6 ONE-DIMENSIONAL ARRAYS

It is fairly common in programs to read a large quantity of input data, process the data and produce the computations as output. Such large amounts of input data cannot be stored in simple variables. We need bigger data structures to store such data in memory. For example, consider a problem to compute the average, given the grades of a number of students as input, and list the grades of those students below average. The grades must be stored in the memory while reading because, after the average is computed, they have to be processed again (to list those below average). For a large number of students, simple variables cannot be used to store the grades. We require structures such as arrays. In this and the following chapter, we introduce data structures that allow storage of large amounts of data.

In the previous chapters, we learnt that a variable represents a single location in the memory. Unlike variables, a one-dimensional *array* (1-D array) represents a group of memory locations. Each member of an array is called an *element*. An element in an array is accessed by the array name followed by a *subscript* (also called an *index*) enclosed in parentheses. Subscripts are *integer* constants or expressions that indicate the location of the element within the array. All elements of an array store the same type of data. Thus all elements in an integer array will contain integer values. In FORTRAN, arrays *must* be declared at the beginning of a program or a subprogram.

6.1 One-Dimensional Array Declaration

Arrays must be declared using a declaration statement. If an integer array is to be declared, then the **INTEGER** declaration statement is used. Similarly, for declaring real, logical or character arrays, the respective declaration statement is used. Before executing a program, a computer should know the total memory space required by the program. Each array declaration informs the computer of the amount of memory space required by that array. Therefore, all arrays must be *declared*.

Example 1: *Declaration of an integer array LIST consisting of 20 elements.*

```
      INTEGER LIST (20)
```

Example 2: *Declaration of a logical array FLAG that consists of 30 elements.*

```
      LOGICAL FLAG (30)
```

Example 3: *Declaration of a character array NAMES that consists of 15 elements with each element of size 20.*

```
      CHARACTER NAMES (15)*20
```

Example 1, declares an array LIST consisting of 20 elements. The first element has the subscript 1 and the last element has the subscript 20. We may also declare arrays with subscript beginning from any integer, positive or negative, other than 1.

Example 4: *Declaration of a real array YEAR used to represent rainfall in years 1983 to 1994.*

```
      REAL YEAR (1983 : 1994)
```

The array YEAR has 12 elements. If an array is declared in the format *array_name (m:n)*, we have to ensure that *n* must be greater than *m*. Also note that both *m* and *n* can be either positive or negative integer as long as *n* is greater than *m*.

Example 5 : *Declaration of a real array TEMP with subscript ranging from -20 to 20.*

```
      REAL TEMP (-20:20)
```

A total of 41 elements in this array can be found using the formula *n - m + 1* where *n* is 20 and *m* is -20.

The declaration statement **DIMENSION** is also used to declare arrays. This statement assumes that the type of the array is implicitly defined. The **DIMENSION** statement can be combined with an explicit type statement declaring the type of the array. If an array is declared using the **DIMENSION** statement, and if the type of the array is not mentioned, it is decided implicitly by the first character of the array name, as in the case of undeclared variables.

Example 6 : *Declaration of arrays using the **DIMENSION** statement.*

```
      DIMENSION ALIST(100), KIT(-3:5), XYZ(15)
      INTEGER XYZ
      REAL BLIST(12), KIT
```

In this example, arrays ALIST, BLIST and KIT are of type **REAL**. Array XYZ is of type **INTEGER**. Since the type of array ALIST is not specified, it is treated as a real variable using the default rule for implicit variables.

6.2 One-Dimensional Array Initialization

The purpose of declaring arrays is to specify the number of elements in each array. By declaring an array, the memory space required by the array is only reserved and not initialized. Arrays can be filled with data using either the assignment statement or the **READ** statement.

6.2.1 Initialization Using the Assignment Statement

The following statements illustrate the initialization of arrays using the assignment statement, in different ways:

Example 1: *Declare a real array LIST consisting of 3 elements. Also initialize each element of LIST with the value zero.*

Solution:

```
      REAL LIST(3)
      DO 5 K = 1, 3
         LIST(K) = 0.0
5     CONTINUE
```

Example 2: *Declare an integer array POWER2 with subscript ranging from 0 up to 10 and store the powers of 2 from 0 to 10 in the array.*

Solution:

```
      INTEGER POWER2(0:10)
      DO 7 K = 0, 10
         POWER2(K) = 2 ** K
7     CONTINUE
```

6.2.2 *Initialization Using the READ Statement*

An array can be read as a whole or in part. To read the whole array, we may use the name of the array without subscripts. We can read part of an array by specifying specific elements of the array in the **READ** statement. We may also use the implied loop in reading arrays. Implied loops provide an elegant approach to reading arrays of varying lengths.

