Array Processing – Part II
Multi-Dimensional Arrays

1. **What is a multi-dimensional array?**

   A multi-dimensional array is simply a table (2-dimensional) or a group of tables. The following is a 2-dimensional table. To find a student's score, we need

   (a) his/her PIN – **ROW** index;

   (b) a program number – **COLUMN** index.

	Prog 1	*Prog 2*	*Prog 3*
1	100	87	100
2	95	100	90
3	80	90	93
⋮	⋮	⋮	⋮
30	100	95	76

A three-dimensional array can be considered as a pile of two-dimensional tables. For example, all three sections of CS201 can be considered as a three dimensional table consisting of three sub-tables:

```
          Sec01                     Sec02                       Sec03
    Prog1 Prog2 Prog3         Prog1 Prog2 Prog3           Prog1 Prog2 Prog3
   +-----+-----+-----+       +-----+-----+-----+         +-----+-----+-----+
 1 | 100 |  98 |  87 |     1 |  78 | 100 |  99 |       1 |  86 |  85 | 100 |
 2 |  90 | 100 |  96 |     2 | 100 |  60 |  87 |       2 |  96 |  82 | 100 |
 3 |  86 | 100 |  78 |     3 |  65 |  76 |  99 |       3 | 100 | 100 |  95 |
 : |  :  |  :  |  :  |     : |  :  |  :  |  :  |       : |  :  |  :  |  :  |
30 | 100 |  86 |  96 |    30 |  88 |  98 | 100 |      30 |  79 |  94 |  97 |
   +-----+-----+-----+       +-----+-----+-----+         +-----+-----+-----+
```

Thus, to find a student's score, we need

(a) a section number – **TABLE** index;

(b) his/her PIN – **ROW** index;

(c) a program number – **COLUMN** index.

So, **THREE** indices are required.

2. **What are the important components of a multi-dimensional array?**

   - Its **type**: `INTEGER`, `REAL`, `LOGICAL` and `CHARACTER`
   - Its **extents**: Since a multi-dimensional array has more than one indices, more than one extents are required.

     Indices (or subscripts) must be integers. The smallest and the largest indices (or subscripts) are referred to as the *lower* bound and *upper* bound of that extent.

3. **How to declare an array?**

   It is exactly identical to declaring one-dimensioanl arrays. But, thre are more extents:

```
REAL, DIMENSION(-5:5, 10:15)    :: x
REAL, DIEMNSION(100:200, -1:1)  :: Sum
REAL, DIMENSION(1:10, 1:10)     :: Value
```

   The above declares three arrays:

   (a) Array `x` is a two-dimensional array. The first (*resp.*, second) extent has lower bound and upper bound `-5` and `5` (*resp.*, `10` and `15`).

   (b) Array `Sum` is a two-dimensional array. The first (*resp.*, second) extent has lower bound and upper bound `100` and `200` (*resp.*, `-1` and `1`).

(c) Array `Value` is a two-dimensional array. The first (*resp.*, second) extent has lower bound and upper bound `1` and `10` (*resp.*, `1` and `10`).

If the lower bound is `1`, the lower bound value and the colon can be removed.

4. **What are the array elements?**

```
REAL, DIMENSION(-5:5, 10:15) :: x
```

The first index gives the **ROW** number, while the second provides the **COLUMN** number. Thus, the `-5`*th* row of array `x()` is `x(-5,10)`, `x(-5,11)`, `x(-5,12)`, `x(-5,13)`, `x(-5,14)` and `x(-5,15)`.

The `0`*th* row of `x()` has elements `x(0,10)`, `x(0,11)`, `x(0,12)`, `x(0,13)`, `x(0,14)` and `x(0,15)`.

The `4`*th* row of `x()` has elements `x(4,10)`, `x(4,11)`, `x(4,12)`, `x(4,13)`, `x(4,14)` and `x(4,15)`.

