5.3 The WHILE Loop

New versions of Fortran compilers do accept explicit **WHILE** loops. One of these general forms can be presented as follows:

```
DO  WHILE (condition)
       block of statements
END DO
```

In this construct, the *condition* is checked before executing the *block of statements*. The *block of statements* is executed only if the *condition*, which is a logical expression, evaluates to a *true* value. At the end of each iteration (**END DO**), the control returns to the beginning of the loop where the *condition* is checked again. Depending on the value of the *condition*, the decision to continue for another iteration is made. This means that the number of iterations the **WHILE** loop makes depends on the *condition* of the loop and could not always be computed before the execution of the loop starts. This is the main difference between **WHILE** and **DO** repetition constructs.

In other words, The following steps occur when a **DO WHILE** statement is executed:

1. The *condition* (logical expression) is evaluated.
2. If the value of the *condition* is *FALSE*, none of the statements within the loop (*block of statements*) is executed and control jumps to the statement following the **END DO**.
3. If the value of the expression is *TRUE*, the statements within the loop (*block of statements*) are executed.
4. when the **END DO** statement is reached, control returns to the **DO WHILE** statement where the *condition* is evaluated and cycle repeats.

The execution of the loop starts if the *condition* evaluates to a *TRUE* value. Once the loop iterations begin, the *condition* must be ultimately changed to a *FALSE* value, so that the loop stops after a finite number of iterations. Otherwise, the loop never stops resulting in what is known as the *infinite* loop. In the following section, we elaborate more on the **WHILE** loop.

5.3.1Examples on WHILE Loops

Example 1: *Computation of the Average: Write a FORTRAN program that reads the grades of 100 students in a course. The program then computes and prints the average of the grades.*

Solution:

```
    REAL X, AVG, SUM
    INTEGER K
    K = 0
    SUM = 0.0
    DO WHILE (K.LT.100)
        PRINT*, 'ENTER GRADE NUMBER ', K
        READ*, X
        PRINT*, 'INPUT: ', X
        K = K + 1
        SUM = SUM + X
    END DO
    AVG = SUM / K
    PRINT*, AVG
    END
```

Note that the variable K starts at 0. The value of K is incremented after the reading of a grade. The **WHILE** condition (K.LT.100) prevents the loop from reading any new grades once the 100th grade is read. Reading the 100th grade causes K to be incremented to the value of 100 as well. Therefore, when the condition is checked in the next iteration, it becomes FALSE and the loop stops.

In each iteration, the value of the variable GRADE is added to the variable SUM. After the loop, the average is computed by dividing the variable SUM by the variable K.

Example 2: *The Factorial: The problem is the same as the one discussed in Example 2 of Section 5.2. In this context, however, we will solve it using a **WHILE** loop.*

Solution:

```
INTEGER M, TERM, FACT
PRINT*, 'ENTER AN INTEGER NUMBER'
READ*, M
PRINT*, 'INPUT: ', M
IF (M.GE.0) THEN
    FACT = 1
    TERM = M
DO WHILE (TERM.GT.1)
    FACT = FACT *TERM
    TERM =TERM - 1
END DO
    PRINT*,'FACTORIAL OF ', M, ' IS ', FACT
ELSE
    PRINT*, 'NO FACTORIAL FOR NEGATIVES'
ENDIF
END
```

Note the similarities between both solutions. The **WHILE** loop starts from **M** (the value we would like to compute the factorial of) and the condition of the loop makes sure that the loop will only stop when TERM reaches the value 1.

Example 3: *Classification of Boxers: Write a FORTRAN program that reads the weights of boxers. Each weight is given on a separate line of input. The boxer is classified according to the following criteria: if the weight is less than or equal to 65 kilograms, the boxer is light-weight; if the weight is between 65 and 85 kilograms, the boxer is middle-weight and if the weight is more than or equal to 85, the boxer is a heavy-weight. The program prints a proper message according to this classification for a number of boxers by reading their weights repeatedly from the input. This repetitive process of reading and classification stops when a weight of* ***-1.0*** *is read.*

Solution:

```
REAL WEIGHT
PRINT*, 'ENTER WEIGHT'
READ*, WEIGHT
PRINT*, 'INPUT: ', WEIGHT
DO WHILE (WEIGHT.NE.-1.0)
    IF (WEIGHT.LT.0.OR.WEIGHT.GE.400) THEN
        PRINT*, ' WEIGHT IS OUT OF RANGE '
    ELSEIF (WEIGHT.LE.65) THEN
        PRINT*, ' LIGHT-WEIGHT '
    ELSEIF (WEIGHT.LT.85) THEN
        PRINT*, ' MIDDLE-WEIGHT '
    ELSE
        PRINT*, ' HEAVY-WEIGHT '
    ENDIF
    READ*, WEIGHT
END DO
END
```

Note that in this example, the condition that stops the iterations of the **WHILE** loop depends on the **READ** statement. The execution of the loop stops when a value of **-1.0** is read. This value is called the ***end marker*** or the ***sentinel***, since it marks the end of the input. A sentinel must be chosen from outside the range of the possible input values.