The rules that apply in reading simple variables also apply in reading arrays. Each **READ** statement requires a *new* line of input data. If the data in the input line is not enough, the **READ** statement ensures that the data is read from the immediately following input line or lines, until all the elements of the **READ** statement are read.

Example 1: *Read all the elements of an integer array X of size 4. The four input data values are in a single input data line as follows*

```
10, 20, 30, 40
```

Solution 1: (Without Array Subscript)

```
      INTEGER X(4)
      READ*, X
```

Solution 2: (Using an Implied Loop)

```
      INTEGER X(4), K
      READ*, (X(K), K = 1, 4)
```

Both **READ** statements *read* all four elements of the array X. However, in both solutions, only one **READ** statement is executed. Ideally, the four input data values may be placed in one input line. If the four values of the input data appear in more than one input line, then reading continues until all four values are read. The two solutions are equivalent with a subtle difference. The **READ** statement in Solution 2 may be used to read all four elements of the array or fewer than four elements by modifying the implied loop. In the next example, we will read one input data value per line.

Example 2: *Read all the elements of an integer array X of size 4. The four input data values appear in four input data lines as follows*

```
10
20
30
40
```

Solution:

```
      INTEGER X(4), J
      DO 22 J = 1, 4
         READ*, X(J)
22    CONTINUE
```

Notice the layout of the input data. Since four **READ** statements are executed in the **DO** loop, four input data lines are required each with one data value. The input data for this example can also be used for the previous example (Example 1) but the input of the previous example cannot be used for the current one. The next three examples further illustrate reading of one-dimensional arrays.

Example 3: *Read an integer one-dimensional array of size 100.*

Solution 1: (Using a WHILE Loop)

```
      INTEGER A(100), K
      K = 0
66    IF (K.LT.100) THEN
          K = K + 1
          READ*, A(K)
          GOTO 66
      ENDIF
```

Note that we require 100 lines of input with one data value per line since the **READ** statement is executed 100 times.

Solution 2: (Using a DO Loop)

```
      INTEGER A(100), K
      DO 77 K = 1, 100
          READ*, A(K)
77    CONTINUE
```

Note again that we require 100 lines of input with one data value per line since the **READ** statement is executed 100 times.

Solution 3: (Using an implied Loop)

```
      INTEGER A(100), K
      READ*, (A(K), K = 1, 100)
```

Note that we require one line with 100 data values since the **READ** statement is executed only once. Even if the input is given in 100 lines with one data value per line, the implied loop will correctly read the input.

Example 4: *Read the first five elements of a logical array PASS of size 20.* The input is:

```
T, F, T, F, F
```

Solution:

```
      LOGICAL PASS(20)
      INTEGER K
      READ*, (PASS(K), K = 1, 5)
```

Example 5: *Read the grades of N students into an array SCORE. The value of N is the first input data value followed by N data values in the next input line.* Assume the input is:

```
6
55, 45, 37, 99, 67, 58
```

Solution:

```
      INTEGER SCORE(100), K, N
      READ*, N
      READ*, (SCORE(K), K = 1, N)
```

In this example, the value of N is 6 and the six grades in the second input line are stored as the first six elements of the array SCORE. The rest of the array SCORE is not initialized. Note that the value of N may range from 1 to 100 depending on the first data value in the input. If the input data were given as follows:

```
4
42, 77, 89, 70
```

the value of N will be 4 and only four elements of the array SCORE are initialized. We assume here that the value of N will never go beyond 100 and that there will *k+1* data values in the input where *k* represents the first data value.

6.3 Printing One-Dimensional Arrays

Just as in the case of reading an array, printing an array without subscripts will produce the whole array as output. If some elements of the array are not initialized before printing, question marks appear in the output indicating elements that do not have a value. Each **PRINT** statement starts printing in a new line. If the line is not long enough to print the array, the output is printed in more than one line.

Example : *Read an integer array X of size 4 and print*:

i. the entire array X in one line;

ii. one element of array X per line; and

iii. array elements greater than 0.

