7 TWO-DIMENSIONAL ARRAYS

A two-dimensional array (2-D array) is a tabular representation of data consisting of rows and columns. A two-dimensional array of size $m \times n$ represents a matrix consisting of $m$ rows and $n$ columns. Figure 1 shows a two-dimensional array $X$ of size $2 \times 3$. An element in a two-dimensional array is addressed by its row and column, for example, $X(2,1)$ refers to the element in row 2 and column 1 which has a value 6.

![Figure 1: A two-dimensional array $X$ of size $2\times3$](image)

Two-dimensional arrays can be pictured as a group of one-dimensional arrays. If we consider a one-dimensional array as a column, then a two-dimensional array $X$ of size $2 \times 3$ can be considered as consisting of three one-dimensional arrays; each one-dimensional array containing 2 elements. In fact, since each location in the memory has a single address, the computer stores a two-dimensional array as a one-dimensional array with column 1 first, followed by column 2 and so on. Figure 2 shows the storage of array $X$ (Figure 1) in the memory.

![Figure 2: Storage of the Two-Dimensional Array $X$ in Memory](image)

7.1 Two-Dimensional Array Declaration

Two-dimensional arrays must be declared using declaration statements like INTEGER, REAL etc. or the DIMENSION statement. The array declaration consists of the name of the array followed by the number of rows and columns in parentheses. This information in the declaration statements is required in order to reserve memory space.
For example, if an array X is declared with 2 rows and 3 columns, there are six elements in the array. Therefore, six memory locations must be reserved for such an array.

**Example 1**: Declaration of an integer array MAT consisting of 3 rows and 5 column.

```plaintext
INTEGER MAT(3,5)
```

**Example 2**: Declaration of a character array CITIES that consists of 9 elements in 3 rows and 3 columns and each element is of size 15.

```plaintext
CHARACTER CITIES (3,3) * 15
```

**Example 3**: Declaration of arrays using the DIMENSION statement.

```plaintext
DIMENSION X(10,10), M(5,7), Y(4,4)
```

In this example, arrays M and Y are of type REAL. Array X is of type INTEGER. Note that the type of arrays M and Y is specified in the two declaration statements. The type of Y is not specified and is taken as REAL by default.

**Example 4**: More array declarations: Consider the following declarations:

```plaintext
DIMENSION C(10,10), NUM(0:2, -2:1), VOL(4,2)
INTEGER ID(3,3)
REAL MSR(100,100), Z(4:7,8)
CHARACTER WORD(5,5)*3, C
LOGICAL TF(5,7)
```

Arrays ID, NUM are integer arrays. Arrays MSR, VOL, Z are real arrays. Array ID has a total of 9 elements in its 3 rows and 3 columns. The starting subscript value of row and column of each array is assumed to be 1 unless it is specified otherwise. In the declaration of arrays NUM and Z, the starting subscript is different than 1. Array NUM has 12 elements with rows numbered as 0, 1, 2; and columns numbered as -2, -1, 0, 1. Array Z has 32 elements with rows numbered from 4 up to 7 and columns numbered from 1 up to 8. Array WORD is a character array that has 5 rows and 5 columns, and stores 3 characters in each element. Array C is a character array and can store 1 character in each of its 100 elements (10 rows and 10 columns). Array TF is a logical array with 35 elements in 5 rows and 7 columns; each can store either a .TRUE. or a .FALSE. value.

### 7.2 Two-Dimensional Array Initialization

A two-dimensional array can be initialized in two possible ways. We can initialize either by rows or by columns. Initializing row after row is known as row-wise initialization. Similarly, initializing column after column is known as column-wise initialization. Remember, a two-dimensional array is always stored in the memory as a one-dimensional array column by column. The initialization may be done using assignment statements or **READ** statements.

#### 7.2.1 Initialization Using the Assignment Statement

**Example 1**: Declare an integer array ID consisting of 3 rows and 3 columns and initialize array ID row-wise as an identity matrix (i.e. all elements of the main diagonal must be 1 and the rest of the elements must be 0).
Solution:

```fortran
INTEGER ID(3,3), ROW, COL
C INITIALIZING ROW-WISE
   DO 17 ROW = 1, 3
      DO 17 COL = 1, 3
         IF (ROW .EQ. COL) THEN
            ID(ROW, COL) = 1
         ELSE
            ID(ROW, COL) = 0
         ENDIF
   17 CONTINUE
```

In this example, nested do loops are used. In fact, we need the nested loops to go to each element of a two-dimensional array. Note here that the index of the outer do loop is ROW which is also the row subscript of array ID. The inner loop index COL corresponds to the columns (the use of the variables ROW and COL has no significance; we could have used any other INTEGER variables). Notice how the value of COL varies within each iteration of the outer loop. When the value of ROW is 1, COL changes its value in the following sequence: 1, 2, 3, and 4. This means the first row has been initialized. Similarly, the next two rows are initialized. Since we initialized row after row, the array ID is initialized row-wise.

