1.9 Two Dimension Array

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of [x, y] you would write something as follows:

Syntex:

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, which will have x number of rows and y number of columns.
- A two-dimensional array **a**, which contains three rows and four columns can be shown as 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.
- Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions. For example:

  The array : int x[10][20] can store total (10*20) = 200 elements.

Example :

int a[3][5] ; // declaration of an integer array a with 3 rows and 5 columns

float tab[2][6] ; // declaration of an float array a with 2 rows and 6 columns

Initializing Two-Dimensional Arrays

Multi dimensioned 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 */
   };
```

0	**1**	**2**	**3**
4	**5**	**6**	**7**
8	**9**	**10**	**11**

- The nested braces, which indicate the intended row, 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 **row index and column index of the array**. For example :

The element in the position a [2][2] is 10

The element in the position a [1][3] is 7

- To output all the elements of a Two-Dimensional array we can use nested for loops. We will require two for loops. One to traverse the rows and another to traverse columns.

Example: write c++ program to print the elements of the array buf [5][2], which initialized as : buf [5][2] ={ 0, 0, 1, 2, 2, 4, 3, 6, 4, 8 }.

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

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

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

Example : Write c++ program to read an integer elements of array with size(4x4), then find the summation of these elements, finally print these elements.

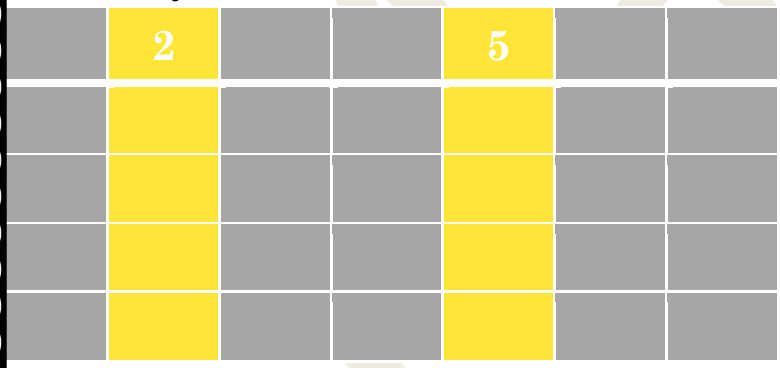
```
#include<iostream.h>
void main ( )
{
        int a [ 4 ] [ 4 ];
        int i , j, sum = 0;
        for ( i = 0 ; i < 4; i++ )
           for ( j = 0 ; j < 4; j++ )
              cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 4; i++ )
           for ( j = 0 ; j < 4; j++ )
              sum += a [ i ] [ j ];
        cout << "summation is: " << sum << endl;

        for ( i = 0 ; i < 4; i++ )
           {
              for ( j = 0 ; j < 4; j++ )
                 cout << a [ i ] [ j ];
              cout << endl;
           }
}
```

```
#include<iostream>
using namespace std;
int main()
 {  const int n = 3;
  int arr[n][n];     int i,j,larg;
//read array 2_D
 for ( i = 0; i < n; ++i)
  for ( j = 0; j < n; ++j)
cin>>arr[i][j];
//Find largest No. in array
larg=arr[0][0];
 for ( i = 0; i < n; ++i)
  for ( j = 0; j < n; ++j)
     if (arr[i][j]>larg)      larg=arr[i][j];
cout<<"Largest No.= "<<larg;
return 0;}
```

Maha

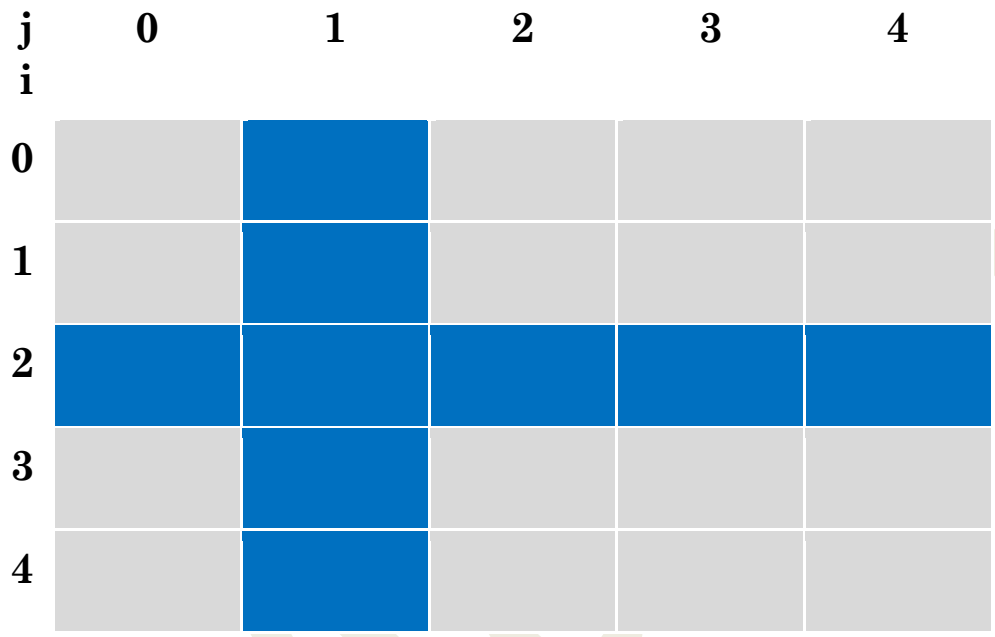
EX// W. P. to exchange 2nd column with 5th column of array s[5,7]?