Solution:

```
      INTEGER X(4), K
      READ*, X
C PRINTING THE ENTIRE ARRAY IN ONE LINE
      PRINT*, 'PRINTING THE ENTIRE ARRAY'
      PRINT*, X
C PRINTING ONE ARRAY ELEMENT PER LINE
      PRINT*, 'PRINTING ONE ARRAY ELEMENT PER LINE'
      DO 33 K = 1, 4
         PRINT*, X(K)
33    CONTINUE
C PRINTING ARRAY ELEMENTS GREATER THAN 0
      PRINT*, 'PRINTING ARRAY ELEMENTS GREATER THAN 0'
      DO 44 K = 1, 4
         IF(X(K) .GT. 0) PRINT*, X(K)
44    CONTINUE
      END
```

If the input is given as

```
7, 0, 2, -4
```

the output of the program is as follows:

```
PRINTING THE ENTIRE ARRAY
7     0     2     -4
PRINTING ONE ARRAY ELEMENT PER LINE
7
0
2
-4
PRINTING ARRAY ELEMENTS GREATER THAN 0
7
2
```

6.4 Errors in Using One-Dimensional Arrays

There are many errors that may occur in the use of arrays. These errors may appear, if the following rules are not followed:

- Array subscripts must not go beyond the array boundaries.
- Array subscripts must always appear as integer expressions.
- The value assigned to an array element, either using the **READ** statement or the assignment statement, must match in type with the array type. This rule, as in the case of simple variables, does not hold for integer and real variables.
- Arrays must be declared before its elements are initialized.

We will now illustrate a few errors through examples. Assume the following declarations:

```
      INTEGER GRADE(25), LIST(3)
      LOGICAL MEM(20)
      CHARACTER TEXT(5) * 3
```

The following statements illustrate *incorrect* initializations of arrays:

Initialization	Type of Error
GRADE(26) = 0.0	array subscript 26 is out of range
LIST(2.0) = X * 3	array subscript 2.0 is not an integer
TEXT(4) = 100	array TEXT is a character array
MEM(3) = 'WRONG'	array MEM is a logical array
READ*, (GRADE(K), K = 1, 100)	array GRADE has only 25 elements
ARR(2) = 3	ARR is not declared as an array

6.5 Complete Examples on One-Dimensional Arrays

In this section, we illustrate the use of one-dimensional arrays through complete examples.

Example 1: *Counting Odd Numbers: Read an integer N and then read N data values into an array. Print the count of those elements in the array that are odd.*

Solution:

```
      INTEGER A(50), COUNT, N , K
      READ*, N, (A(K), K = 1, N)
      COUNT = 0
      DO 44 K = 1, N
         IF (MOD (A(K), 2) .EQ. 1) COUNT = COUNT + 1
44    CONTINUE
      PRINT 'COUNT OF ODD ELEMENTS = ', COUNT
      END
```

If the input is:

```
7, 35, 66, 83, 22, 33, 1, 89
```

The value of variable N in this example is 7. The next seven input data values are placed in the array. There are 5 odd values among the seven elements of the array. For the given input, the output is as follows:

```
COUNT OF ODD ELEMENTS = 5
```

Example 2: *Reversing a One-Dimensional Array: Write a FORTRAN program that reads an integer one-dimensional array of size N. The program then reverses the elements of the array and stores them in reverse order in the same array. For example, if the elements of the array are:*

33 20 2 88 97 5 71

the elements of the array after reversal should be:

71 5 97 88 2 20 33

The program prints the array, one element per line.

Solution:

```
      INTEGER NUM(100), TEMP
      READ*, N, (NUM(L), L = 1, N)
      DO 41 K = 1, N / 2
         TEMP = NUM(K)
         NUM(K) = NUM(N + 1 - K)
         NUM(N + 1 - K) = TEMP
41    CONTINUE
      DO 22 L = 1, N
         PRINT*, NUM(L)
22    CONTINUE
      END
```

Note that we used an implied loop to read the array and a **DO** loop to print the array. Since the problem asks for an array of size N to be read, we first read N and then use an implied loop to read N elements into the array. One common mistake here is to declare an array of size N. This is not allowed since the size of an array in a declaration statement must be an integer constant (except in the case of subprograms where it may be a dummy argument as we shall see in an example later in this chapter). The array is reversed by exchanging the elements of the array. The expression N+1-K gives the index of the element corresponding to K from the end of the array. Thus, using this expression, the first element is exchanged with the last, the second element is exchanged with the second last and so on. This operation is called *swapping*. The swapping of elements in the array stops at the middle element.

