Two Dimensional Arrays

It is also possible for arrays to have two or more dimensions. The two-dimensional array is also called a matrix. A two- dimensional array is nothing but a collection of a number of one- dimensional arrays placed one below the other.

Declaration: The general form of declaring a 2D array is –

Data type variable_name[size][size];
Example- int marks[10][2];

Accessing Elements of an Array:

```
for ( i = 0 ; i <10 ; i++ )
 {
  for ( j = 0 ; j <= 1 ; j++ )
   {
     printf ( "\n Enter roll no. and marks" ) ;
     scanf ( "%d ", &marks[i][j]) ;
   }
 }
```

In **marks** the zeroth column contains the roll no. or the first column which contains the marks. Remember the counting of rows and columns begin with zero.

	Col 0	Col 1
row 0	1	50
row 1	2	49
row 2	3	47
…		
row 9	10	48

Note: In this sample program the array elements have been stored row wise and accessed row wise. However, you can access the array elements column wise as well. Traditionally, the array elements are being stored and accessed row wise.

Initializing a 2-Dimensional Array: The following is the general initialization statement-

```
int num[2][2] = { { 1234, 56 },
                  { 1212, 33 }
                } ;
```

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.
Thus **int num[ ][3] = { 64, 87, 93, 14, 61, 92 } ;** is perfectly acceptable. Whereas the following would never work -

```
int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;
```

Memory Map of a 2-Dimensional Array: In memory whether it is a one-dimensional or a two-dimensional array the array elements are stored in one continuous chain. The arrangement of array elements of a two-dimensional array in memory is shown below:

num[0][0]	num[0][1]	num[0][2]	num[1][0]	num[1][1]	num[2][2]
25	**45**	**58**	**57**	**14**	**65**
1000	**1002**	**1004**	**1006**	**1008**	**1010**

Passing 2-D Array to a Function: There are three ways in which we can pass a 2-D array to a function. These are illustrated in the following program-

```
/* Three ways of accessing a 2-D array */
 void main( )
 {
  int a[3][4] = {
                  1, 2, 3, 4,
                  5, 6, 7, 8,
                  9, 10, 11, 12
                 } ;
display ( a, 3, 4 ) ;
show ( a, 3, 4 ) ;
print ( a, 3, 4 ) ;
}                 // end of main()

display ( int *q, int row, int col )
 {
   int i, j ;
  for ( i = 0 ; i < row ; i++ )
  {
    for ( j = 0 ; j < col ; j++ )
      printf ( "%d ", * ( q + i * col + j ) ) ;
    printf ( "\n" ) ;
   }
   printf ("\n" ) ;
}
show ( int ( *q )[4], int row, int col )
{
  int i, j ;
  int *p ;
  for ( i = 0 ; i < row ; i++ )
  {
   p = q + i ;
   for ( j = 0 ; j < col ; j++ )
   printf ( "%d ", * ( p + j ) ) ;
   printf ( "\n" ) ;
   }
  printf ( "\n" ) ;
}
```

```
print ( int q[ ][4], int row, int col )
{
  int i, j ;
  for ( i = 0 ; i < row ; i++ )
   {
     for ( j = 0 ; j < col ; j++ )
      printf ( "%d ", q[i][j] ) ;
      printf ( "\n" ) ;
    }
   printf ( "\n" ) ;
}
```

Explanation:

- In the **display( )** function we have collected the base address of the 2-D array being passed to it in an ordinary **int** pointer. Then through the two **for** loops using the expression *** (q + i * col + j)** we have reached the appropriate element in the array. Suppose **i** is equal to 2 and **j** is equal to 3, then we wish to reach the element **a[2][3]**. Let us see how the expression *** (q + i * col + j)** gives this element-

1	2	3	4	5	6	7	8	9	10	11	12
65502	04	06	08	10	12	14	16	18	20	22	24

The expression *** (q + i * col + j)** becomes *** (65502 + 2 * 4 + 3)**. This turns out to be *** (65502 + 11).** Since **65502** is address of an integer, *** (65502 + 11)** turns out to be * (**65524)**. Value at this address is 12. This is indeed same as **a[2][3]**. A more general formula for accessing each array element would be:

*** (base address + row no. * no. of columns + column no.)**

- In the **show( )** function we have defined **q** to be a pointer to an array of 4 integers through the declaration**:** **int (*q)[4] ;**

To begin with, **q** holds the base address of the zeroth 1-D array. This address is then assigned to **p**, an **int** pointer, and then using this pointer all elements of the zeroth 1-D array are accessed. Next time through the loop when **i** takes a value 1, the expression **q + i** fetches the address of the first 1-D array. This is because, **q** is a pointer to zeroth 1-D array and adding 1 to it would give us the address of the next 1-D array. This address is once again assigned to **p**, and using it all elements of the next 1-D array are accessed.

- In the third function **print( )**, the declaration of **q** looks like this:

int q[ ][4] ;

This is same as **int (*q)[4]**, where **q** is pointer to an array of 4 integers. The only advantage is that we can now use the more familiar expression **q[i][j]** to access array elements. We could have used the same expression in **show( )** as well.

Array of Pointers:

The way there can be an array of **int**s or an array of **float**s, similarly there can be an array of pointers. Since a pointer variable always contains an address, an array of pointers would be nothing but a collection of addresses. The addresses present in the array of pointers can be addresses of isolated variables or addresses of array elements or any other addresses. All rules that apply to an ordinary array apply to the array of pointers as well. Consider the following program-

```
/* array of integer pointers */
    void main( )
     {
        int *arr[3] ;
        int x = 10, y = 5, z=8, i;
        arr[0] = &x ; arr[1] = &y ; arr[2] = &z ;
        for ( i = 0 ; i <=2 ; i++ )
        printf ( "%d ", * ( arr[i] ) ) ;
      }
```

……

Problem Set-2

1. Write a program to find the maximum & minimum element of a 5x5 matrix.
2. Write a program to add, multiply two 4x4 matrices.
3. Write a program to obtain the Determinant values of a 4x4 matrix.
4. Write a program to find the sum of diagonals of a 4x4 matrix.
5. Write a program to transpose a 4 x 4 matrix. The transpose of a matrix is obtained by exchanging the elements of each row with the elements of the corresponding column.
6. Write a program to sort all the elements of a 4 x 4 matrix.