In general, if the outer loop index is the row subscript, then we are moving row-wise inside the array. Similarly, if the outer loop index is the column subscript, then we are moving column-wise inside the array.

**Example 2**: Declare a real array X consisting of 2 rows and 3 columns and initialize array X column-wise. Each element of array X should be initialized to its row number.

Solution:

```fortran
REAL X(2,3)
INTEGER J, K
C INITIALIZING COLUMN-WISE
   DO 27 J = 1, 3
      DO 27 K = 1, 2
         X(K, J) = K
      27 CONTINUE
```

### 7.3 Initialization Using the READ statement

As was the case in one-dimensional arrays, a two-dimensional array can be read as a whole or in part. To read the entire array, we may just use the name of the array without subscripts. In such case, the array is read column-wise. We can read part of an array by specifying specific elements of the array in the READ statement. We can either read row-wise or column-wise. Remember that 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 MATRIX of size 3×3 column-wise (i.e. the first element of input data is the first element of the first column of MATRIX, the second element of input data is the second element of the first column, the third element of input is the third element of the first column, the fourth element of input is the first element of the second column, and so on).
The input data is given as follows:

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

The contents of array MATRIX after reading the input data is as follows:

<table>
<thead>
<tr>
<th>3</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Solution 1: (Without Array Subscripts)**

```plaintext
INTEGER MATRIX(3, 3)
C READING COLUMN-WISE
READ*, MATRIX
```

**Solution 2: (Using Implied Loops)**

```plaintext
INTEGER MATRIX(3, 3), J, K
C READING COLUMN-WISE
READ*, ((MATRIX(K,J), K = 1, 3), J =1, 3)
```

**Solution 3: (Using DO and Implied Loop)**

```plaintext
INTEGER MATRIX(3, 3), J, K
C READING COLUMN-WISE
DO 28 J = 1, 3
   READ*, (MATRIX(K,J), K = 1, 3)
   CONTINUE
```

In all the three solutions, the array MATRIX is read column-wise. In Solution 1, the array MATRIX is read without any subscripts. In such cases, the computer reads the array column-wise, since all arrays are stored in the memory column-wise. In Solution 2, the outer loop index is J which corresponds with the column. Hence, the array is read column-wise. In Solution 3, the outer loop index is also J and, therefore, the array is read column-wise. The difference between the three solutions is that in Solution 1 and 2, only one READ statement is executed and, therefore, only one input line of data is required. If the input data is not given in one line, then data is read from the next line or the one after, until all data is read. In Solution 3, since three READ statements are executed, a minimum of three lines of input data is required.

**Example 2:** Read all the elements of an integer array X of size 3×5 row-wise (i.e. the first element of input data is the first element of the first row of array X, the second element of input is the second element of the first row, the third element of input is the third element of the first row, the fourth element of input is the fourth element of the first row, the fifth element of input is the fifth element of the first row, the sixth element of input is the first element of the second row and so on).

The input data is given as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>5</th>
<th>9</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

The contents of array X after reading the input data is as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>5</th>
<th>9</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
Solution 1: (Using Implied Loops)

```
INTEGER X(3, 5), J, K
READ*, ((X(K, J), J = 1, 5), K = 1, 3)
```

Solution 2: (Using DO and an implied Loop)

```
INTEGER X(3, 5), J, K
C READING COLUMN-WISE
DO 33 K = 1, 3
   READ*, (X(K,J), J = 1, 5)
33 CONTINUE
```

In both solutions, the array X is read row-wise, since the outer loop index is K which corresponds to the row of array X. The difference between the two solutions is that in Solution 1, only one READ statement is executed and, therefore, only one input line of data is required. If the input data is not given in one line, then data is read from the next line or the one after, until all data is read. In Solution 2, since three READ statements are executed, a minimum of three lines of input data is required.