Example 3: *Manipulating One-Dimensional Arrays: Write a FORTRAN program that reads a one-dimensional integer array X of size 10 elements and prints the maximum element and its index in the array.*

Solution:

```
      INTEGER X(10), MAX, INDEX, K
      READ*, X
      MAX = X(1)
      INDEX = 1
      DO 1 K = 2, 10
         IF (X(K) .GT. MAX) THEN
            INDEX = K
            MAX = X(K)
         ENDIF
1     CONTINUE
      PRINT*, 'MAXIMUM:', MAX, ' INDEX:',INDEX
      END
```

In the above program, we need to keep track of the position of the maximum element within the array. The variable MAX stores the current maximum and the variable

INDEX represents the position of the maximum element in the array. Whenever a new maximum is found by the **IF** statement condition, we update both variables MAX and INDEX.

Example 4: *Printing Perfect Squares: Read 4 data values into an array LIST (of size 10) and print those values that are perfect squares (1, 4, 9, 25 .. are perfect squares). Assume that the input is:*

```
81, 25, 10, 169
```

Solution:

```
      INTEGER LIST(10), N, K
      LOGICAL PSQR
C    STATEMENT FUNCTION TO CHECK FOR PERFECT SQUARES
      PSQR(N) = INT(SQRT(REAL(N))) ** 2 .EQ. N
      READ*, (LIST(K), K = 1, 4)
      K = 0
55    IF (K .LE. 4) THEN
         IF(PSQR(LIST(K))) PRINT*, LIST(K)
         K = K + 1
         GOTO 55
      ENDIF
      END
```

In this example, only four elements of the array LIST are initialized by the **READ** statement. The other six elements are not initialized. Notice the use of the logical statement function PSQR that checks whether its argument N is a perfect square. The simple **IF** statements check if the four elements of the array LIST are perfect squares. For the given input, the output is as follows:

```
81
25
169
```

6.6 One-Dimensional Arrays and Subprograms

One-dimensional arrays can be passed to a subprogram or can be used locally within a subprogram. In both the cases, the array must be declared within the subprogram. The size of such an array can be declared as a constant or as a variable. Variable-sized declaration of one-dimensional arrays in a subprogram is allowed only if both the variable size is a dummy argument and the array itself is a dummy argument. The following examples illustrate the use of one-dimensional arrays in a subprogram.

Example 1: *Summation of Array Elements: Read 4 data values into an array LIST (of size 10) and print the sum of all the elements of array LIST using a function SUM.*

Solution:

```
      INTEGER LIST(10), SUM, K
      READ*, (LIST(K), K = 1, 4)
      PRINT*, SUM(LIST, 4)
      END
      INTEGER FUNCTION SUM(MARK, N)
      INTEGER N, MARK(N)
      SUM = 0
      DO 13 J = 1, N
              SUM = SUM + MARK(J)
13    CONTINUE
      RETURN
      END
```

In this example, four elements of the array LIST are read by the **READ** statement. The function SUM is called and the sum of the first four elements of array LIST is printed. The first argument to the function is the one-dimensional array LIST. The second argument is passed as the size of the array. In function SUM, the argument N is used in the declaration of the array MARK. The declaration ***INTEGER*** *MARK(N)* implies that the size of the array MARK is the value of N. This type of declaration is allowed in *functions* and *subroutines* only. The elements of the array MARK are added and the result is returned as the function value.

If the input to this program is as follows:

```
19, 25, 10, 82
```

the output would be as follows:

```
136
```

Example 2: *A Function to Compare One-Dimensional Arrays: Write a program that has a* logical *function* COMPAR. *The function gets A, B, and N as arguments. A and B are integer one-dimensional arrays of equal size. N is an integer that represents the size of arrays A and B. The function compares the elements of A and B. If all elements of A are equal to the corresponding elements of B, the function returns the value .TRUE.. Otherwise, it returns a .FALSE. value. In the main program, N is read. The program also reads two one-dimensional arrays (each of maximum size 100). Only N elements of each array are read. The program then calls the function* COMPAR. *If the value returned is .TRUE., it prints one of the arrays. Otherwise, it prints the two arrays.*

Solution:

```
      LOGICAL FUNCTION COMPAR(A, B, N)
      INTEGER N, A(N), B(N), K
      COMPAR = .TRUE.
      DO 10 K = 1, N
          IF (A(K).NE.B(K)) THEN
              COMPAR = .FALSE.
              RETURN
          ENDIF
10    CONTINUE
      RETURN
      END
      LOGICAL COMPAR
      INTEGER A(100), B(100), K, N
      READ*, N, (A(K), K=1,N), (B(K), K=1,N)
      IF (COMPAR(A,B,N)) THEN
          PRINT*, 'A = B = ', (A(K), K=1,N)
      ELSE
          PRINT*, 'A = ', (A(K), K=1,N)
          PRINT*, 'B = ', (B(K), K=1,N)
      ENDIF
      END
```

Notice how the array declarations are different in the main program from the subprogram. Array A is declared as A(100) in the main program while it is declared with *variable size* as A(N) in the subprogram.

