Lesson:

2D Arrays in C++

PW SKILLS

Pre-Requisites

- Basic C++ syntax
- Basics of arrays

List of Concepts Involved

- Multidimensional Array
- 2D Array
- 2D Array Problems

Topic: 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.

	Col1	Col2	Col3	Col4	Col5
Row 1	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]	arr[0][4]
Row 2	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]	arr[1][4]
Row 3	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]	arr[2][4]
Row 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
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 array, 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 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:

3*2*5=30

Topic: Taking 2D Array as input

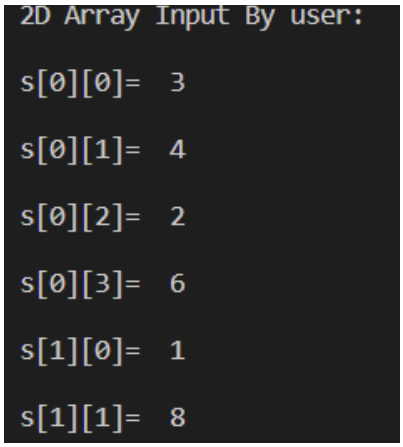
For all implementations, we will have to take inputs from user and work on that data.
Let us learn to take inputs in a 2-D array:

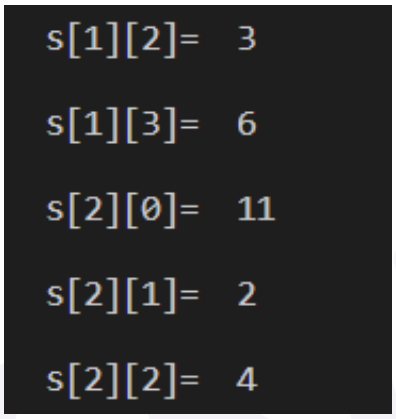
A 2D array is an array that contains elements in the form of rows and columns. It means we require both rows and columns to populate a two-dimensional array. Matrix is the best example of a 2D array. We have already learnt to declare 2D arrays and the way to access each element.

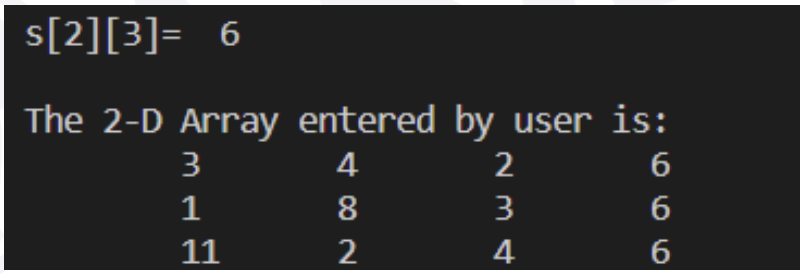
Let us have a glance at the code to have a clear idea.

```
#include<iostream>
using namespace std;
int main()
{
  int arr[3][4];
  int i, j;
  cout<<"\n2D Array Input By user:\n";
  for(i=0;i<3;i++)
  {
    for(j=0;j<4;j++)
    {
      cout<<"\ns["<<i<<"]["<<j<<"]= ";
      cin>>arr[i][j];
    }
  }
  cout<<"\nThe 2-D Array entered by user is:\n";
  for(i=0;i<3;i++)
  {
    for(j=0;j<4;j++)
    {
      cout<<"\t"<<arr[i][j];
    }
    cout<<endl;
  }
}
```

OUTPUT :







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.

Topic: 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 array is 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.

Questions:

Q1. Write a program to display multiplication of two matrices entered by the user.

Input
enter the number of row=3
enter the number of column=3
enter the first matrix element=
1 2 3 4 5 6 7 8 9
enter the second matrix element=
1 2 3 4 5 6 7 8 9
Output
multiply of the matrix=
30 36 42
66 81 96
102 126 150

Explanation:

- For multiplication of two matrices, the number of columns of the first matrix should be equal to the number of rows of the second matrix. Here, we have assumed both matrices have equal number of rows and columns.
- The program below asks for the number of rows and columns of two matrices and then elements of the matrices.
- The product of two matrices, m and n, is the sum of the products across some row of m with the corresponding entries down some column of n. In simple words, the dot product of the first row of the first matrix and the first column of the second matrix will result in the first element of the product matrix.