7.4 Printing Two-Dimensional Arrays

Just as in the case of reading a two-dimensional array, printing an array without subscripts will produce the whole array as output. In such a case, the array is printed column-wise. 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 a 3 × 3 integer array WHT column-wise and print:

i. the entire array row-wise in one line;
ii. the entire array column-wise in one line;
iii. one row per line;
iv. one column per line;
v. the sum of column 3;
Complete Examples on Two-Dimensional Arrays

Example 1: More on Reading Two-Dimensional Arrays: Write a FORTRAN program that reads a two dimensional array of size $5 \times 4$ row-wise. Each value is read from a

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The contents of WHT after reading are as follows:

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

The output of the program is as follows:

PRINTING THE ENTIRE ARRAY ROW-WISE

| 5 | 3 | 4 | 2 | 1 | 6 | 0 | 8 | 7 |

PRINTING THE ENTIRE ARRAY COLUMN-WISE

| 5 | 2 | 0 | 3 | 1 | 8 | 4 | 6 | 7 |

PRINTING ONE ROW PER LINE

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

PRINTING ONE COLUMN PER LINE

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

SUM OF COLUMN 3 IS 17

7.5 Complete Examples on Two-Dimensional Arrays

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

Example 1: More on Reading Two-Dimensional Arrays: Write a FORTRAN program that reads a two dimensional array of size $5 \times 4$ row-wise. Each value is read from a
separate line of input. The program then prints the same array column-wise such that the elements of the first column are printed on the first line of output and the elements of the second column are printed on the second line of output and so on.

Solution:

```
INTEGER TDIM(5, 4), ROW, COL
DO 10 ROW = 1, 5
  DO 12 COL = 1, 4
    READ*, TDIM(ROW, COL)
  12 CONTINUE
10 CONTINUE
DO 30 COL = 1, 4
  PRINT*, (TDIM(ROW, COL), ROW = 1, 5)
30 CONTINUE
END
```

Let us first consider the reading segment. Reading is done using two nested loops. The outer loop index corresponds to the rows of the two-dimensional array. The inner one corresponds to the columns. Hence, the array TDIM is read row-wise. Note that the READ statement is executed 20 times and therefore 20 input lines are required with one data value per line.

In the printing segment, we used an implied loop inside a DO loop. Remember that we were asked to print each column on one line of output. This tells us that each column must be printed using one and only one PRINT statement. Using two nested DO loops will cause each element to be printed on a separate line. Therefore, we used an implied loop for the elements of the columns. Consider the case of the first column. The value of COL is fixed to 1 by the DO loop whereas the value of ROW in the implied loop varies from 1 to 5 covering all the elements of the first column. The same logic applies to the rest of the columns.

Consider next the following segment as a substitute for the reading segment in the above program.

```
READ*, ((TDIM(ROW, COL), COL = 1, 4), ROW = 1, 5)
```

In the previous reading segment, we used nested DO loops and the data values were given one in each line. Here, we use nested implied loops. When using nested implied loops, the values can be provided either on one line or on multiple lines. This results from the fact that in the nested DO loops, we execute $5 \times 4 = 20$ READ statements and each statement takes input from a different line. In the nested implied loops, we execute only one READ statement.

In general, the index of the outer loop indicates the way the array is read or printed. If the outer loop index represents the row, the array is read or printed row-wise. If the outer loop index represents the column, the array is read or printed column-wise.

**Example 2:** Summation of Even Numbers in a Two-Dimensional Array: Write a FORTRAN program that reads a two-dimensional array of size $3 \times 4$ column-wise. It then computes and prints the sum of all even numbers in the array.
Solution:

```fortran
INTEGER A(3, 4), SUM, J, K
READ*, ((A(K, J), K = 1, 3), J = 1, 4)
SUM = 0
DO 1 K = 1, 3
    DO 2 J = 1, 4
        IF (MOD(A(K, J), 2) .EQ. 0) THEN
            SUM = SUM + A(K, J)
        ENDIF
    2 CONTINUE
1 CONTINUE
PRINT*, SUM
END
```

In this example, after reading the array column-wise, we go to each element of the array A using the nested DO loops. The intrinsic function MOD is used to check if the remainder is zero when each element is divided by two. Only those elements in the array which return a zero value for the function MOD are added to the variable SUM.

**Example 3 : Manipulating Two-Dimensional Arrays**

Write a FORTRAN program that reads a two-dimensional array of size 3 x 3 row-wise. The program finds the minimum element in the array and changes each element of the array by subtracting the minimum from each element. Print the updated array row-wise in one output line.