5.4 Nested WHILE Loops

WHILE loops may be nested, that is you can put a **WHILE** loop inside another **WHILE** loop. However, one must start the inner loop after starting the outer loop and end the inner loop before ending the outer loop for a logically correct nesting. (The following example is equivalent to the nested **DO** loop example given earlier.)

Example: Consider the following program.

```
    M = 1
    DO WHILE( M .LE. 2)
        J = 1
        DO WHILE(J .LE. 6)
            PRINT*, M, J
            J = J + 2
        END DO
        M = M + 1
    END DO
    END
```

The output of the above program is:

```
1    1
1    3
1    5
2    1
2    3
2    5
```

There are two nested **WHILE** loops in the above program. The outer loop is controlled by the variable M. The inner loop is controlled by the variable J. For each value of the variable M, the inner loop variable J takes the values 1, 3 and 5.

5.5 Examples on DO and WHILE Loops

Example 1: *Evaluation of Series: Write a FORTRAN program that evaluates the following series to the 7th term.*

$$\sum_{i=1}^{N} 3^i$$

(Summation of base 3 to the powers from 1 to N. Assume N has the value 7)

Solution:

```
    INTEGER SUM
    SUM = 0
    DO 11 K = 1, 7
        SUM = SUM + 3 ** K
11  CONTINUE
    PRINT*, SUM
    END
```

Example 2: *Alternating Sequences/ Series: Alternating sequences, or series, are those which have terms alternating their signs from positive to negative. In this example, we find the sum of an alternating series.*

Question: Write a FORTRAN program that evaluates the following series to the 100th term.

1 - 3 + 5 - 7 + 9 - 11 + 13 - 15 + 17 - 19 + ...

Solution:

It is obvious that the terms differ by 2 and start at the value of 1.

```
    INTEGER SUM, TERM,NTERM
    SUM = 0
    TERM = 1
    DO 10 NTERM = 1, 100
        SUM = SUM + (-1) ** (NTERM + 1) * TERM
        TERM = TERM + 2
10  CONTINUE
    PRINT*, SUM
    END
```

Notice the summation statement inside the loop. The expression (-1) ** (NTERM + 1) is positive when NTERM equals 1, that is for the first term. Then, it becomes negative for the second term since NTERM + 1 is 3 and so on.

Example 3: *Series Summation using a WHILE loop: Question: Write a FORTRAN program which calculates the sum of the following series :*

$$\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+\frac{4}{5}+\mathrm{L}+\frac{99}{100}$$

Solution:

```
REAL N, SUM
N = 1
SUM = 0
DO WHILE(N.LE.99)
    SUM = SUM + N / (N + 1)
     N = N + 1
END DO
PRINT*, SUM
END
```

In the above program, if N is not declared as **REAL**, the expression N/(N+1), in the summation inside the loop, will always compute to zero.

Example 4: *Conversion of a WHILE loop to a **DO** loop: Convert the following **WHILE** loop into a **DO** loop.*

```
REAL X, AVG, SUM
INTEGER K
K = 0
SUM = 0.0
DO WHILE(K.LT.100)
    READ*, X
    K = K + 1
    SUM = SUM + X
END DO
AVG = SUM / K
PRINT*, AVG
END
```

In the **WHILE** loop, K starts with the value of 0, and within the loop it is incremented by 1 in each iteration. The *termination condition* is that the value of K must exceed 99. In the equivalent program using a **DO** loop, K starts at 0 and stops at 99 and gets incremented by 1 in each iteration.

Solution:

The equivalent program using a **DO** loop is as follows:

```
REAL X, AVG, SUM
INTEGER K
SUM = 0.0
DO 25 K = 0, 99, 1
    READ*, X
    SUM = SUM + X
25 CONTINUE
AVG = SUM / 100
PRINT*, AVG
END
```

An important point to note in this example is the way the average is computed. The statement that computes the average divides the summation of the grades SUM by 100. Note that the value of the K is 100 because the loop stops when the value of K exceeds 99. Keeping in mind that the increment is 1, the value of K after the loop terminates is 100. However, it is not recommended to use the value of the index outside the **DO** loop.