Look at the code to see how it can be implemented.

Solution

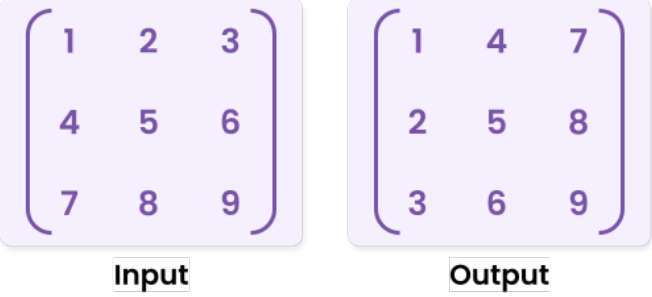
```
#include <iostream>
using namespace std;
int main()
{ int m,n;
  cout<<"enter the number of row=";
  cin>>m;
  cout<<"enter the number of column=";
  cin>>n;
  int arr1[m][n],arr2[m][n],ans[m][n],i,j,k;

  cout<<"enter the first matrix element=\n";
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      cin>>arr1[i][j];
    }
  }
  cout<<"enter the second matrix element=\n";
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      cin>>arr2[i][j];
    }
  }
  cout<<"multiply of the matrix=\n";
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      ans[i][j]=0;
      for(k=0;k<n;k++)
      {
        ans[i][j]+=arr1[i][k]*arr2[k][j];
      }
    }
  }
  //for printing result
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      cout<<ans[i][j]<<" ";
    }
    cout<<"\n”;
  }
  return 0;
}
```

OUTPUT :

```
enter the number of row=3
enter the number of column=3
enter the first matrix element=
2 3 1
3 4 6
4 7 3
enter the second matrix element=
2 7 8
6 9 0
5 4 3
multiplication of the matrices=
27 45 19
60 81 42
65 103 41
```

Q2. Write a program to Print the transpose of the matrix entered by the user.

Explanation:
Input :
row=3
col=3
arr[] = {{1,2,3}, {4,5,6}, {7,8,9}}
Output : 1 4 7 2 5 8 3 6 9



Input **Output**

Explanation:
Transpose of a matrix is obtained by changing rows to columns and columns to rows.

- Run a nested loop using two integer pointers i and j for 0 <= i < row and 0 <= j < col
- Set trans[i][j] equal to arr[j][i]

Solution:

```
#include<iostream>
using namespace std;
int main()
{
  int row,col;
  //Get size of matrix
  cout<<"Enter the no of rows in the Matrix:";
  cin>>row;
  cout<<"Enter the no of columns in the Matrix:";
  cin>>col;

  int arr[row][col];

  //Taking input of the matrix
  int i,j;
  cout<<"Enter the Elements:\n";
  for(i=0;i<row;i++)
  {
    for(j=0;j<col;j++)
    {
      cin>>arr[i][j];
    }
  }
  //compute the transpose matrix
  int trans[col][row];
    for(i=0;i<col;i++)
  {
    for(j=0;j<row;j++)
    {
      trans[i][j]=arr[j][i];
    }
  }
  //display the transpose Matrix
  cout<<"Transpose of the given Matrix is:\n";
  for(i=0;i<col;i++)
  {
    for(j=0;j<row;j++)
    {
      cout<<trans[i][j]<<" ";
    }
    cout<<"\n";
  }
}
```

OUTPUT :

```
Enter the no of rows in the Matrix:3
Enter the no of columns in the Matrix:3
Enter the Elements:
5
4
3
6
7
8
92
1
34
Transpose of the given Matrix is:
5 6 92
4 7 1
3 8 34
```

That is all for the class today, let us catch up in the next lesson to solve problems based on 2-D arrays

Upcoming Class Teasers:

- Practice problems based on 2D Arrays