LAB NO. 1

C++ Review

OBJECTIVE:

- To Review the basic concepts of C++.
- To Review Arrays, how to declare, initialize and access 2D and 3D arrays Implement Arrays in C++.

DESCRIPTION:

Fundamental Data Types

Category	*Available data types*		
Boolean	bool		
Character	char	signed char	unsigned char
Signed integer	short	int	long
Unsigned integer	unsigned short	unsigned	unsigned long
Floating point	float	double	long double

Named Constants

Cannot be changed during program execution

C style constants:

```
#define zero 0
```

C++ style constants:

```
const int zero = 0;
const float PI = 3.14159;
```

Arithmetic Expressions

Binary operators: +, –, *, /, %
Unary operators: +, –, ++, – –

Usual precedence rules apply

Unary operators are *right*-associative: ++ X – – means ++ (X – –)

Binary operators are *left*-associative: A / B * C means (A / B) * C

Relational & Logical Operator

Relational operators: <, >, <=, >=
Equality operators: = =, !=
Logical operators:
Unary: !
Binary: &&, | |

Examples:

```
(5 = = 4) && (a < b)      // false, since 5 != 4
(5 = = 5)  | |  (a < b)      // true, since 5 = = 5
```

Conditional Statements

expression **?** *expression* **:** *expression*

Executed like *if – else* statement, but has a value

Example:

```
larger = (A > B) ? A : B;
```

Switch Statement: -

```
switch (expression)
{
case const-1:
        Statements;
        break;
case const-2:
        Statements;
        break;
case const-n:
      Statements;
      break;
default:
      statements;
}
```

Loops: -

For loop

for (initialization; condition; increment/decrement)

While Loop

while (condition)

{ Statement;

}

Do while Loop

do

{ statements; }

while(condition);

Functions

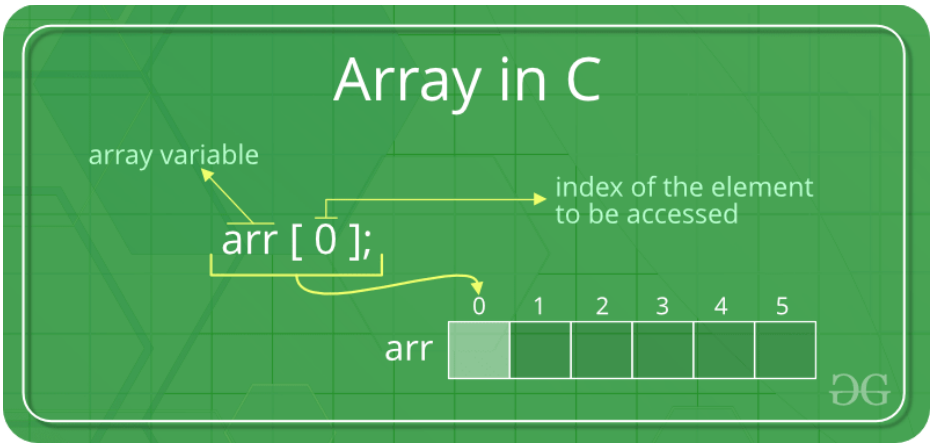
```
int  max1( int X, int Y )
{
        return (X > Y) ? X : Y;              // result returned as function value
}

void  max2( int X, int Y, int &Larger )
{
   Larger = (X > Y) ? X : Y;                 // result returned by reference
}

void  max3( int X, int Y, int *Larger )
{
     *Larger = (X > Y) ? X : Y;              // result returned by pointer
}
```

Arrays

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers [0], numbers [1], and ..., numbers [99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in **C++,** the programmer specifies the type of the elements and the number of elements required by an array as follows:

$$\text{type arrayName [arraySize];}$$

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type. For example, to declare a 10element array called balance of type double, use this statement:

$$\text{double balance[10];}$$

Initializing Arrays

You can initialize C++ array elements either one by one or using a single statement as follows:

$$\text{double balance[5]} = \{1000.0, 2.0, 3.4, 17.0, 50.0\};$$

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array:

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

$$\text{double balance}[] = \{1000.0, 2.0, 3.4, 17.0, 50.0\};$$

You will create exactly the same array as you did in the previous example.

$$\text{balance}[4] = 50.0;$$

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index.

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

$$\text{double salary} = \text{balance}[9];$$

The above statement will take 10th element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays:

Lab Activity

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

#include <iomanip>
using std::setw;

int main () {
  int n[ 10 ]; // n is an array of 10 integers

  // initialize elements of array n to 0
for ( int i = 0; i < 10; i++ )
{
    n[ i ] = i + 100; // set element at location i to i + 100
}
```

```
   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output each array element's value

  for ( int j = 0; j < 10; j++ ) {
     cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;
   }
   return 0;
 }
```

This program makes use of setw() function to format the output. Compile and run your program to see the output.

2D Arrays

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

```
int threedim[5][10][4];
```

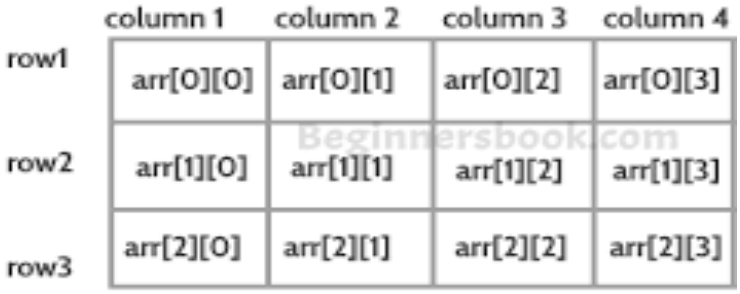
Two-Dimensional Arrays

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 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, which will have x number of rows and y number of columns. A 2-dimensional array a, which contains three rows and four columns can be shown as below:



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

Multidimensional 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 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 the subscripts, i.e., row index and column index of the array. For example:

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

The above statement will take 4th element from the 3rd row of the array.

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

int main () {
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};    // an array with 5 rows and 2 columns.
  for ( int i = 0; i < 5; i++ )    // output each array element's value
        for ( int j = 0; j < 2; j++ ) {
```

```
cout << "a[" << i << "][" << j << "]: ";
cout << a[i][j]<< endl; }
return 0; }
```

When the above code is compiled and executed, it produces the following result:

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

LAB TASK:

1. You're given with marks of 10 students in Mathematics, write a program to determine the grade of each student.

80, 72, 93, 87, 90, 55, 66, 74, 69, 56

Assume:

Grade is A if score is equal and greater than 90

Grade is B+ if score is less than 90 and greater than 81

Grade is B if score is less than 82 and greater than 71

Grade is C if score is less than 72 and greater than 66

Grade is D if score is less than 66 and greater than 59

Grade is F if score is less than 60.

2. Write a program to ask user to enter 5 floating numbers and find the maximum and minimum of all by calling min() and max() functions.

3. Write a program that shows following output

```
Please enter 10 Numbers:
1 2 3 4 5 6 7 8 9 10
Element Value   Histograam
 0      1       *
 1      2       **
 2      3       ***
 3      4       ****
 4      5       *****
 5      6       ******
 6      7       *******
 7      8       ********
 8      9       *********
 9      10      **********
Press any key to continue . . .
```

4. Write a program that will print multi-subscripted array as shown below using function printArray().

```
Values in Array1 by row are
1 2 3
4 5 6
Values in Array2 by row are
1 2 3
4 5 0
Values in Array3 by row are
1 2 0
4 0 0
Press any key to continue . . .
```