It is also important to note that any other parameters such as:

```
DO 25 K = 200, 101, -1
```

would also have the same effect. Note that the variable K exits the loop with the value 100 in this case as well.

It is not always possible to convert a **WHILE** loop into a **DO** loop. As an example, consider the **WHILE** loop in the Classification of Boxers example. There, we cannot accomplish the conversion because the number of times the **WHILE** loop gets executed is not known. It depends on the number of data values before the *end marker*.

5.6 Implied Loops

Implied loops are only used in **READ** and **PRINT** statements. The implied loop is written in the following manner :

```
   READ*,(list of variables, index = initial, limit, increment)
   PRINT*,(list of expressions, index = initial, limit,
increment)
```

As in the case of explicit **DO** loops, the index must be either an integer or real expression. The variables in the **READ** statement can be of any type including array elements. The expressions in the **PRINT** statement can be of any type as well. All the rules that apply to **DO** loop parameters also apply to implied loop parameters. Usage of implied loops is given in the following examples :

Example 1: *Printing values from 100 to 87: The following segment prints the integer values from 100 down to 87 in a single line.*

```
   PRINT*, (K, K = 100 , 87 , -1)
```

Output:

```
100  99   98   97   96   95   94   93   92   91   90   89   88
87
```

Notice that the increment is -1, which means that the value of K decreases from 100 to 87. In each iteration, the value of K is printed. The value of K is printed $\left\lceil \frac{87-100}{-1} \right\rceil + 1 = 14$ times. Since K is the index of the loop, the value printed here is the value of the index, which varies in each iteration. Consider the following explicit **DO** loop version of the implied loop :

```
   DO 60 K = 100, 87 , -1
      PRINT*, K
60 CONTINUE
```

Output:

```
100
 99
 98
...
...
...
 87
```

The two loops are equivalent except in terms of the shape of the output. In the implied loop version, the output will be printed on one line. In the explicit **DO** loop version, the output will be printed as one value on each line.

Example 2: *Printing more than one value in each iteration of an implied loop: The following segment prints a percentage sign followed by a + sign three times :*

```
   PRINT*, ('%' , '+' , M = 1 , 3)
```

This produces the following output:

```
%+%+%+
```

Notice that the parenthesis encloses both the % and the + signs, which means they both have to be printed in every iteration the loop makes.

Example 3: Nested Implied Loops: *An implied loop may be nested either in another implied loop or in an explicit **DO** loop. There is no restriction on the number of levels of nesting. The following segment shows nested implied loops.*

```
   PRINT*, ((K, K = 1 , 5 , 2), L = 1 , 2)
```

Nested implied loops work in a similar manner as the nested **DO** loops. One very important point to note here is the double parenthesis before the K in the implied version. It means that the inner loop with index variable K is enclosed within the outer one with index variable L. The L loop is executed $\left\lceil \frac{2-1}{1} \right\rceil + 1 = 2$ times. The K loop forces the value of K to be printed $\left\lceil \frac{5-1}{2} \right\rceil + 1 = 3$ iterations. However, since the K loop is nested inside the L loop, the K loop is executed 3 times in each iteration of the L loop. Thus, K is printed 6 times. Therefore, the output of the implied version is:

```
1    3   5   1   3   5
```

5.7 Repetition Constructs in Subprograms

Subprograms in FORTRAN are considered separate programs during compilation. Therefore, repetition constructs in subprograms are given the same treatment as in programs. The following is an example that shows how repetition is used in subprograms.

Example: *Count of Integers in some Range that are Divisible by a given Value: Write a function subprogram that receives three integers as input. The* ***first*** *and* ***second*** *input integers make the range of values in which the function will conduct the search. The function searches for the integers in that range that are divisible by the* ***third*** *input integer. The function returns the count of such integers to the main program. The main program reads five lines of input. Each line consists of three integers. After each read, the main program calls the function, passes the three integers to it and receives the output from it and prints that output with a proper message :*

