C++ Course Notes | 2D Arrays | Week 6

Hello everyone welcome to the weekly lecture notes

Topics to be covered:

- Multidimensional Array
- 2D Array
- 2D Vector

Multidimensional Array Introduction

In C++, we can create " array of an array" which are known as a multidimensional array. It stores homogeneous data in a tabular form. Data in multidimensional arrays are stored in row-major order i.e. elements are filled in the current row before moving to the next row.

Syntax to declare an N-Dimensional array:

```
datatype array_name[size1][size2].....[sizeN];
```

A combination of multiple 1D arrays is known as 2D array.

Syntax to declare a 2D array:

```
datatype array_name[rows][columns];
```

where rows imply the number of rows needed for the 2D array and column implies the number of columns needed.

For example:

```
int arr[4][5];
```

Here, arr is a two-dimensional array. It can hold a maximum of 20 elements. Let us understand how.

We can think of this array as a table with 4 rows and each row has 5 columns as shown below.

arr[0][0]	**arr[0][1]**	**arr[0][2]**	**arr[0][3]**	**arr[0][4]**
arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]	arr[1][4]
arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]	arr[2][4]
arr[3][0]	arr[3][1]	arr[3][2]	arr[3][3]	arr[3][4]

In this array you can store the values as required. Suppose, in the above array you want to store 10 at every index, you can do so using the following code:

```
#include<iostream>
using namespace std;
void main(){
  int arr[4][5];
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 5; j++) {
      arr[i][j] = 10;
    }
  }
}
```

There are two methods to initialize two-dimensional arrays.

Method 1

`int arr[2][3]={11,22,33,44,55,66};`

Method 2

`int arr[2][3]={{11,22,33},{44,55,66}};`

Now that we are equipped with all the relevant information about 2-D arrays, let us move a step ahead to understand 3-D arrays. Although these are rarely used in the common problems that we solve but it is always better to have a slight idea of how the dimensions of an array are scaled up.

Three-dimensional arrays also work in a similar way. For example:

`double arr[3][2][5];`

This array arr can hold a maximum of 30 elements of double type.

This array ‘arr’ can be considered as 3 arrays of 2D which has 2 rows and 5 columns.

`arr[0][0][0]`	`arr[0][0][1]`	`arr[0][0][2]`	`arr[0][0][3]`	`arr[0][0][4]`
`arr[0][1][0]`	`arr[0][1][1]`	`arr[0][1][2]`	`arr[0][1][3]`	`arr[0][1][4]`

`arr[1][0][0]`	`arr[1][0][1]`	`arr[1][0][2]`	`arr[1][0][3]`	`arr[1][0][4]`
`arr[1][1][0]`	`arr[1][1][1]`	`arr[1][1][2]`	`arr[1][1][3]`	`arr[1][1][4]`

`arr[2][0][0]`	`arr[2][0][1]`	`arr[2][0][2]`	`arr[2][0][3]`	`arr[2][0][4]`
`arr[2][1][0]`	`arr[2][1][1]`	`arr[2][1][2]`	`arr[2][1][3]`	`arr[2][1][4]`

Here, we have 5 elements in each row and 2 such rows so total 10 elements in one 2-D array then we have 3 such 2-D arrays so the total number of elements will be 3*10 that is 30.

We can find out the total number of elements in the array simply by multiplying its dimensions:

325=30

Taking 2D Array as input

For all implementations, we will have to take inputs from user and work on that data.

Let us learn to take inputs in a 2-D array:

A 2D array is an array that contains elements in the form of rows and columns. It means we require both rows and columns to populate a two-dimensional array. Matrix is the best example of a 2D array. We have already learnt to declare 2D arrays and the way to access each element.

Let us have a glance at the code to have a clear idea.

```
#include<iostream>
using namespace std;
int main(){
  int arr[2][3];
  int i, j;
  cout<<"\n2D Array Input By user:\n";
  for(i=0;i<2;i++){
    for(j=0;j<3;j++){
      cout<<"\ns["<<i<<"]["<<j<<"]=  ";
      cin>>arr[i][j];
    }
  }
  cout<<"\nThe 2-D Array entered by user is:\n";
  for(i=0;i<2;i++){
    for(j=0;j<3;j++){
      cout<<"\t"<<arr[i][j];
    }
  cout<<endl;
  }
}
```

OUTPUT :

`2D Array Input By User :`

`s[0][0] = 2`

`s[0][1] = 4`

`s[0][2] = 3`

`s[1][0] = 6`

`s[1][1] = 7`

```
s[1][2] = 9
```

```
The 2-D Array entered by user is :
```

```
2    4    3
```

```
6    7    9
```

Explanation : In the above code firstly we are taking the input of the different elements of the array and after taking input we are printing the elements of the array.

Why do we need Multi-Dimensional Arrays?

- Multi-dimensional arrays are the best choice for representing grids/matrices.
- The most commonly used multidimensional array is the two-dimensional array, also known as a table or matrix.
- The advantage of a multidimensional array is that multi-dimensional input can be taken from the user, with faster access and a predefined size.
- Multidimensional or 2D arrays are easy to access and maintain.
- You don't have to use multiple variables for each entity which can reside in a single variable throughout your application. Every variable created takes up a specific resource that has to be looked up when accessed.

Introduction to 2D Vectors

A 2D vector is a vector of vectors. Like 2D arrays, values can be declared and assigned to a 2D vector.

A 2D vector is initialized as:

```
vector<vector<datatype>> vector_name
```

Initializing a 2-D vector with help of 1-D vector

```
#include<bits/stdc++.h>
using namespace std;

int main(){
  vector<int>v1(3 , 5);        //{5,5,5}
  vector<int>v2(5 , 6);        //{6,6,6,6,6}
  vector<int>v3(1 , 7);        //{7}

  vector<vector<int>>vec;
  vec.push_back(v1);
  vec.push_back(v2);
  vec.push_back(v3);

  for(int i=0;i<vec.size();i++){
    for(int j=0;j<vec[i].size();j++){
        cout<<vec[i][j]<<" ";
    }
    cout<<endl;
  }
}
```

Output :

`5 5 5`

`6 6 6 6 6`

7

Initializing a 2-D vector of n rows and m columns

`vector<vector<datatype>>vector_name(NumOfRows , vector<datatype>(NumOfColumns));`

For example ,

`vector<vector<int>>vec(3 , vector<int>(4));`

This statement will create a vector of 3 rows and 4 columns.