Example 3: *Counting Negative Numbers within a One-Dimensional Array: Write a subroutine* FIND *that takes a one-dimensional array and its size as two input arguments. It returns the count of the negative and non-negative elements of the array.*

Solution:

```
      SUBROUTINE FIND(A, N, COUNT1, COUNT2)
      INTEGER N, A(N), COUNT1, COUNT2, K
      COUNT1 = 0
      COUNT2 = 0
      DO 13 K = 1,N
      IF (A(K).LT.0) THEN
          COUNT1= COUNT1 + 1
      ELSE
          COUNT2= COUNT2 + 1
      ENDIF
13    CONTINUE
      RETURN
      END
```

The variable COUNT1 counts the negative numbers in the array. The variable COUNT2 counts the non-negative integers in the array.

Example 4: *Updating the Values in a One-Dimensional Array: The two input arguments to a certain subroutine* UPDATE *is an array* A *of real numbers and its size* N. *The subroutine replaces the value of every element in* A *with its absolute value. Write the subroutine* UPDATE *and a main program which will invoke (call) the subroutine. The maximum size of the array is 100.*

Solution:

```
      SUBROUTINE UPDATE(A,N)
      INTEGER K, N
      REAL A(N)
      DO 44 K = 1,N
          A(K) = ABS(A(K))
44    CONTINUE
      RETURN
      END
      INTEGER J, N
      REAL A(100)
      READ*, N, (A(J),J=1,N)
      PRINT*, 'THE ORIGINAL ARRAY: ', (A(J),J=1,N)
      CALL UPDATE(A,N)
      PRINT*, 'THE NEW ARRAY: ', (A(J),J=1,N)
      END
```

6.7 Exercises

1. What is printed by the following programs?

```
1.    INTEGER A(3), J
      A(1) = 1
      DO 30 J = 2, 3
          A(J) = 3 * A(J - 1)
30    CONTINUE
      PRINT*, A
      END
```

```
2.    INTEGER X(3), Y(3), K
      LOGICAL Z(3)
      READ*, X
      READ*, Y
      DO 80 K = 1, 3
          Z(K) = X(K) .EQ. Y(K)
80    CONTINUE
      IF(Z(1) .AND. Z(2) .AND. Z(3)) THEN
          PRINT*, 'EQUAL ARRAYS '
      ELSE
          PRINT*, 'DIFFERENT ARRAYS'
      ENDIF
      END
```

Assume the input for the program is:

```
1, 5, 7
7, 5, 1
```

```
3.    INTEGER A(4), B(4), G, K, N
      G(K) = K ** 2
      READ*, A
      DO 60 N = 1, 4
          B(N) = G(A(5 - N))
60    CONTINUE
      PRINT*, B
      END
```

Assume the input for the program is:

```
10, 20, 30, 40
```

```
4.     SUBROUTINE FUN(A)
       INTEGER A(4), TEMP
       TEMP = A(1)
       A(1) = A(2)
       A(2) = A(3)
       A(3) = A(4)
       A(4) = TEMP
       RETURN
       END
       INTEGER LIST(4)
       READ*, LIST
       CALL FUN (LIST)
       PRINT*, LIST
       END
```

Assume the input for the program is:

```
3, 6, 9, 2
```

```
5.     INTEGER X(3), Y(3)
       LOGICAL EQUAL
       READ*, X
       READ*, Y
       IF (EQUAL (X, Y))THEN
           PRINT*, 'EQUAL ARRAYS '
       ELSE
           PRINT*, 'DIFFERENT ARRAYS'
       ENDIF
       END
       LOGICAL FUNCTION EQUAL(X, Y)
       INTEGER X(3), Y(3), K
       LOGICAL Z(3)
       DO 45 K = 1, 3
           Z(K) = X(K) .EQ. Y(K)
       45 CONTINUE
       EQUAL = Z(1) .AND. Z(2) .AND. Z(3)
       RETURN
       END
```

Assume the input for the program is:

```
1, 5, 7
7, 5, 1
```

```
6.     INTEGER A(2), B(3), C(4), D(3)
       READ*, A, D(1)
       READ*, B, D(2)
       READ*, C, D(3)
       PRINT*, A
       PRINT*, B
       PRINT*, C
       PRINT*, D
       END
```