Solution:

```
    INTEGER K, L, M, COUNT, J, N
    DO 10 J = 1 , 5
        READ*, K, L, M
        N = COUNT(K , L , M)
        PRINT*, 'COUNT OF INTEGERS BETWEEN',K,'AND', L
        PRINT*, 'THAT ARE DIVISIBLE BY', M, 'IS', N
        PRINT*
10  CONTINUE
    END
    INTEGER FUNCTION COUNT(K , L , M)
    INTEGER K, L, M, INCR, NUM, J
    INCR = 1
    NUM = 0
    IF (L .LT. K) INCR = -1
    DO 10 J = K, L, INCR
        IF (MOD(J , M) .EQ. 0) NUM = NUM + 1
10  CONTINUE
    COUNT = NUM
    RETURN
    END
```

If we use the following input:

```
2    34  2
-15  -30  5
70   32   7
0    20   4
-10  10  10
```

The typical output would be as follows:

```
COUNT OF INTEGERS BETWEEN 2 AND 34
THAT ARE DIVISIBLE BY 2 IS 12

COUNT OF INTEGERS BETWEEN -15 AND -30
THAT ARE DIVISIBLE BY 5 IS 4

COUNT OF INTEGERS BETWEEN 70 AND 32
THAT ARE DIVISIBLE BY 7 IS 6

COUNT OF INTEGERS BETWEEN 0 AND 20
THAT ARE DIVISIBLE BY 4 IS 6

COUNT OF INTEGERS BETWEEN -10 AND 10
THAT ARE DIVISIBLE BY 10 IS 3
```

Remember what we said about the subprogram being a separate entity from the main program invoking it. Accordingly, note the following in the above example:

- It is allowed to use the same statement number in the main program and subprograms of the same file. Notice the statement number **10** in both the main program and the function subprogram

- It is also allowed to use the same variable name as index of **DO** loops in the main program and the subprogram. Notice the variable **J** in the above

5.8 Exercises

1. What will be printed by the following programs?

```
1.  LOGICAL FUNCTION PRIME(K)
    INTEGER N, K
    PRIME = .TRUE.
    DO 10 N = 2, K / 2
        IF (MOD(K , N) .EQ. 0) THEN
            PRIME = .FALSE.
        ENDIF
10  CONTINUE
    RETURN
    END
    LOGICAL PRIME
    PRINT*, PRIME(5), PRIME(8)
    END
```

```
2.  INTEGER FUNCTION FACT(K)
    INTEGER K,L
    FACT = 1
    DO 10 L = 2 , K
        FACT = FACT * L
10  CONTINUE
    RETURN
    END
    INTEGER FUNCTION COMB(N , M)
    INTEGER FACT
    IF (N .GT.M) THEN
        COMB = FACT(N) / (FACT(M) * FACT(N-M))
    ELSE
        COMB = 0
    ENDIF
    RETURN
    END
    INTEGER COMB
    PRINT*, COMB(4 , 2)
    END
```

```
3.  INTEGER K, M, N
    N = 0
    DO 10 K = -5 , 5
        N = N + 2
        DO 20 M = 3 , 1
            N = N + 3
20      CONTINUE
        N = N + 1
10  CONTINUE
    PRINT*, N
    END
```

```
4.  INTEGER ITOT, N
    READ*, N
    ITOT = 1
    DO WHILE(N .NE. 0)
        ITOT = ITOT * N
        READ*, N
    END DO
    READ*, N
    DO WHILE(N .NE. 0)
        ITOT = ITOT * N
        READ*, N
    END DO
    PRINT*,ITOT
    END
```

Assume the input is

```
2
0
3
0
4
```

```
5.  INTEGER FUNCTION CALC(A,B)
    INTEGER A,B,R, K
    R = 1
    DO 10 K=1,B
        R = R*A
10  CONTINUE
    CALC = R
    RETURN
    END
    INTEGER CALC
    READ*,M,N
    PRINT*,CALC(M,N)
    END
```

Assume the input is

```
2    5
```

```
6.  INTEGER KK, J, K
        KK = 0
    DO WHILE( KK.LE.0)
        READ*, J , K
        KK = J - K
    END DO
    PRINT*,KK,J,K
    END
```

Assume the input is

```
2   3
-1   2
3   3
4   -3
2   5
4   3
```

```
7.  INTEGER K, J
    K = 2
    DO WHILE( K.GT.0 )
        DO 15 J = K, 3, 2
            PRINT*, K, J
15      CONTINUE
        K = K - 1
    END DO
    END
```

```
8.  INTEGER N, C
    LOGICAL FLAG
    READ*, N
    FLAG = .TRUE.
    C = N ** 2
    DO WHILE( FLAG )
        C = ( C + N ) / 2
        FLAG = C.NE.N
        PRINT*, C
    END DO
    END
```

