGER Y
      REAL X
      X = -20.2451
      Y = 25
      PRINT 6, X, 'AND', Y
6     FORMAT(                                )
      END
```

The required output is:

```
....+....1....+....2....+....3....+....4.
    -20.25   AND   25
```

4. Write a program segment to print the heading "FORTRAN-77--LANGUAGE" centered at the top of a new page. assume the output line contains 80 characters.

5. Write a program that reads any real number, separates the integer and real parts of the number and prints it in the format shown below. For example, if the input is as follows:

```
123.45
```

your formatted output should be as follows:

```
....+....1....+....2....+....3....+....4.
123.450=123+0.450
```

6. Consider the following program

```
      INTEGER X
      REAL Y
      X = 469
      Y = 17.38
      PRINT2, X, Y
2     FORMAT(                    )
      END
```

Given the following format statements below:

```
a.    2   FORMAT(5X, I3, 2X, F4.1)
```

```
b.    2   FORMAT(6X, I3, 2X, F4.1)
```

```
c.    2   FORMAT(1X, I8, F6.1)
```

Which of the above **FORMAT** statements can be used in place of the **FORMAT** statement in the program to print the output as follows?

```
....+....1....+....2....+....3....+....4.
     469  17.4
```

7. The output of the program given below is as follows

```
....+....1....+....2....+....3....+....4.
 TEST = -3.527  M=***

M = 2531  TEST = -3.5270
 M =    -3.53  M=2531
```

Place the proper **FORMAT** statement numbers with the **PRINT** statements such that the output is as given above.

```
      REAL TEST
      INTEGER M
      TEST = -3.527
      M = 2531
      PRINT   A   , TEST, M
      PRINT   B   , M, TEST
      PRINT   C   , TEST, M
10    FORMAT(2X, 'TEST = ',F6.3, 2X, 'M=', I3)
20    FORMAT(2X, 'M = ',F8.2, 2X, 'M=', I4)
30    FORMAT('0','M =',I5, 2X, 'TEST = ', F7.4)
      END
```

8.5.2 Exercises on FILES

1. Consider the following statement:

```
      READ(8, *, END = 10) A
```

Which of the following statements is (are) correct about the above statement?

1. The value of A will be read from the area after Assume the input for the program is:.

2. At the end of the file, this read statement will transfer control to statement labeled 10.

3. The value of A will be read from the file linked to unit 8.

2. Which of the following statements is/are FALSE about files:

1. The statement that assigns unit number 9 to the input file "DATA" is:

```
OPEN (UNIT = 9, FILE = 'DATA', STATUS = 'OLD')
```

2. The **OPEN** statement for a data file must precede any **READ** or **WRITE** statements that uses that file.
3. A statement that reads two numbers from a file may look like:

```
READ ( 9, *, END = 31) K, L
```

4. The **OPEN** statement for a file should be executed only once in the program.
5. A statement that writes two numbers into a file may look like:

```
PRINT(9, *) K, L
```

6. A file is a collection of data records.
7. A file is usually used only once.
8. A file can be opened at the same time with two different unit numbers.
9. Two files with the same unit number can not be opened at the same time.
10. We store data in files when we do not need them any more.

3. What will be printed by the following programs?

```
1.     INTEGER M, K
       OPEN ( UNIT = 10, FILE = 'INPUT DATA', STATUS = 'OLD')
       READ ( 10, *, END = 10) ( M, K = 1,100)
10     PRINT*, M, K-1
       END
```

Assume that the file 'INPUT DATA' contains the following:

```
1 2 3
4 5
6 7 8 9
6
```

```
2.     INTEGER J, K
       OPEN ( UNIT = 3, FILE = 'FF1', STATUS = 'OLD')
       DO 50 J=1,100
       READ ( 3,*,END = 60) K
50     CONTINUE
60     PRINT*,'THE VALUES ARE:'
       PRINT*,K,J
       END
```

The contents of the file 'FF1' are:

```
20 50 67 45 18 -2 -20
88 66 77 105 55 300
```

```
3.     INTEGER M
       OPEN ( UNIT = 10, FILE = 'INPUT',STATUS = 'OLD')
       READ (10,*) M
20     IF ( M.NE.-1) THEN
           PRINT*,M
           READ(10, *, END = 30) M
           GOTO 20
       ENDIF
       PRINT*, 'DONE'
30     PRINT*, 'FINISHED'
       END
```

Assume that the file 'INPUT' contains the following :

```
7
```

```
3
9
4
-1
```

```
4.      INTEGER N, K
        OPEN ( UNIT = 12, FILE = 'INFILE', STATUS = 'OLD')
        READ*,N
        DO 10 K=1,N
        PRINT*, N
        READ(12,*,END = 15) N
10      CONTINUE
        PRINT*,N
15      CONTINUE
        END
```

Assume the input for the program is:

```
4
```

Given that the file 'INFILE' contains the following data

```
2
3
```

```
5       INTEGER A, B
        OPEN ( UNIT = 10, FILE = 'INPUT DATA', STATUS = 'OLD')
        OPEN ( UNIT = 11, FILE = 'OUTPUT DATA', STATUS = 'NEW')
        READ*,A,B
        READ(10,*) A,B,A
        WRITE(11,*) A, B
        READ(10, *, END = 10) A, B
10      WRITE(11,*) A, B
        END
```

Assume the input for the program is:

```
10 11
```

Assume that the file 'INPUT DATA' contains the following data

```
4 5
6 7
8
```

What will be written in the file 'OUTPUT DATA' file ?