Array Input and Output

You can read in array elements Row-by-Row, one on each line

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
IF (ROW < 1 .OR. ROW > MAXROW) THEN
   WRITE(*,*)  'Bad row number'
ELSE IF (COL < 1 .OR. COL > MAXCOL) THEN
   WRITE(*,*)  'Bad column number'
ELSE
   DO I = 1, ROW
      DO J = 1, COL
         READ(*,*)  X(I,J)
      END DO
   END DO
END IF
```

The input is:

```
2     3                        ROW = 2, COL = 3
   4                     Col 1 Col 2 Col 3 Col 4 Col 5
   8                    +-----+-----+-----+-----+-----+
   3              Row 1 |  4  |  8  |  3  |  ?  |  ?  |
   9              Row 2 |  9  |  7  |  5  |  ?  |  ?  |
   7              Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
   5                    +-----+-----+-----+-----+-----+
```

You can also read in array elements Column-by-Column, one on each line

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
IF (ROW < 1 .OR. ROW > MAXROW) THEN
   WRITE(*,*)  'Bad row number'
ELSE IF (COL < 1 .OR. COL > MAXCOL) THEN
   WRITE(*,*)  'Bad column number'
ELSE
   DO J = 1, COL <------------------- changed
      DO I = 1, ROW <---------------- changed
         READ(*,*)  X(I,J)
      END DO
   END DO
END IF
```

The input is the same, but the array content is <u>**VERY**</u> different.

```
 2     3                          ROW = 2, COL = 3
    4                       Col 1 Col 2 Col 3 Col 4 Col 5
    8                     +-----+-----+-----+-----+-----+
    3             Row 1 |  4  |  3  |  7  |  ?  |  ?  |
    9             Row 2 |  8  |  9  |  5  |  ?  |  ?  |
    7             Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
    5                     +-----+-----+-----+-----+-----+
```

Or, you can read in array elements Row-by-Row with an implied `DO`

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
IF (ROW < 1 .OR. ROW > MAXROW) THEN
   WRITE(*,*)  'Bad row number'
ELSE IF (COL < 1 .OR. COL > MAXCOL) THEN
   WRITE(*,*)  'Bad column number'
ELSE
   DO I = 1, ROW
      READ(*,*)  (X(I, J), J = 1, COL) <------ changed
   END DO
END IF
```

Now you have to organize the input row-by-row!

```
2     3                     ROW = 2, COL = 3
4 8 3                Col 1 Col 2 Col 3 Col 4 Col 5
9 7 5               +-----+-----+-----+-----+-----+
              Row 1 |  4  |  8  |  3  |  ?  |  ?  |
              Row 2 |  9  |  7  |  5  |  ?  |  ?  |
              Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
                    +-----+-----+-----+-----+-----+
```

Or, you can read array elements in Column-by-Column with an implied DO

```
INTEGER, PARAMETER  = MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
IF (ROW < 1 .OR. ROW > MAXROW) THEN
   WRITE(*,*)  'Bad row number'
ELSE IF (COL < 1 .OR. COL > MAXCOL) THEN
   WRITE(*,*)  'Bad column number'
ELSE
   DO J = 1, COL <------------------------ changed
      READ(*,*)  (X(I, J), I = 1, ROW) <------ changed
   END DO
END IF
```

You have to organize the input column-by-column!

```
2     3                                ROW = 2, COL = 3
4 9                              Col 1 Col 2 Col 3 Col 4 Col 5
8 7                             +-----+-----+-----+-----+-----+
3 5                      Row 1 |  4  |  8  |  3  |  ?  |  ?  |
                         Row 2 |  9  |  7  |  5  |  ?  |  ?  |
                         Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
                                +-----+-----+-----+-----+-----+
```

The following reads in the table column-by-column

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
IF (ROW < 1 .OR. ROW > MAXROW) THEN
   WRITE(*,*)  'Bad row number'
ELSE IF (COL < 1 .OR. COL > MAXCOL) THEN
   WRITE(*,*)  'Bad column number'
ELSE
   DO J = 1, COL <------------------------- changed
      READ(*,*)  (X(I, J), I = 1, ROW) <------ changed
   END DO
END IF
```

What if you **DID NOT** change your input accordingly? The following input is organized row-by-row:

```
 2     3                       ROW = 2, COL = 3
 4 8 3                   Col 1 Col 2 Col 3 Col 4 Col 5
 9 7 5                  +-----+-----+-----+-----+-----+
                  Row 1 |  4  |  9  |  ?  |  ?  |  ?  |
                  Row 2 |  8  |  7  |  ?  |  ?  |  ?  |
                  Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
                        +-----+-----+-----+-----+-----+
```

3 on the first line and 5 on the second are **IGNORED**, and there is no more data for the third column. OOPS!