Assume the input is

```
4
```

```
9.  INTEGER N, K
    READ*, N
    K = SQRT(REAL(N))
    DO WHILE( K*K .LT. N )
        K = K + 1
    END DO
    PRINT*, K*K
    END
```

Assume the input is

```
 6
```

```
10. INTEGER J, K
    DO 10 K = 1,2
        PRINT*, K
        DO 10 J = 1,3
10          PRINT*,K,J
    END
```

```
11. INTEGER X, K, M
    M = 4
    DO 100 K = M ,M+2
        X = M + 2
        IF ( K.LT.6 ) THEN
            PRINT*,'HELLO'
        ENDIF
100 CONTINUE
    END
```

```
12. INTEGER SUM, K, J, M
    SUM = 0
    DO 1 K = 1,5,2
        DO 2 J = 7,-2,-3
            DO 3 M = 1980,1989,2
                SUM = SUM + 1
3           CONTINUE
2       CONTINUE
1   CONTINUE
    PRINT*,SUM
    END
```

```
13. LOGICAL T, F
    INTEGER BACK, FUTURE, K
    BACK = 1
    FUTURE = 100
    T = .TRUE.
    F = .FALSE.
    DO 99 K = BACK,FUTURE,5
        T = ( T.AND..NOT.T ) .OR. ( F.OR..NOT.F )
        F = .NOT.T
        FUTURE = FUTURE*BACK*(-1)
99  CONTINUE
    IF (T) PRINT*, 'DONE'
    IF (F) PRINT*, 'UNDONE'
    END
```

2. Find the number of iterations of the WHILE-LOOPS in each of the following programs:

```
1.  INTEGER K, M, J
    K = 80
    M = 5
    J = M-M/K*K
    DO WHILE (J.NE.0)
        PRINT*, J
        J = M-M/K*K
        M = M + 1
    END DO
    END
```

```
2.  REAL W
    INTEGER L
    W = 2.0
    L = 5 * W
    DO WHILE( L/W.EQ.((L/4.0)*W) )
        PRINT*, L
        L = L + 10
    END DO
    END
```

3. Which of the following program segments causes an infinite loop?

```
(I) J = 0
    DO WHILE( J.LT.5 )
        J = J + 1
    END DO
    PRINT*, J
```

```
II. J = 0
25  DO WHILE( J.LT.5 )
        J = J + 1
    END DO
    GOTO 25
    PRINT*, J
```

```
III.    X = 2.0
5   X = X + 1
    IF ( X.GT.4 ) X = X + 1
    GOTO 5
    PRINT*, X
```

```
IV. M = 2
    K = 1
10  DO WHILE ( K.LE. M )
20      M = M + 1
        K = K + 2
        GOTO 20
    END DO
    GOTO 10
```

```
V.  X = 1
4   IF ( X.GE.1 ) GOTO 5
5   IF ( X.LE.1 ) GOTO 4
```

```
VI. J = 1
33  IF ( J.GT.5 ) THEN
        GOTO 22
    ENDIF
    PRINT*, J
    J = J + 1
    GOTO 33
22  STOP
```

4. Convert the following WHILE loops to **DO** loops :

```
I.  ID = N
10  DO WHILE( ID.LE.891234 )
        PRINT*, ID
        ID = ID + 10
    END DO
```

```
II. L = 1
    SUM =0
    DO WHILE(L.LE.15)
        J = -L
        DO WHILE(J.LE.0)
            SUM =SUM+J
            J = J + 1
        END DO
        L = L+3
    END DO
    PRINT*,SUM
```

5. What will be printed by the following program :

```
    INTEGER ISUM, K, N
    ISUM = 0
    READ*, N
    DO 6 K = 1,N
        ISUM = ISUM +(-1)**(K-1)
6   CONTINUE
    PRINT*, ISUM
    END
```

If the input is:

a.

```
9
```

b.

```
8
```

c.

```
51
```

d.

```
98
```

6. The following program segments may or may not have errors. Identify the errors (if any).

```
1.  INTEGER K, J
    DO 6 K = 1,4
        DO 7 J = K-1,K
            PRINT*, K
6       CONTINUE
7   CONTINUE
    END
```

```
2.  INTEGER K, J
    K = 10
    J = 20
    DO WHILE( J.GT. K )
        K = K/2
    END DO
    END
```

7. Write a FORTRAN 77 program to calculate the following summation:

$$\sum_{k=1}^{200}\left((-1)^k \frac{5k}{k+1}\right)$$

8. Write a program that reads the values of two integers M and then prints all the odd numbers between the two integers.(Note: M may be less than or equal to N or vice-versa).

