# EXPERIMENT NO 9

## Multi-Dimensional Arrays

### Objectives:

In this lab students will learn:

- To know about multi-dimensional arrays
- To declare multi-dimensional arrays.
- To perform fundamental operations on 2D arrays.

### Equipment required:

- Dev-C++/Eclipse/Visual Studio installed in PC/Windows

### DISCUSSION

## 1. Pre-Lab

### Multi-Dimensional Arrays

In C++ we can use multi-dimensional arrays. The general form to declare a multidimensional array is,

data_type name[size1] [size2] ... [size_N];

For example, the following declaration creates a three-dimensional integer array of size 5, 10 and 4.

int numin3D [5][10][4].

### Two-Dimensional Array

The simplest form of the multi-dimensional array is the two-dimensional array. A two-dimensional array is a list of one-dimensional arrays. To declare a two-dimensional integer array of size (x, y) you would write something as follows:

Type arrayName [ x ][ y ];

Where type can be any valid C++ data type and arrayName will be a valid C++ identifier. A two-dimensional array can be think as a table or matrix, which will have x number of rows and y number of columns. A 2-dimensional array named “a”, which contains three rows, and four columns is shown below:

| | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Row 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Row 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |

Thus, every element in array “a” is identified by an element name of the form a [ i ][ j ], where “a” is the name of the array, and “i” and “j” are the subscripts that uniquely identify each element in “a”.

## Initializing Two-Dimensional Arrays

Multi-dimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a [3][4] = {
        {0, 1, 2, 3},     /* initializers for row indexed by 0 */
        {4, 5, 6, 7},      /* initializers for row indexed by 1 */
        {8, 9, 10, 11}   /* initializers for row indexed by 2 */
       };
```

The nested braces, which indicate the intended rows, are optional. The following initialization is equivalent to previous example;

```
int a [3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

## Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example,

```
int val = a[2][3];
```

Here the value of the row indexed 2 and the column indexed 3 will be assigned to the variable "val".

## Example of 2D arrays

```
#include <iostream>
using namespace std;

int main () {
   // an array with 5 rows and 2 columns.
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

   // output each array element's value
   for ( int i = 0; i < 5; i++ )
      for ( int j = 0; j < 2; j++ ) {

         cout << "a[" << i << "][" << j << "]: ";
         cout << a[i][j]<< endl;
      }

   return 0;
}
```

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

## 2. Post-Lab (Lab Tasks)

1. Write a C++ Program to display all elements of an initialized two-dimensional array.
2. Add and subtract two matrices, take values of matrices from user.
3. Find the Transpose of a 3x3 Matrix.
4. Find the Determinant of a 3x3 matrix.
5. Find the adjoint of 2x2 matrix.
6. Take 2, 3x3 matrices from user and display the multiplication result at the output.

END