```
6.      INTEGER S, T, U
        OPEN ( UNIT = 10, FILE = 'INPUT',STATUS = 'OLD')
10      READ(10, *, END = 30) S, T
        U = S
        T = U
        U = S
        IF ( S.NE.T) THEN
            U = 1
        ELSE
            U = 0
        ENDIF
        GOTO 10
30      PRINT*, U, S, T
        END
```

Assume the file 'INPUT' contains the following data:

```
3
4
5
6
7
```

```
8
```

```
7.    INTEGER X(6), M, K
      OPEN ( UNIT = 10, FILE = 'INPUT1', STATUS = 'OLD')
      OPEN ( UNIT = 11, FILE = 'INPUT2', STATUS = 'OLD')
      M = 0
10    M = M + 1
      READ(10,*) X(M)
      IF ( X(M).GT.0) GOTO 10
20    M = M + 1
      READ(11,*) X(M)
      IF ( X(M).GT.0) GOTO 20
      PRINT 1, (X(K),K=1,M)
1     FORMAT(' ',I2,I2,I2,I2,I2,I2)
      END
```

Assume you have two files 'INPUT1' and 'INPUT2' with the following data:

INPUT1	INPUT2
3	6
8	0
0	7
5	0

```
8.    INTEGER N, K
      OPEN(UNIT=22, FILE = 'INPUT', STATUS = 'OLD')
33    READ (22,*) N
      IF (N.EQ.0) GOTO 44
      PRINT*, ('*', K=1,N)
      GOTO 33
44    PRINT*, 'HISTOGRAM'
      END
```

Given that the file 'INPUT' contains the following data

```
5
2
4
0
```

4. A set of three real numbers are read from the file TEST and the number associated to the file is 10. The output is then written to a new file called REST and the number associated to the file is 12. Write a FORTRAN 77 program to do the above operations.
5. Write a FORTRAN 77 program to copy an old file "TEST1" to a new "TEST2". It is assumed that each line of "TEST1" contains a student ID and his garde out of 100. The number of data lines in the old file is not known.
6. Write a FORTRAN 77 program which will read values from a data file, the file name is: INPUT and its type is DATA.
    1. Open the INPUT file.
    2. Open a new output file called: ODD DATA.
    3. open a new output file called: EVEN DATA. It is not known exactly how many data there is in the INPUT file.
    4. Use the read (... END =..) to read the values from the file one by one and
    5. If the value is odd, write it in the file: ODD DATA.
    6. If the value is even, write it in the file: EVEN DATA.

7. A file called INPUT is assumed to contain an unknown number of lines, however, we know that every line contains exactly two numbers. Write a program that reads each line from file INPUT and prints the smaller of the two numbers in a file called SMALL and the larger in a file called BIG.
8. The following incomplete program was written to compare two files 'INFOR1' and 'INFOR2'. If the data in the files is the same then the program prints the message 'SAME FILES'. Otherwise the program prints 'DIFFERENT FILES'. Each line in both files contain two integer numbers followed by one logical value. Assume both files have the same number of records. Complete the program:

```
      INTEGER X1, X2, X3, X4
      LOGICAL    (1)   ,    (2)   ,FLAG
      OPEN ( UNIT = 1, FILE = 'INFOR1', STATUS = 'OLD')
                         (3)
      FLAG =    (4)
10    READ (1,*,END =    (5)   ) X1, X2, VAL1
      READ (2,*) X3, X4, VAL2
      IF ( X1.EQ.X3 .AND.          (6)           ) THEN
         GOTO 10
      ELSE
         FLAG = .FALSE.
      ENDIF
20    IF ( FLAG) THEN
         PRINT*,     (7)
      ELSE
         PRINT*,     (8)
      ENDIF
      END
```

8.6 Solutions to Exercises

8.6.1 Solutions to Exercises on Output Design

Ans 1.

1.

```
....+....1....+....2....+....3....+....4.
 123.84  123.84123.83670
```

2.

```
....+....1....+....2....+....3....+....4.
123456789
```

3.

(new page)

```
....+....1....+....2....+....3....+....4.
81.60  9.2

***  48       8.87
```

4.

```
....+....1....+....2....+....3....+....4.
 -35+0.0IS NOT EQUAL 120.0-25
```

5.

```
....+....1....+....2....+....3....+....4.
  T  AND  F


T  F
```

6.