Assume the input for the program is:

```
1,2,3,4,5
6,7,8,9,10
11,12,13,14,15
16,17,18,19,20
```

```
7.     INTEGER A(3), K
       READ*, A
       DO 10 K = 1,3
           A(3) = A(3) + A(K)
10     CONTINUE
       PRINT*, A(3)
       END
```

Assume the input for the program is:

```
10,20,30
```

```
8.     INTEGER X(5), Y(5), N, K
       READ*, N, (X(K),Y(K),K=1,N)
       DO 5 K=X(N),Y(N)
           PRINT*, ('X',J=X(K),Y(K))
5      CONTINUE
       END
```

Assume the input for the program is:

```
4,1,2,3,3,3,4,2,4
```

```
9.     NTEGER A(0:4), K
       DO 10 K = 1,2
           READ*, A
10     CONTINUE
       READ*,(A(K), K = 0,2)
       DO 30 K = 1,20,3
           A(MOD(K,4)) = A(MOD(K,5))
30     CONTINUE
       PRINT*, A
       END
```

Assume the input for the program is:

```
1,2,3,4,5,6,7,8
9,10,11
12,13,14,15
18,19,20
```

```
10.    LOGICAL X(0:4)
       INTEGER J, K
       X(0) = .TRUE.
       DO 30 J = 0,4
           K = MOD(J+1,5)
           X(K) = .NOT. X(J)
30     CONTINUE
       PRINT*, X
       END
```

```
11.    INTEGER A(5), B(5), K
       REAL F, Z
       READ*, (A(K),K=1,4), (B(K),K=1,4)
       Z = F(A,B)
       PRINT*, Z
       END
       REAL FUNCTION F(L,M)
       INTEGER L(5), M(5), K
       F = 0
       DO 10 K = 1,4
           IF (L(K).EQ.M(L(K))) THEN
               F = M(K) + K
           ELSE
               RETURN
           ENDIF
10     CONTINUE
       F = F + K
       RETURN
       END
```

Assume the input for the program is:

```
3,1,2,4,1,2,3,4
```

```
12.    INTEGER A(100), I, J, N
       REAL ENDAVE
       DO 2 I=1,4
           READ*, N, (A(J),J=1,N)
           PRINT*, ENDAVE(A,N)
2      CONTINUE
       END
       FUNCTION ENDAVE(X,V)
       INTEGER V, X(V)
       REAL ENDAVE
       ENDAVE = (X(1)+X(V)) / 2.0
       END
```

Assume the input for the program is:

```
4 5 7 3 1
5 7 3 1 4 5
3 1 5 4
1 2
```

```
13.    INTEGER FUNCTION SUM(X,N)
       INTEGER J, N
       REAL X(N), Z
       Z = 0
       DO 10 J = 1,N
           Z = Z +X(J)
10     CONTINUE
       SUM = Z
       RETURN
       END
       INTEGER SUM
       REAL A(4), B(4)
       READ*, A, B
       PRINT*, SUM (A,2)/SUM(B,3)
       END
```

Assume the input for the program is:

```
4 5 3 4 2 1 1 0
```

```
14.    SUBROUTINE EXCESS(RESULT, OPA, OPB, N)
       INTEGER OPA(10), OPB(10), RESULT(10), CARRY
       CARRY = 0
       DO 10 K = N,1,-1
          RESULT(K+1) = MOD(OPA(K)+OPB(K)+CARRY,10)
          CARRY = (OPA(K)+OPB(K)+CARRY) / 10
10     CONTINUE
       RESULT(1) = CARRY
       RETURN
       END
       INTEGER A(10), B(10), C(10)
       READ*, N
       READ*, (A(K),K=1,N)
       READ*, (B(K),K=1,N)
       CALL EXCESS(C,A,B,N)
       PRINT*,(C(K), K=1,N+1)
       END
```

Assume the input for the program is:

```
7
4 5 6 7 0 9 4
8 3 7 5 2 0 8
```

```
15.    SUBROUTINE INTER(A, NA, B, NB, C, NC)
       INTEGER NA, NB, A(NA), B(NB), C(NA), K, M, NC
       NC = 0
       DO 10 K = 1, NA
          DO 20 M = 1, NB
             IF (A(K).EQ. B(M)) THEN
                NC = NC + 1
                C(NC) = A(K)
                GOTO 10
             ENDIF
20        CONTINUE
10     CONTINUE
       RETURN
       END
       INTEGER X(9), Y(9), Z(9), L, NX, NY, NZ
       READ*, NX, (X(L), L = 1,NX)
       READ*, NY, (Y(L), L = 1,NY)
       CALL INTER (X,NX,Y,NY,Z,NZ)
       PRINT*, (Z(J), J = 1,NZ)
       END
```

