## Programming Fundamentals

## LAB MANUAL13-14

**Implementation of different programs using arrays.**

**C++ Array**

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier. That means that, for example, five values of type **int**can be declared as an array without having to declare 5 different variables (each with its own identifier). Instead, using an array, the five **int** values are stored in contiguous memory locations, and all five can be accessed using the same identifier, with the proper index.

**Example:**   
For example, an array containing 5 integer values of type **int**called **foo** could be represented as:  
  
http://www.cplusplus.com/doc/tutorial/arrays/arrays1.png 

**Declaring arrays**  
like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

**type name [elements];** .

Therefore, the foo array, with five elements of type int, can be declared as:

|  |  |  |
| --- | --- | --- |
|  | **int foo [5];** |  |

The elements field within square brackets [], representing the number of elements in the array, must be constant expression, since arrays are blocks of static memory whose size must be determined at compile time, before the program runs.

**Initializing arrays**

The elements in an array can be explicitly initialized to specific values when it is declared, by enclosing those initial values in braces [].

For example:

|  |  |  |
| --- | --- | --- |
|  | **int foo [5] = { 16, 2, 77, 40, 12071 };** |  |

   
The number of values between braces [] shall not be greater than the number of elements in the array. For example, in the example above, foo was declared having 5 elements (as specified by the number enclosed in square brackets, []), and the braces [] contained exactly 5 values, one for each element.

If declared with less, the remaining elements are set to their default values (which for fundamental types, means they are filled with zeroes). For example:

|  |  |  |
| --- | --- | --- |
|  | **int bar [5] = { 10, 20, 30 };** |  |

Will create an array like this:  
  
http://www.cplusplus.com/doc/tutorial/arrays/arrays3.png   
The initializer can even have no values, just the braces:

|  |  |  |
| --- | --- | --- |
|  | **int baz [5] = { };** |  |

This creates an array of five i**nt** values, each initialized with a value of zero:  
  
http://www.cplusplus.com/doc/tutorial/arrays/arrays4.png 

When an initialization of values is provided for an array, C++ allows the possibility of leaving the square brackets empty []. In this case, the compiler will assume automatically a size for the array that matches the number of values included between the braces {}:

|  |  |  |
| --- | --- | --- |
|  | **int foo [] = { 16, 2, 77, 40, 12071 };** |  |

After this declaration, array foo would be 5 int long, since we have provided 5 initialization values.

**Accessing the values of an array**

The values of any of the elements in an array can be accessed just like the value of a regular variable of the same type. The syntax is: For example, the following statement stores the value 75 in the third element of foo:

|  |  |  |
| --- | --- | --- |
|  | **foo [2] = 75;** |  |

Following copies the value of the third element of foo to a variable called x:

|  |  |  |
| --- | --- | --- |
|  | **x = foo[2];** |  |

In C++, it is syntactically correct to exceed the valid range of indices for an array. This can create problems, since accessing out-of-range elements do not cause errors on compilation, but can cause errors on runtime. The reason for this being allowed will be seen in a later when pointers are introduced.

**Some other valid operations with arrays**:

**Example 1:**

**// sum of all the elements of arry using loop.**

#include <iostream>

using namespace std;

int foo [] = {1, 2, 3, 4, 5};

int n, result=0;

int main ()

{

for ( n=0 ; n<5 ; ++n )

{

result += foo[n];

}

cout << result<<endl;

return 0;

}

**Example 2:**

#include<iostream>

using namespace std;

int main() {

int a=2;

int b=1;

int foo[5];

foo[0] = a;

cout<<foo[0]<<endl;

foo[a] = 75;

cout<<foo[a]<<endl;

b = foo [a+2];

cout<<b<<endl;

foo[foo[a]] = foo[2] + 5;

cout<<foo[foo[a]]<<endl;

}

**Arrays as parameters**

At some point, we may need to pass an array to a function as a parameter. In C++, it is not possible to pass the entire block of memory represented by an array to a function directly as an argument. But what can be passed instead is its address. In practice, this has almost the same effect, and it is a much faster and more efficient operation.

To accept an array as parameter for a function, the parameters can be declared as the array type, but with empty brackets, omitting the actual size of the array. For example:

|  |  |  |
| --- | --- | --- |
|  | **voidfunction (int arg[])** |  |

This function accepts a parameter of type "array of **int**" called arg. In order to pass to this function an array declared as:

|  |  |  |
| --- | --- | --- |
|  | **int myarray [40];** |  |

it would be enough to write a call like this:

|  |  |  |
| --- | --- | --- |
|  | **procedure (myarray);** |  |
|  |  |  |

**Example Program:**

// arrays as parameters

#include <iostream>

using namespace std;

void printarray (int arg[], int length) {

for (int n=0; n<length; ++n)

cout << arg[n] << ' ';

cout << '\n';

}

int main ()

{

int firstarray[] = {5, 10, 15};

int secondarray[] = {2, 4, 6, 8, 10};

printarray (firstarray,3);

printarray (secondarray,5);

}

**Operations on Arrays**

|  |
| --- |
| Each member of an array is a pseudo-variable and can be processed as such. This means that you can add the values of two members of the array (Number[2]+Number[0]), you can subtract the value of one of the members from another member(member[1]-Number[4]). In the same way, you can perform multiplication, division, or remainder operations on members of an array.  One of the regular operations performed on an array consists of adding the values of the members to produce a sum.  **Example Program:** |

#include <iostream>

using namespace std;

int main()

{

// We know that we need a constant number of elements

const int max = 10;

int number[max];