```
....+....1....+....2....+....3....+....4.
X=  25.0  SQUARE ROOT =  5.0
Y=  -35.0  ABSOLUTE VALUE = 35.0
N=  -35  ABSOLUTE VALUE = 35
```

7.

```
....+....1....+....2....+....3....+....4.
THE CAPITAL IS  RIYA
```

8.

```
....+....1....+....2....+....3....+....4.
  10
  20
  30
  40
  50
```

9.

```
....+....1....+....2....+....3....+....4.
1020304050
```

10.

```
....+....1....+....2....+....3....+....4.
10  20  30  40  50
```

11.

```
....+....1....+....2....+....3....+....4.
10.0  30.0  50.0
20.0  40.0  60.0
```

12.

(new page)

```
....+....1....+....2....+....3....+....4.
DOT PRODUCT = 100.0
```

13.

```
....+....1....+....2....+....3....+....4.
23+**  = 124
S1   ***** 427.50
```

Ans 2.

1. VALID
2. VALID

Ans 3.

1.

```
1      FORMAT (5X, 'X=',F5.3)
2      FORMAT ('+', 14X, 'X=', F3.1)
```

2.

```
1520   FORMAT (3X, F4.2, 2X, I2, 1X, F5.1)
```

3.

```
5      FORMAT (' ', 9X, F5.2, 5X, I2)
```

4.

```
5      FORMAT (3X, 'X= ', F4.2,1X, 'Y= ',2X,F5.2, 2X,'Z= ', I3)
```

5

```
1      FORMAT (' ', 8X, A)
2      FORMAT ('+', 1X, A)
```

6.

```
6      FORMAT (' ', 4X, F6.2, 3X, A, 3X, I2)
```

Ans 4.

```
       PRINT 10
10     FORMAT('1', 30X, 'FORTRAN-77--LANGUAGE')
```

Ans 5.

```
       REAL X, RPART
       INTEGER IPART
       READ*, X
       IPART = X
       RPART = X - IPART
       PRINT 5,  X, IPART, RPART
5      FORMAT (' ', F7.3, '=', I3, '+', F5.3)
       END
```

Ans 6.

b or c

Ans 7.

(a) 10
(b) 30
(c) 20

8.6.2 Solutions to Exercises on Files

Ans 1.

2 3

Ans 2.

```
      4       5       7       8       10
```

Ans 3.

```
      6       10
      THE VALUES ARE:
   88  3
      7
   3
   9
   4
   DONE
   FINISHED
      4
   2
   3
      6 5
   8  5
      0  7  7
      3  8  0  6  0
      *****
   **
   ****
   HISTOGRAM
```

Ans 4.

```
      REAL RN1, RN2, RN3
      OPEN( UNIT = 10, FILE = 'TEST', STATUS = 'OLD' )
      OPEN( UNIT = 12, FILE = 'REST', STATUS = 'UNKNOWN' )
      READ(10, *) RN1, RN2, RN3
      WRITE(12, *) RN1, RN2, RN3
      END
```

Ans 5.

```
      INTEGER ID, GRD
      OPEN( UNIT = 1, FILE = 'TEST1', STATUS = 'OLD' )
      OPEN( UNIT = 2, FILE = 'TEST2', STATUS = 'UNKNOWN' )
5     READ(1, *, END = 10) ID, GRD
      WRITE(2, *) ID, GRD
      GOTO 5
10    PRINT*, 'DONE'
      END
```

Ans 6.

```
      INTEGER NUM
      OPEN( UNIT = 20, FILE = 'INPUT DATA', STATUS = 'OLD' )
      OPEN( UNIT = 30, FILE = 'ODD DATA', STATUS = 'UNKNOWN' )
      OPEN( UNIT = 40, FILE = 'EVEN DATA', STATUS = 'UNKNOWN' )
100   READ(20, *, END = 200) NUM
      IF ( MOD( NUM, 2) .EQ. 1 ) THEN
          WRITE(30, *) NUM
      ELSE
          WRITE(40, *) NUM
      ENDIF
      GOTO 100
200   PRINT*, 'DONE'
      END
```

Ans 7.

```
      INTEGER N1, N2
      OPEN( UNIT = 11, FILE = 'INPUT', STATUS = 'OLD' )
      OPEN( UNIT = 12, FILE = 'SMALL', STATUS = 'UNKNOWN' )
      OPEN( UNIT = 13, FILE = 'BIG', STATUS = 'UNKNOWN' )
20    READ(11, *, END = 25) N1, N2
      IF ( N1 .LT. N2 ) THEN
          WRITE(12, *) N1
          WRITE(13, *) N2
      ELSE
          WRITE(12, *) N2
          WRITE(13, *) N1
      ENDIF
      GOTO 20
25    PRINT*, 'DONE'
      END
```

Ans 8.

1. VAL1
2. VAL2
3. **OPEN**(UNIT = 2, FILE = 'INFOR2', STATUS = 'OLD')
4. TRUE.
5. 20
6. X2 .EQ. X4 .AND. VAL1 .EQV. VAL2
7. 'SAME FILES'
8. 'DIFFERENT FILES'