Assume the input for the program is:

```
5 12 23 45 65 67 84
4 84 64 12 21
```

2. The following program segments may or may not have errors. For each one of the segments, identify the errors(if any). Assume the following declarations :

```
       INTEGER M(4)
       LOGICAL L
```

```
a.     DO 5 K = 2,5,2
          READ*, M(K-1)
5      CONTINUE
```

Assume the input for the program is:

```
20,40,50,30,60
```

```
b.     DO 10 K = 1,4
           M(K+1) = -K
10     CONTINUE
       END
```

3. Consider the following subroutine :

```
       SUBROUTINE CHECK(A,B,C,N)
       INTEGER A(10), B(5)
       C = 0
       DO 10 M = 1,N
           C = C + A(M)*B(M)
10     CONTINUE
       RETURN
       END
```

If the only declaration and assignment statement in the main program are the following:

```
       INTEGER X(5), M(10), A
       A = 3
```

Which of the following **CALL** statements is correct assuming that X and M have some value ?

```
A)     CALL CHECK(M,X,C)
```

```
B)     CALL CHECK(M(10),X(5),C,5)
```

```
C)     CALL CHECK(M,X,B,A+2)
```

```
D)     CALL CHECK(M,X,N,A)
```

```
E)     CALL CHECK
```

4. The following function returns TRUE if the integer number X is found in an integer array A which has N elements. It returns FALSE otherwise. Complete the missing line.

```
       LOGICAL FUNCTION FOUND(A, X, N)
       INTEGER N, A(N), X, K
       DO 20 K=1,N
           IF(A(K) .EQ. X) THEN
               FOUND = .TRUE.
               ---------------
           ENDIF
20     CONTINUE
       FOUND = .FALSE.
       RETURN
       END
```

5. The following subroutine has 4 parameters: A, N, X and Y, where A is an integer array of size N and X and Y are integer numbers. The subroutine changes each element of A that has the value X by the value Y. Complete the missing line.

```
       SUBROUTINE CHANGE(A, N, X, Y)
       INTEGER N, A(N), X, Y, K
       DO 20 K=1,N
           IF(A(K) .EQ. X) THEN
               -----------------------
           ENDIF
20     CONTINUE
       RETURN
       END
```

6. Write a program to initialize a real 1-D array SERIES with the first 8 terms of the series 1, 4, 16, 64, ...
7. Write a logical function subprogram ZERO that takes a 1-D integer array LIST of size 5 and checks if all the elements of array LIST are zero. Write a main program to test the function.
8. Write a program to read a 1-D integer array X and check if all the elements of array X are in increasing order. Print a proper message.
9. Write a subroutine REVRSE to reverse a 1-D real array DAT with 5 elements. Write a main program to test the subroutine.
10. Write a program which reads the elements of three 1-Dimensional arrays A, B and C each of size N (where N<10). The program stores these elements in an array D of size M (where M = 3×N) such that the elements of D array will be as follows :

    A(1) B(1) C(1) A(2) B(2) C(2) ... A(N) B(N) C(N)
11. Write a program that reads a 1-D integer array of 10 elements and prints the element that appears the maximum number of times. (If there is more than one element, it prints the first one only).
12. Write a program to read a 1-D array AR1 of size 15 and another 1-D array AR2 of size 75. The program then finds and prints the number of occurrences of the array AR1 in the array AR2.
13. Write a program that reads ten integers and stores them into a one-dimensional array X.. The main program then calls a subroutine SUMS passing it the one-dimensional array. The subroutine computes the sum S of all the ten elements and the sum of the square of these ten values. Finally the main program prints the sum S and the sum of the squares S2.

6.8 Solutions to Exercises

Ans 1.

1. 1 3 9
2. DIFFERENT ARRAYS
3. 1600 900 400 100
4. 6 9 2 3
5. DIFFERENT ARRAYS
6.
```
  1   2
  6   7   8
 11  12  13  14
  3   9  15
```
7. 120
8. X
XX
XXX
9. 20 20 13 13 13

10. F F T F T
11. 13.0
12. 3.0
    6.0
    2.5
    2.0
13. 2
14. 1 2 9 4 2 3 0 2
15. 12

Ans 2.

a) End of file encountered (The program needs 2 lines of input)
b) Subscript out of range; m(5) is undefined

Ans 3.