Using Implied DO Loop to Read a Table

Syntax

1. The following is equivalent to

```
REAL, DIMENSION(1:3,1:5) :: A

READ(*,*)  ((A(I,J), J = 1, 5), I = 1, 3)
```

   The outer implied DO expands to

```
(A(1,J), =1,5),  (A(2,J),j=1,5),  (A(3,J),j=1,5)
```

   Expanding all three implied DOs yields:

```
READ(*,*)  A(1,1), A(1,2), A(1,3), A(1,4), A(1,5), &
           A(2,1), A(2,2), A(2,3), A(2,4), A(2,5), &
           A(3,1), A(3,2), A(3,3), A(3,4), A(3,5)
```

2. This is a row-by-row read:

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J
     ......
READ(*,*)   ROW, COL
READ(*,*)  ((X(I,J), J = 1, COL), I = 1, ROW)
```

```
2     3                              ROW = 2, COL = 3
4 8 3                          Col 1 Col 2 Col 3 Col 4 Col 5
9 7 5                         +-----+-----+-----+-----+-----+
                        Row 1 |  4  |  8  |  3  |  ?  |  ?  |
                        Row 2 |  9  |  7  |  5  |  ?  |  ?  |
                        Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
                              +-----+-----+-----+-----+-----+
```

3. This is a column-by-column read:

```
INTEGER, PARAMETER :: MAXROW = 3, MAXCOL = 5
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: X
INTEGER                               :: ROW, COL, I, J

     ......
READ(*,*)   ROW, COL
READ(*,*)  ((X(I,J), I = 1, ROW), J = 1, COL)
```

```
2     3                              ROW = 2, COL = 3
4 9                          Col 1 Col 2 Col 3 Col 4 Col 5
8 7                         +-----+-----+-----+-----+-----+
3 5                   Row 1 |  4  |  8  |  3  |  ?  |  ?  |
                      Row 2 |  9  |  7  |  5  |  ?  |  ?  |
                      Row 3 |  ?  |  ?  |  ?  |  ?  |  ?  |
                            +-----+-----+-----+-----+-----+
```

Using Arrays in Computation

1. Clear an array to zero:

```
INTEGER, PARAMETER :: MAXROW = 20, MAXCOL = 100
INTEGER, DIMENSION(1:MAXROW,1:MAXCOL) :: A
INTEGER                               :: i, j
DO i = 1, MAXROW
   DO j = 1, MAXCOL
      A(i,j) = 0
   END DO
END DO
```

2. Even rows have 1 and odd rows have 0

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A
INTEGER                             :: i, j
DO i = 1, BOUND
   IF (MOD(i,2) == 0) THEN
      DO j = 1, BOUND
         A(i,j) = 1
      END DO
   ELSE
      DO j = 1, BOUND
         A(i,j) = 0
      END DO
   END IF
END DO
```

3. Diagonal elements `A(1,1)`, `A(2,2)`, `A(3,3)`, ... receive 1 while others receive zero.

   (a) **Method 1**

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A
INTEGER                             :: i, j
DO i = 1, BOUND
   DO j = 1, BOUND
      IF (i == j) THEN
         A(i,i) = 1
      ELSE
         A(i,j) = 0
      END IF
   END DO
END DO
```

   (b) **Method 2**

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A
INTEGER                             :: i, j
DO i = 1, BOUND
   DO j = 1, BOUND
      A(i,j) = 0
   END DO
   A(i,i) = 1
END DO
```

4. Compute the sum of all array elements:

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A
INTEGER                             :: Sum, i, j
Sum = 0
DO i = 1, BOUND
   DO j = 1, BOUND
      Sum = Sum + A(i,j)
   END DO
END DO
```

5. Compute the sum of the corresponding elements of two arrays into a third one:

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A, B, C
INTEGER                             :: i, j
DO i = 1, BOUND
   DO j = 1, BOUND
      C(i,j) = A(i,j) + B(i,j)
   END DO
END DO
```

6. Clear the boarder of a table to 1 and interior elements to zero.

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(1:BOUND,1:BOUND) :: A
INTEGER                             :: i, j
DO i = 2, BOUND-1    <---+
   DO j = 2, BOUND-1     |
      A(i,j) = 0         |
   END DO                +- row 2 to row BOUND-1
   A(i,1)     = 1        |
   A(i,BOUND) = 1        |
END DO <-----------------+
DO j = 1, BOUND
   A(1,j)     = 1
   A(BOUND,j) = 1
END DO
```

7. Transpose a square table.