Solution:

```fortran
INTEGER A(3, 3), MIN, J, K
READ*, ((A(K, J), J = 1, 3), K = 1, 3)
MIN = A(1, 1)
DO 3 K = 1, 3
    DO 2 J = 1, 3
        IF (A(K, J) .LT. MIN) THEN
            MIN = A(K, J)
        ENDIF
    2 CONTINUE
3 CONTINUE
DO 4 K = 1, 3
    DO 3 J = 1, 3
        A(K, J) = A(K, J) - MIN
    3 CONTINUE
4 CONTINUE
PRINT*, ((A(K, J), J = 1, 3), K = 1, 3)
END
```

The array A cannot be changed unless the minimum element in the array is found. All the elements in the array are checked for the minimum element in the first nested DO loop. The array is updated in the second nested DO loop by replacing each element of the array by subtracting the minimum from that element.

### 7.6 Two-Dimensional Arrays and Subprograms

Two-dimensional arrays can be passed to a subprogram or can be used locally within the subprogram. Unlike one-dimensional arrays, it is not recommended to pass a variable-sized two-dimensional array to a subprogram (even though this does not produce an error, it may give wrong results). Whenever a two-dimensional array is passed to a subprogram, the row and column size of the array may be declared using a constant in both the main and the subprogram.
Example 1: Counting Zero Elements: Read a $3 \times 2$ integer array $MAT$ row-wise. Using a function $COUNT$, count the number of elements in $MAT$ with the value equal to 0.

Solution:

```fortran
INTEGER MAT(3,2), COUNT, J, K
READ*, (MAT (K, J), J = 1, 2), K =1, 3)
PRINT*, 'COUNT OF ELEMENTS WITH VALUE 0 IS ', COUNT (MAT)
END

INTEGER FUNCTION COUNT(MAT)
INTEGER MAT(3,2), J, K
COUNT = 0
DO 77 K = 1, 3
   DO 77 J = 1, 2
      IF (MAT(K, J) .EQ. 0) COUNT = COUNT + 1
77 CONTINUE
RETURN
END
```

The input of the program is

```
12, 0, 1, 9, 2, 0
```

The output of the program is as follows:

```
COUNT OF ELEMENTS WITH VALUE 0 IS 2
```

In this example, another possibility is to call the function $COUNT$ by passing three arguments: $MAT$, $M$ and $N$ where $M$ and $N$ are the variables representing the row and the column size of array $MAT$. The declaration of $MAT$ within the function $COUNT$ may then be given as follows: INTEGER $MAT(M,N)$. This type of variable-sized two-dimensional array declaration is allowed in a subprogram. However, the use of such declarations is not recommended due to reasons beyond the scope of this book.

Example 2: Addition of Matrices: Write a subroutine $CALC(A, B, C, N)$ that receives 2 two-dimensional arrays $A$ and $B$ of size $10 \times 10$. It returns the result of adding the two arrays (matrices) in another array $C$ of the same size.

Solution:

```fortran
SUBROUTINE CALC(A, B, C, N)
INTEGER A(10,10), B(10,10), C(10,10), N
DO 10 K = 1,N
   DO 15 J = 1,N
      C(K,J) = A(K,J) + B(K,J)
15 CONTINUE
10 CONTINUE
RETURN
END
```

7.7 Common Errors in Array Usage

We have already seen errors that may occur in the use of one-dimensional arrays in the previous chapter. Such errors can occur in using two-dimensional arrays as well. The following errors are commonly seen while using arrays:

1. Array declaration is missing: All arrays must be declared. Otherwise, a message would appear as 'FUNCTION array name IS NOT DEFINED.' Since the array declaration is missing, the computer assumes it to be a function. Therefore, the misleading message appears.
2. Array subscript is out-of-bounds: This error occurs when an array subscript is outside the range of the array elements. For example, for a one-dimensional array X declared as `INTEGER X(10)`, the expression `X(12)` would produce an error. Similarly, in a 2-D array Y declared as `INTEGER Y(-3:2, 5)`, the expression `Y(-5,1)` would produce an error.

3. Array subscript is not an integer: All array subscripts must be integers. This error occurs when an array subscript is real. For example, for a one-dimensional array X declared as `INTEGER X(10)`, the expression `X(2.0)` would produce an error. Similarly, in a 2-D array Y of size `3×2`, an expression `Y(1,3.0)` would produce an error.