C

Ans 4.

```
      RETURN
```

Ans 5.

```
      A(K) = Y
```

Ans 6.

```
      REAL SERIES(8)
      INTEGER K
      DO 12 K = 1, 8
          SERIES(K) = 4**(K-1)
12    CONTINUE
      END
```

Ans 7.

```
      LOGICAL FUNCTION ZERO(LIST, N)
      INTEGER N, LIST(N), K
      ZERO = .TRUE.
      K = 0
18    IF (K .LE. N .AND. ZERO) THEN
          IF(LIST(K) .NE. 0) ZERO = .FALSE.
          K = K + 1
          GOTO 18
      ENDIF
      RETURN
      END
      LOGICAL ZERO
      INTEGER LIST(5)
      IF (ZERO(LIST, 5)) THEN
          PRINT*, 'ALL ELEMENTS ARE ZEROS'
      ELSE
          PRINT*, 'NOT ALL ELEMENTS ARE ZEROS'
      ENDIF
      END
```

Ans 8.

```
      INTEGER X(3)
      READ*, X
      IF(X(1) .LT. X(2) .AND. X(2) .LT. X(3)) THEN
          PRINT*, 'INCREASING ORDER'
      ELSE
          PRINT*, 'NOT INCREASING ORDER'
      ENDIF
      END
```

Ans 9.

```
      SUBROUTINE REVERSE (DAT)
      REAL DAT(5), TEMP
      TEMP   = DAT(5)
      DAT(5) = DAT(1)
      DAT(1) = TEMP
      TEMP   = DAT(2)
      DAT(2) = DAT(4)
      DAT(4) = TEMP
      RETURN
      END
      REAL DAT(5)
      READ*, DAT
      CALL REVERSE(DAT)
      PRINT*, DAT
      END
```

Ans 10.

```
      INTEGER A(10) , B(10) , C(10) , D(30), N, M, K, J
      READ*, N
      M = 3 * N
      J = 1
      READ*, (A(K), K= 1 ,N),(B(K),K=1,N),(C(K),K=1,N)
      DO 10 K = 1 , N
          D(J)   = A(K)
          D(J+1) = B(K)
          D(J+2) = C(K)
          J = J + 3
10    CONTINUE
      PRINT*, (D(K) , K = 1 ,M)
      END
```

Ans 11.

```
      INTEGER A(10) , FREQ(10) , MAXFRQ , LOC, I, J
      READ*, A
      DO 10 I = 1 ,10
          FREQ(I) = 0
10    CONTINUE
      DO 20 I = 1 ,10
          DO 30 J = 1 ,10
              IF(A(J) .EQ. A(I)) FREQ(I) = FREQ(I) + 1
30        CONTINUE
20    CONTINUE
      MAXFRQ = FREQ(1)
      LOC  = 1
      DO 40 J = 1 ,10
          IF(MAXFRQ .LT. FREQ(J)) THEN
              MAXFRQ = FREQ(J)
              LOC = J
          ENDIF
40    CONTINUE
      PRINT*, ' THE ELEMENT WITH IS MAX APPEARANCE IS ',A(LOC)
      END
```

Ans 12.

```
      INTEGER COUNT , AR1(15),AR2(75), K, COUNT, M
      LOGICAL FOUND
      READ*,AR1
      READ*,AR2
      COUNT = 0
      DO 10 K=1,61
          FOUND = .TRUE.
          DO 20 M = K,K+14
              IF(AR1(M-K+1).NE. AR2(M)) FOUND=.FALSE.
20        CONTINUE
          IF(FOUND) COUNT = COUNT+1
10    CONTINUE
      PRINT*,'COUNT = ' , COUNT
      END
```

Ans 13.

```
      INTEGER X(10) , S , S2, J
      READ*, (X(J), J =1,10)
      CALL SUMS(X , S ,S2)
      PRINT*, ' THE SUM OF VALUES =', S
      PRINT*, ' THE SUM OF THE SQUARE OF VALUES =', S2
      END
      SUBROUTINE SUMS (X , S ,S2)
      INTEGER X(10) , S , S2, K
      S = 0
      S2 = 0
      DO 20 K = 1 ,10
          S = S + X(K)
          S2 = S2 + X(K) ** 2
20    CONTINUE
      RETURN
      END
```