```
INTEGER, PARAMETER :: BOUND = 20
INTEGER, DIMENSION(BOUND,BOUND) :: A
INGETER                         :: i, j, Temp
DO i = 1, BOUND-1
   DO j = i+1, BOUND      1  2  3  4 -> 1  5  9 13
      Temp   = A(i,j)     5  6  7  8    2  6 10 14
      A(i,j) = A(j,i)     9 10 11 12    3  7 11 15
      A(j,i) = Temp      13 14 15 16    4  8 12 16
   END DO
END DO
```

Multi-Dimensional Arrays as Arguments

1. Multi-dimensional arrays can be passed to FUNCTIONSs and SUBROUTINEs. The corresponding formal arguments must be declared as arrays with the same number of extents and structure:

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER, PARAMETER :: L1 = 1, U1 = 10, L2 = -5, U2 = 5
   INTEGER, DIMENSION(L1:U1,L2:U2) :: x
      ......
   CALL Sub(x, L1, U1, L2, U2)
      ......
CONTAINS
   SUBROUTINE  Sub(a, Lo1, Up1, Lo2, Up2)
      IMPLICIT  NONE
      INTEGER, DIMENSION(Lo1:Up1,Lo2:Up2), INTENT(IN) :: a
      INTEGER, INTENT(IN)                             :: Lo1, Up1
      INTEGER, INTENT(IN)                             :: Lo2, Up2
         ......
   END SUBROUTINE  Sub
END PROGRAM  Example
```

2. Or, you may want to use **assumed-shape** arrays. In this case, only lower bounds are required.

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER, PARAMETER :: L1 = 1, U1 = 10, L2 = -5, U2 = 5
   INTEGER, DIMENSION(L1:U1,L2:U2) :: x
      ......
   CALL Sub(x, L1, L2)
      ......
CONTAINS
   SUBROUTINE  Sub(a, Lo1, Lo2)
      IMPLICIT  NONE
      INTEGER, DIMENSION(Lo1:,Lo2:), INTENT(IN) :: a
      INTEGER, INTENT(IN)                       :: Lo1
      INTEGER, INTENT(IN)                       :: Lo2
         ......
   END SUBROUTINE  Sub
END PROGRAM  Example
```

3. As a reminder, if a lower bound is 1, you do not have to pass it.

```
PROGRAM  Example
   IMPLICIT  NONE
   INTEGER, PARAMETER :: L1 = 1, U1 = 10, L2 = -5, U2 = 5
   INTEGER, DIMENSION(L1:U1,L2:U2) :: x
      ......
   CALL Sub(x, L2)
      ......
CONTAINS
   SUBROUTINE  Sub(a, Lo2)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:,Lo2:), INTENT(IN) :: a
      INTEGER, INTENT(IN)                     :: Lo2
         ......
   END SUBROUTINE  Sub
END PROGRAM  Example
```

Example 1: Table and Their Row/Column Sums

```
PROGRAM  TableSums
   IMPLICIT  NONE
   INTEGER, PARAMETER :: ROW_SIZE = 5, COLUMN_SIZE = 6
   INTEGER, DIMENSION(1:ROW_SIZE,1:COLUMN_SIZE) :: Table
   INTEGER, DIMENSION(1:ROW_SIZE)               :: RowSum
   INTEGER, DIMENSION(1:COLUMN_SIZE)            :: ColumnSum
   INTEGER                                  :: RowSize, ColumnSize

   CALL  ReadTable(Table, RowSize, ColumnSize)
   CALL  ComputeRowSum(Table, RowSize, ColumnSize, RowSum)
   CALL  ComputeColumnSum(Table, RowSize, ColumnSize, ColumnSum)
   CALL  DisplayTable(Table, RowSize, ColumnSize, RowSum, ColumnSum)

   CALL  ComputeSums(Table, RowSize, ColumnSize, RowSum, ColumnSum)
   CALL  DisplayTable(Table, RowSize, ColumnSize, RowSum, ColumnSum)

CONTAINS
   SUBROUTINE  ReadTable(Table, Rows, Columns)
      IMPLICIT   NONE
      INTEGER, DIMENSION(1:,1:), INTENT(OUT) :: Table
      INTEGER, INTENT(OUT)                   :: Rows, Columns
      INTEGER                                :: i, j