```
#include <iostream>
using namespace std;
int main()
{ int s[5][7],i,j,t;
for(i=0;i<5;i++) // Read array 2_D, s[5,7]
for(j=0;j<7;j++)
cin>>s[i][j];
for(i=0;i<5;i++) // exchange operation
{ t=s[i][1];
s[i][1]=s[i][4];
s[i][4]=t; }
for(i=0;i<5;i++) // print array s[5,7] after exchange
{ for(j=0;j<7;j++)
cout<<" "<<s[i][j];
cout<<"\n";
 }
return 0;}
```



Q / write program section to switch the second column place the third row in the matrix A[5,5]?

```
for (i=0;i<5;i++)
 for (j=0;j<5;j++)
if (i==j)
 { t=a[i][1];  a[i][1]=a[2][j];  a[2][j]=t; }
```



Q/W.P. to convert array 2-D a[3][3] to 1-D?

```
#include<iostream>
using namespace std;
int main()
{
const int n=3;
int a[n][n]={{1,2,3},{4,5,6},{7,8,9}};
int b[9];
int i,j,k=0;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
 {      b[k]=a[i][j]; k++;}
 for(i=0;i<9;i++)
cout<<"  "<<b[i];
return 0;}
```

Q1 / write program section to switch the second row place fourth row in the matrix A[5,5]?

Q2/W.P. to convert array 1-D a[9] into array 2-D?

Q3/W.P. to find largest No. to each row in array a[3][3]?

Q4/W.P. to calculate summation to each row in array a[3][3]?

Maha

Lecture 10

Functions

Function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

Function declaration tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Defining a Function

The general form of a C++ function definition is as follows:

```
return_type  function_name  ( parameters)
{
   body of the function
}
```

- C++ function definition consists of **a function header** and a **function body.**

Here are all the parts of a function:

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the **return_type** is the eyword **void.**
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the

type, order, and number of the parameters of a function. **Parameters are optional; that is, a function may contain no parameters.**

- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

Following is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum between the two:

(1) Defining a Function

```
// This function returning the max between two numbers

int max(int num1, int num2)

{
int result;   // local variable declaration

if (num1 > num2)
result = num1;
else
result = num2;
return result;

}
```

- For the above defined function **max(),** following is the **function declaration**:

int max (int num1, int num2) ;

- Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

int max (int, int) ;

- you should declare the function at the top of the file calling the function.

(3) Calling a Function

- To use a function, you will have to call or invoke that function.
- When a program calls a function, program control is transferred to the called function.
- A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.
- **For example:**

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

int max(int num1, int num2);   // function declaration
```

```
int main ()
```

Maha

Maha

Example : Write C++ program to calculate the **squared value** of a given integer number passed from the main function. Use this function in the program to calculate the squares of numbers from 1 to 10.

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

int square(int  y);   // function declaration

 void main( )
 {
        int x;
        for ( x=1; x <= 10; x++ )
```

Maha

Example: Write a function to find the largest integer number among three integer numbers entered by the user in the main function.

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

int max(int , int,  int) ;  // function declaration

void main()
{
```

Maha

Example: Write C++ program to create a simple calculator to (add, subtract, multiply and divide) using function.

```
# include <iostream>
using namespace std;
 float calc( float x, float y, char op) ; // function declaration
int main()
{   char op;
    float num1, num2, result;
    cout << "Enter operator either + or - or * or /: ";
    cin >> op;
    cout << "Enter two numbers: ";
```

Scope of variables

Scope is a region of a program. **Variable Scope** is a region in a program where a variable is declared and used. Variables are thus of two types depending on the region where these are declared and used.