4. Array size is a variable in the main program: All array sizes must be integer constants, if the array is declared in the main program. This error occurs when an array subscript is a variable. For example, a one-dimensional array X declared in a main program as `INTEGER X(N)` would produce an error. In a subprogram, a declaration such as `INTEGER X(N)` is valid as long as both X and N are dummy arguments. Similar declarations can be made for two-dimensional arrays as long as the array name, its column size and its row size are dummy arguments. Such declarations (for example `INTEGER Y(M,N)`) are valid in a subprogram but may not be used due to reasons beyond the scope of this book.

7.8 Exercises

1. What is printed by the following programs?

   ```fortran
   1. INTEGER X(3,3), J
      READ*, X
      PRINT*, (X(J,J), J = 1, 3)
      PRINT*, (X(J,3), J = 1, 3)
      END
   
   Assume the input is:
   1, 5, 7
   7, 5, 1
   3, 8, 9
   ```

   ```fortran
   2. REAL B(2,3), F
      INTEGER J, K
      F(X, Y) = X + Y * 2
      READ*, ((B(J,K), K = 1, 2), J = 1, 2)
      DO 2 J = 1, 2
           B(J,3) = F(B(J,1), B(J,2))
      2 CONTINUE
      PRINT*, B
      END
   
   Assume the input is:
   10, 20, 30, 40
   ```
3. **SUBROUTINE** ADD(A, B, C)
   INTEGER A(2,2), B(2,2), C(2,2), J, K
   DO 33 J = 1, 2
       DO 22 K = 1, 2
           C(J,K) = A(J,K) + B(J,K)
       22 CONTINUE
   33 CONTINUE
   RETURN
END

Assume the input is:

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

4. **INTEGER** A(3,3), J, K
   **READ***, (A(K,J), K=1,3), J=1,3
   **PRINT***, A
   **PRINT***, (A(K,J), J=1,2), K=1,3
   **PRINT***, A(3,2)
   **PRINT***, (A(K,2), K=3,1,-2)
END

Assume the input is:

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

5. **INTEGER** A(2,2), J, K
   **READ***, A
   **DO** 3 J = 1, 2
       **PRINT***, (A(J,K), K=1,2)
   3 **CONTINUE**
   **END**

Assume the input is:

```
1 2 3 4
```

6. **INTEGER** TDAR(3,3), ODAR(10), ROW, COL, J, K, M, N
   **NUM(M,N)** = M + N - 1
   **READ***, TDAR
   **READ***, ROW, COL
   **DO** 10 J = 1, 3
       **DO** 10 K = 1, 3
           **ODAR(NUM(J,K))** = TDAR(J,K)
       10 **CONTINUE**
   **PRINT***, ODAR(NUM(ROW,COL)), ODAR(NUM(COL,ROW))
**END**

Assume the input is:

```
9 6 4 3 2 1 8 5 7
2 3
```
7. \textbf{INTEGER} \ A(2,2), \ B(2,2), \ C(2,2), \ X, \ Y, \ K, \ M
\textbf{D(M,N)} = M + N
\textbf{READ*}, \ A, \ B
\textbf{DO} \ 35 \ K = 1,2
    \textbf{DO} \ 35 \ M = 1,2
        \textbf{X} = A(K,M)
        \textbf{Y} = B(K,M)
        \textbf{C(M,K)} = D(X,Y)
\textbf{35 \ CONTINUE}
\textbf{DO} \ 22 \ K = 1,2
    \textbf{PRINT*}, \ (C(K,M), \ M=1,2)
\textbf{22 \ CONTINUE}
\textbf{END}

Assume the input is:
3 7 2 6
5 8 4 1

8. \textbf{INTEGER} \ A(10,10), \ B(10), \ L, \ K, \ N
\textbf{READ*}, \ N, \ ((A(K,L),K=1,N),L=1,N), \ (B(K),K=1,N)
\textbf{PRINT*}, \ C(A,B,N)
\textbf{END}
\textbf{REAL \ FUNCTION} \ C(A,B,N)
\textbf{INTEGER} \ A(10,10),B(10), \ L, \ N
\textbf{C} = 0.0
\textbf{DO} \ 44 \ L = 1,N
    \textbf{IF} \ (L/3*3 .NE.L) \textbf{B(L)} = A(L,L)
    \textbf{C} = \textbf{B(L)} * A(L,L)
\textbf{44 \ CONTINUE}
\textbf{return}
\textbf{END}