      READ(*,*)  Rows, Columns
      DO i = 1, Rows
         READ(*,*)  (Table(i,j), j=1, Columns)
      END DO
   END SUBROUTINE  ReadTable
```

```
SUBROUTINE  DisplayTable(Table, Rows, Columns, RowSum, ColumnSum)
   IMPLICIT   NONE
   INTEGER, DIMENSION(1:,1:), INTENT(IN) :: Table
   INTEGER, DIMENSION(1:), INTENT(OUT)   :: RowSum, ColumnSum
   INTEGER, INTENT(IN)                   :: Rows, Columns
   INTEGER                               :: i, j
   INTEGER                               :: Total

   WRITE(*,*)
   WRITE(*,*)  "The input table and row/column sums:"
   WRITE(*,*)
   Total = 0
   DO i = 1, Rows
      WRITE(*,*)  (Table(i,j), j=1, Columns),  RowSum(i)
      Total = Total + RowSum(i)
   END DO
   WRITE(*,*)  (ColumnSum(j), j=1, Columns), Total
END SUBROUTINE  DisplayTable

SUBROUTINE  ComputeRowSum(Table, Rows, Columns, RowSum)
   IMPLICIT   NONE
   INTEGER, DIMENSION(1:,1:), INTENT(IN) :: Table
   INTEGER, DIMENSION(1:), INTENT(OUT)   :: RowSum
   INTEGER, INTENT(IN)                   :: Rows, Columns
   INTEGER                               :: i, j

   DO i = 1, Rows
      RowSum(i) = 0
      DO j = 1, Columns
         RowSum(i) = RowSum(i) + Table(i,j)
      END DO
   END DO
END SUBROUTINE  ComputeRowSum
```

```fortran
   SUBROUTINE  ComputeColumnSum(Table, Rows, Columns, ColumnSum)
      IMPLICIT   NONE
      INTEGER, DIMENSION(1:,1:), INTENT(IN) :: Table
      INTEGER, DIMENSION(1:), INTENT(OUT)   :: ColumnSum
      INTEGER, INTENT(IN)                   :: Rows, Columns
      INTEGER                               :: i, j

      DO j = 1, Columns
         ColumnSum(j) = 0
         DO i = 1, Rows
            ColumnSum(j) = ColumnSum(j) + Table(i,j)
         END DO
      END DO
   END SUBROUTINE  ComputeColumnSum

   SUBROUTINE  ComputeSums(Table, Rows, Columns, RowSum, ColumnSum)
      IMPLICIT   NONE
      INTEGER, DIMENSION(1:,1:), INTENT(IN) :: Table
      INTEGER, DIMENSION(1:), INTENT(OUT)   :: RowSum, ColumnSum
      INTEGER, INTENT(IN)                   :: Rows, Columns
      INTEGER                               :: i, j

      DO j = 1, Columns
         ColumnSum(j) = 0
      END DO
      DO i = 1, Rows
         RowSum(i) = 0
         DO j = 1, Columns
            RowSum(i)    = RowSum(i) + Table(i,j)
            ColumnSum(j) = ColumnSum(j) + Table(i,j)
         END DO
      END DO

   END SUBROUTINE  ComputeSums
END PROGRAM  TableSums
```

Example 2: Matrix Multiplication

Given matrix $A = [a_{ik}]$ of ℓ rows and m columns and matrix $B = [b_{kj}]$ of m rows and n colums, their product $C = [c_{ij}]$ is computed as follows:

$$\begin{aligned} c_{ij} &= \sum_{k=1}^{m} a_{ik} b_{kj} \\ &= a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{im}b_{mj} \end{aligned}$$

where matrix C has ℓ rows and n columns. The following is an example:

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 5 & 7 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 7 & 15 & 23 & 31 \\ 10 & 22 & 34 & 46 \\ 19 & 43 & 67 & 91 \end{bmatrix}$$

where 34 on the row 2 and column 3 is computed as follows:

$$34 = 2 \times 5 + 4 \times 6$$

Note that $[2, 4]$ is row #2 of matrix A and $[5, 6]$ is column #3 of matrix B.

The number of columns of A and the number of rows of B must be equal in order to compute $A \cdot B$