1. Local Variables :

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables .

```
#include <iostream>
using namespace std;
 int main () {
   int a, b; // Local variable declaration:
   int c;
   a = 10;
   b = 20;
   c = a + b;
    cout << c;
    return 0;
}
```

2. Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables.

```
#include <iostream>
```

using namespace std;

```
 void func1(void); // declaration function
```

```
int g =10;        // Global variable declaration:
int main}()
     func1();   // call function
     g = 30;
     cout << g << endl;
     return 0;
}

void func1()   // function definition
 {
     g = 20;
     cout << g << endl;
}
```

Ex// a simple function which raises an integer to the power of another, positive integer?

```
#include <iostream>
using namespace std;
      int Power (int base, int exponent)
      {
      int result = 1;
      for (int i = 0; i < exponent; ++i)
      result *= base;
```

```
        return result;
        }
int main ()
{
cout << "2 ^ 8 = " << Power(2,8) << '\n';
return 0;}
```

Ex// W.P to read positive integer and compute factorial X! value?

```
#include <iostream>
using namespace std;
 int fict (int x1)//Interface function
{
int result = 1;                    //body function
for (int i = 1; i <= x1; i++)
result *= i;
return result;
}
int main ()
{
int x;
cout<<"enter X value : "; cin>>x;
cout << "X! = " << fict(x) << '\n';   //call function
return 0;}
```

other method to solve last example:

```
#include <iostream>
using namespace std;
int fict (int); //declare function
int main ()
{
int x;
cout<<"enter X value : "; cin>>x;
cout << "X! = " << fict(x) << '\n';   //call function
```

```
Return 0;}
 int fict (int x1)//Interface function
{
int result = 1;                  //body function
for (int i = 1; i <= x1; i++)
result *= i;
return result;
}
```

x//find y value by using function:

$$y = \frac{K! * R!}{(K - R)!}$$

```
#include <iostream>
using namespace std;
 int fict (int x1)//Interface function
{
int result = 1;                  //body function
for (int i = 1; i <= x1; i++)
result *= i;
return result;
}


 int main ()
 {
 int K,R,N; double Y;
 cout<<"enter K value : "; cin>>K;
 cout<<"enter R value : "; cin>>R;
 N=K-R;
 Y=fict(K)*fict(R)/fict(N); //call function three times
```

Maha

```
cout << "Y = " << Y;
return 0;
}
```

Ex//print the following figure by using function?

```
#include <iostream>
using namespace std;
disp( )
{
int i,j;
for(i=0;i<5;i++)
{ for (j=0;j<=i;j++)
if (i%2==1)
        cout<<"*";        else  cout<<"+";
cout<<'\n'; }
}
int main()
{
disp();
return 0;
}
```

Output:

```
*
++
***
++++
*****
```

Ex// write a program in c++ to read and print array a[5] by using function?

```
#include <iostream>
```

```
using namespace std;
sa(int g[5])
        {
        int i;
        for(i=0;i<5;i++)
                cin>>g[i];
        for(i=0;i<5;i++)
                cout<<"\t"<<g[i];
        }
int main()
{ int a[5];
sa(a);
return 0;
}
```

Ex// write a program in c++ to read array a[10] and find largest element by using function?

```
#include <iostream>
using namespace std;
int larg(int b[10])
{
int i,r;
r=b[0];
for(i=1;i<10;i++)
if (b[i]>r)
```

Maha

```
        r=b[i];
return r;
}
int main()
{
int a[10],i;
for(i=0;i<10;i++)
cin>>a[i];
cout<<"larg="<<larg(a);
return 0;
}
```

Ex// write a program in c++ to read array a[10] and print element by using function? Other method

```
#include <iostream>
using namespace std;
int a[5]; //global variables
sarr(int i1)
{
cin>>a[i1];
}
int main()
{
int i;
for(i=0;i<5;i++)
sarr(i);
for(i=0;i<5;i++)
cout<<"\t"<<a[i];
return 0;
}
```

EX//W. P. to read 10 record to students contains (id,avr) fields and find largest and smallest average?

```
#include <iostream>
using namespace std;
int main()
```

```
{    struct {
 int id;  float avr;
         } stu[10];
for(int i=0;i<10;i++)
{ cin>>stu[i].id;  cin>>stu[i].avr; }
larg=stu[0].avr;
small=stu[0].avr;
for (i=0;i<10;i++)
{ if (stu[i].avr>larg)
    larg=stu[i].avr;
 if (stu[i].avr<small)
    small=stu[i].avr;
}
cout<<larg;
cout<<small;
}
```

Maha

Scope of variables

Scope is a region of a program. **Variable Scope** is a region in a program where a variable is declared and used. Variables are thus of two types depending on the region where these are declared and used.

3. Local Variables :

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables .

```
#include <iostream>
using namespace std;
 int main () {
   int a, b; // Local variable declaration:
   int c;
   a = 10;
   b = 20;
   c = a + b;
    cout << c;
    return 0;}
```

4- Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables.

```
#include <iostream>
```

using namespace std;

```
 void func1(void); // declaration function
 int g =10;        // Global variable declaration:
 int main}()
      func1();   // call function
      g = 30؛
      cout << g << endl؛
      return 0؛
}
void func1()   // function definition
 {
      g = 20؛
      cout << g << endl؛}
```