Assume the input is:
3 1 1 1 2 2 2 3 3 3 4 4 4

9. \textbf{INTEGER} \ A(5,5), \ J, \ K, \ M, \ N
\textbf{READ*}, \ N, \ ((A(K,J),J=1,N),K=1,N)
\textbf{CALL \ TEST(A,N,M)}
\textbf{PRINT*}, \ M
\textbf{END}
\textbf{SUBROUTINE} \ TEST \ (X,Y,Z)
\textbf{INTEGER} \ X(5,5), \ Y, \ K, \ J
\textbf{Z} = X(1,1)
\textbf{DO} \ 10 \ K = 1,Y
    \textbf{DO} \ 10 \ J = 1, Y
        \textbf{IF} \ (Z.GT.X(K,J)) \textbf{Z}=X(K,J)
\textbf{10 \ CONTINUE}
\textbf{return}
\textbf{END}

Assume the input is:
3 1 3 6 -3 0 4 5 9 -1

2. Assume the array declaration:
\textbf{INTEGER} \ Z(10,10)

is given. Which of the following \textbf{READ} statements will read the array column-wise if the data is given one value per line?

I. \textbf{READ*}, \ Z
This Copy was edited & prepared by Husni Al-Muhtaseb as part of KFUPM open Course initiative
7.9 Solutions to Exercises

Ans 1.

1. \[1 5 7 7 5 1 3 8 9\]  
   \[1 5 9\]  
   \[3 8 9\]  
2. \[10.0 30.0 20.0 40.0 50.0 110.0\]  
3. \[10 10 14 3\]  
   \[17 14 19 4\]  
4. \[1 2 3 4 5 6 7 8 9\]  
   \[1 4 2 5 3 6\]  
   \[6\]  
   \[6 4\]  
5. \[1 3\]  
6. \[1 1\]  
7. \[8 15\]  
   \[6 7\]  
8. \[12.0\]  
9. \[-3\]

Ans 2.

I, II, III

Ans 3

1) \(L = 1, 4\)  
2) \(K + L \equiv 5\)  
3) 1 4) 0

Ans 4.

```fortran
REAL PRD(3,4)  
INTEGER J, K  
DO 10 K = 1, 3  
   DO 20 J = 1, 4  
      PRD(K, J) = K * J  
20 CONTINUE  
10 CONTINUE  
PRINT*, PRD  
END
```
Ans 5.

```fortran
SUBROUTINE IDINIT(IMAT)
INTEGER IMAT(3,3), J, K
DO 77 K = 1, 3
    DO 77 J = 1, 3
        IMAT(K, J) = 0
    IF (K .EQ. J) IMAT(K, J) = 1
77 CONTINUE
RETURN
END

INTEGER IMAT(3,3), K
READ*, IMAT
CALL IDINIT(IMAT)
DO 77 K = 1, 3
    PRINT*, IMAT(K,1),IMAT(K,2),IMAT(K,3)
77 CONTINUE
END
```

Ans 6.

```fortran
INTEGER X(3,4) , ROW(3) , COL(4), J, K
READ*, X
DO 55 K = 1, 3
    ROW(K) = 0
    DO 55 J = 1, 4
        ROW(K) = ROW(K) + X(K, J)
55 CONTINUE
DO 66 J = 1, 4
    COL(J) = 0
    DO 66 K = 1, 3
        COL(J) = COL(J) + X(K, J)
66 CONTINUE
PRINT*, ROW
PRINT*, COL
END
```

Ans 7.

```fortran
INTEGER CNT, I, J
REAL TAB(8,10)
DO 10 I = 1 ,8
    READ*, (TAB(I,J), J =1,10)
10 CONTINUE
CNT = 0
DO 20 I = 1 ,8
    DO 30 J = 1 ,10
        IF(INT(SQRT(TAB(I,J)))**2.EQ.TAB(I,J))CNT=CNT+1
30 CONTINUE
PER = CNT / 80.0 * 100
PRINT*, ' THE PERCENTAGE = ', PER
END
```
Ans 8.

```
INTEGER MAT(10,10), N, SUM, MAXSUM, COL, I, J
READ*, N
DO 10 I = 1, N
   READ*, (MAT(I,J), J = 1,N)
10    CONTINUE
SUM = 0
COL = 1
DO 20 K = 1, N
   SUM = SUM + MAT(K,I)
20    CONTINUE
MAXSUM = SUM
DO 30 J = 2, N
   SUM = 0
   DO 40 K = 1, N
      SUM = SUM + MAT(K,J)
40    CONTINUE
   IF(SUM .GT. MAXSUM) THEN
      MAXSUM = SUM
      COL = J
   ENDIF
30    CONTINUE
PRINT*, (MAT(K,COL), K = 1, N)
END
```