9. Write a program that prints all the numbers between two integers M and N which are divisible by an integer K. The program reads the values of M, N and K.

10. Write a program that prints all the perfect squares between two integers M and N. Your program should read the values of M and N. (Note: A perfect square is a square of an integer, example $25 = 5 \times 5$)

11. Using nested WHILE loops, print the multiplication table of integers from 1 to 10. Each multiplication table goes from 1 to 20. Your output should be in the form :

```
1 * 1 = 1
1 * 2 = 2
:
1 * 20 = 20
:
10 * 1 = 10
10 * 2 = 20
:
10 * 20 = 200
```

12. Rewrite the program in the previous question using nested **DO** loops.

13. Complete the **PRINT** statement in the following program to produce the indicated output.

```
    DO 1 K = 1,5
       PRINT*,
1   CONTINUE
    END
```

OUTPUT:

```
=****
*=***
**=**
***=*
****=
```

14. Complete the following program in order to get the required output.

```
    DO 10 K = 10,___(1)____ ,___(2)___
       PRINT*,(__(3)__, L = __(4)__, K )
10  CONTINUE
    END
```

The required output is :

```
5       6   7   8   9   10
5       6   7   8   9
5       6   7   8
5       6   7
5       6
5
```

5.9 Solutions to Exercises

Ans 1.

1. T F
2. 12
3. 33
4. 6
5. 25
6. 7 4 -3
7. 1 0 50
8. 10
   7
   5
   4
9. 9
10. 1
    1 1
    1 2
    1 3
    2
    2 1
    2 2
    2 3
11. HELLO
    HELLO
12. 60
13. DONE

Ans 2.

1. 76

2. INFINITE LOOP

Ans 3.

II , III , IV , V

Ans 4.

I)

```
    DO 10 ID = N , 891234 , 10
        PRINT*, ID
10  CONTINUE
```

II)

```
    SUM = 0
    DO 3 L = 1 , 15 , 3
        DO 2 J = -L , 0 , 1
            SUM = SUM + J
2       CONTINUE
3   CONTINUE
```

Ans 5.

A) 1 B) 0 C) 1 D) 0

Ans 6

1) IMPROPER NESTING OF **DO** LOOPS

2) INFINITE LOOP

Ans 7.

```
    REAL SUM
    INTEGER K
    SUM = 0
    DO 10 K = 1 , 200
        SUM = SUM + (-1) ** K * (REAL(5*K) / ( K+1))
10  CONTINUE
    PRINT*, SUM
    END
```

Ans 8.

```
    INTEGER M , N , TEMP
    READ*, M , N
    IF( M .LT. N ) THEN
        TEMP = N
        N = M
        M = TEMP
    ENDIF
    DO 5 L = M , N
        IF( L/2 * 2 .NE. L ) PRINT*,L
5   CONTINUE
    END
```

Ans 9.

```
    INTEGER M , N , K , TEMP
    READ*, M , N , K
    IF( M .LT. N ) THEN
        TEMP = N
        N = M
        M = TEMP
    ENDIF
    DO 5 L = M , N
        IF( L/K * K .EQ. L ) PRINT*,L
5   CONTINUE
    END
```

Ans 10.

```
    INTEGER M , N , TEMP
    READ*, M , N
    IF( M .LT. N ) THEN
        TEMP = N
        N   = M
        M   = TEMP
    ENDIF
    DO 5 L = M , N
        IF( INT(SQRT(REAL(L)) ** 2 .EQ. L )) PRINT*,L
5   CONTINUE
    END
```

Ans 11.

```
    INTEGER I, J
    I = 1
    DO WHILE(I .LE. 10 )
        J = 1
        DO WHILE( J .LE. 20 )
            PRINT*, I, ' * ', J, ' = ', I*J
            J = J + 1
        END DO
        I = I + 1
    END DO
    END
```

Ans 12.

```
    INTEGER I, J
    DO 10 I = 1 , 10
        DO 10 J = 1 , 20
           PRINT*, I, ' * ', J, ' = ', I*J
10  CONTINUE
    END
```

Ans 13.

```
PRINT*, ('*', J = 1, K-1), '=' , ('*', M = 1 , 5-K)
```

Ans 14.

1) 5 2) -1 3) L 4) 5