```
PROGRAM  MatrixMultiplication
   IMPLICIT  NONE
   INTEGER, PARAMETER :: MAX_SIZE = 20
   INTEGER, DIMENSION(1:MAX_SIZE,1:MAX_SIZE) :: X, Y, Z
   INTEGER                                   :: RowX, ColumnX
   INTEGER                                   :: RowY, ColumnY
   INTEGER                                   :: RowZ, ColumnZ

   CALL ReadMatrix(X, RowX, ColumnX)
   WRITE(*,*) "Input Matrix A:"
   CALL DisplayMatrix(X, RowX, ColumnX)

   CALL ReadMatrix(Y, RowY, ColumnY)
   WRITE(*,*)
   WRITE(*,*) "Input Matrix B:"
   CALL DisplayMatrix(Y, RowY, ColumnY)
   IF (ColumnX /= RowY) THEN
      WRITE(*,*)  "ERROR: Cannot multiply"
   ELSE
      RowZ    = RowX
      ColumnZ = ColumnY
      CALL  Multiply(X, RowX, ColumnX, Y, RowY, ColumnY, Z)
      WRITE(*,*)
      WRITE(*,*)  "Matrix A*B:"
      CALL DisplayMatrix(Z, RowZ, ColumnZ)
   END IF

CONTAINS
   SUBROUTINE  ReadMatrix(A, Row, Column)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:,1:), INTENT(OUT) :: A
      INTEGER, INTENT(OUT)                   :: Row, Column
      INTEGER                                :: i, j

      READ(*,*)  Row, Column
      DO i = 1, Row
         READ(*,*)  (A(i,j), j = 1, Column)
      END DO
   END SUBROUTINE  ReadMatrix
```

```
   SUBROUTINE  DisplayMatrix(A, Row, Column)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:,1:), INTENT(IN) :: A
      INTEGER, INTENT(IN)                   :: Row, Column
      INTEGER                               :: i, j

      DO i = 1, Row
         WRITE(*,*)  (A(i,j), j = 1, Column)
      END DO
   END SUBROUTINE  DisplayMatrix

   SUBROUTINE  Multiply(A, RowA, ColumnA, B, RowB, ColumnB, C)
      IMPLICIT  NONE
      INTEGER, DIMENSION(1:,1:), INTENT(IN)  :: A, B
      INTEGER, DIMENSION(1:,1:), INTENT(OUT) :: C
      INTEGER, INTENT(IN)                    :: RowA, ColumnA
      INTEGER, INTENT(IN)                    :: RowB, ColumnB
      INTEGER                                :: Sum
      INTEGER                                :: i, j, k

      DO i = 1, RowA
         DO j = 1, ColumnB
            Sum = 0
            DO k = 1, ColumnA
               Sum = Sum + A(i,k)*B(k,j)
            END DO
            C(i,j) = SUm
         END DO
      END DO
   END SUBROUTINE  Multiply
END PROGRAM  MatrixMultiplication
```

Ordering of Array Elements

Although FORTRAN supports two-dimensional arrays, **IN MEMORY** a two dimensional array is always organized **COLUMNWISE**.

```
INTEGER, DIMENSION(1:3,1:4) :: X

+--------+--------+--------+--------+      +--------+
| X(1,1) | X(1,2) | X(1,3) | X(1,4) |      | X(1,1) + <--+
+--------+--------+--------+--------+      +--------+    |
| X(2,1) | X(2,2) | X(2,3) | X(2,4) |      | X(2,1) |    +-- Col. 1
+--------+--------+--------+--------+      +--------+    |
| X(3,1) | X(3,2) | X(3,3) | X(3,4) |      | X(3,1) | <--+
+--------+--------+--------+--------+      +--------+
                                           | X(1,2) | <--+
                                           +--------+    |
                                           | X(2,2) |    +-- Col. 2
                                           +--------+    |
                                           | X(3,2) | <--+
                                           +--------+
                                           | X(1,3) | <--+
                                           +--------+    |
                                           | X(2,3) |    +-- Col. 3
                                           +--------+    |
                                           | X(3,3) | <--+
                                           +--------+
                                           | X(1,4) | <--+
                                           +--------+    |
                                           | X(2,4) |    +-- Col. 4
                                           +--------+    |
                                           | X(3,4) | <--+
                                           +--------+
```