// We will calculate their sum

int sum = 0;

cout << "Please type 10 integers.\n";

for( int i = 0; i < max; i++ )

{

cout << "Number " << i + 1 << ": ";

cin >> number[i];

sum += number[i];

}

cout << "\n\nThe sum of these numbers is " << sum << "\n\n";

return 0;

}

|  |
| --- |
|  |

**Character Type Array**

The elements of an array in C++ can be of any type. Arrays of floats, doubles, and longs are all possible; however, arrays of characters have particular significance.Human words and sentences can be expressed as an array of characters. An array of characters containing thename Stephen would appear as

char sMyName[] = {'S', 't', 'e', 'p', 'h', 'e', 'n'};

Example Program:

#include <iostream>

using namespace std;

// prototype declarations

void displayCharArray(char charArray[], int sizeOfArray);

int main(int nNumberofArgs, char\* pszArgs[])

{

char charMyName[]={'S', 't', 'e', 'p', 'h', 'e', 'n'};

displayCharArray(charMyName, 7);

cout << endl;

// wait until user is ready before terminating program

// to allow the user to see the program results

//cin.ignore()ignores the amount of characters you specify when you call it, up to the char you specify as a breakpoint.

cout << "Enter some pharse and then press enter..." << endl;

cin.ignore(10, '\n');//add this function before getting characters from user

cin.get();//The unformatted get member function works like the >> operator but get function includes white-space characters, whereas the extractor excludes

return 0;

}

// displayCharArray - display an array of characters

// by outputing one character at

// a time

void displayCharArray(char charArray[], int sizeOfArray)

{

for(int i = 0; i< sizeOfArray; i++)

{

cout << charArray[i];

}

}

**Multidimensional Arrays**

Multidimensional arrays can be described as "arrays of arrays". For example, a **bi-dimensional** array can be imagined as a **two-dimensional table** made of elements, all of them of a same uniform data type.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];**

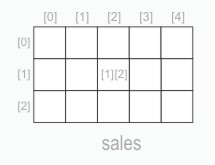
**Two Dimensional Arrays**

It is a collection of data elements of same data type arranged in rows and columns (that is, in two dimensions).

#### **Declaration**

**arrayName [numberOfRows] [numberOfColumn];**

**For example,  
int Sales[3][5];**



#### **Initialization**

A two-dimensional array can be initialized along with declaration. For two-dimensional array initialization, elements of each row are enclosed within curly braces and separated  
by commas. All rows are enclosed within curly braces.

**int A[4][3] = {{22, 23, 10},  
              {15, 25, 13},  
              {20, 74, 67},  
              {11, 18, 14}};**

#### **Referring to Array Elements**

To access the elements of a two-dimensional array, we need a pair of indices: one for  
the row position and one for the column position. The format is as simple as:

**name[rowIndex][columnIndex]**

**Examples:**

cout<<A[1][2];

A[1][2]=13;  
cin>>A[1][2];

#### **Using Loop to input an Two-Dimensional Array from user**

int mat[3][5], row, col ;  
for (row = 0; row < 3; row++)  
 for (col = 0; col < 5; col++)  
  cin >> mat[row][col];

### **Example**

**//C++ Program to display all elements of an initialized two dimensional array.**

#include<iostream>

usingnamespace std;

int main(){

int test[3][2]={

{2,-5},

{4,0},

{9,1}

};

for(int i =0; i <3;++i){

for(int j =0; j <2;++j){

cout<<"test["<< i <<"]["<< j <<"] = "<< test[i][j]<<endl;

}

}

return0;

}

#### **Arrays as Function’s Parameters**

Two-dimensional arrays can be passed as parameters to a function, and they are passed by reference. When declaring a two-dimensional array as a formal parameter, we can omit the size of the first dimension, but not the second; that is, we must specify the number of columns. For

**Example:**   
  void print(int A[][3],int N, int M)  
In order to pass to this function an array declared as:  
   int arr[4][3];

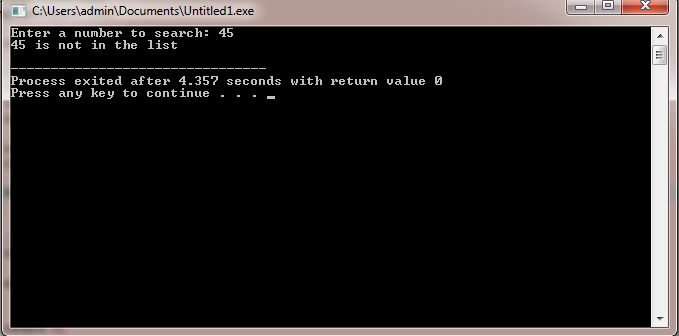
**Example:**

#include <iostream>  
using namespace std;   
void print(int A[][3],int N, int M)  
{  
  for (R = 0; R < N; R++)  
    for (C = 0; C < M; C++)  
       cout << A[R][C];  
}

int main ()  
{  
  int arr[4][3] ={{12, 29, 11},  
                  {25, 25, 13},  
                  {24, 64, 67},  
                  {11, 18, 14}};  
  print(arr,4,3);  
  return 0;  
}

**Task 1:**

Write a C++ Program that includes an array of particular size that contains some elements of **int** type. Program should ask the user to find an element from that array .whether this element is present in array or not.



**Task 02:**

C++ Program that takes 2 , 2d arrays (Matrices) of size 3X3 from user and then add these two Matrices Using Multi-dimensional Arrays and display the results in 2d Form.

