LAB #08: Arrays

Name:_____ Reg #:_____

Lab Objective:

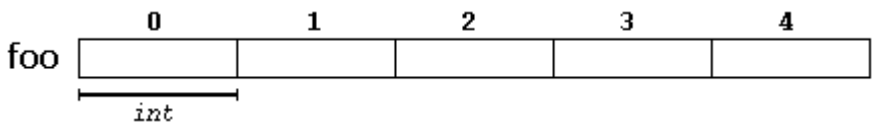
- To be able to declare an array.
- To be able to perform fundamental operations on a two-dimensional array.
- To be able to pass two-dimensional arrays as parameters.
- To be able to view a two-dimensional array as an array of arrays.

Lab Description:

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.

For example, an array containing 5 integer values of type int called foo could be represented as:



where each blank panel represents an element of the array. In this case, these are values of type int. These elements are numbered from 0 to 4, being 0 the first

and 4 the last; In C++, the first element in an array is always numbered with a zero (not a one), no matter its length.

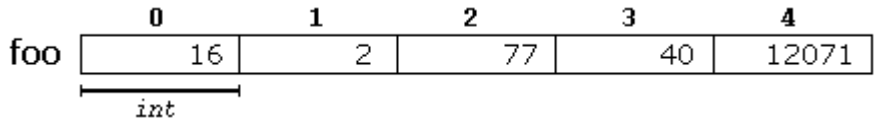
Initializing arrays:

By default, regular arrays of local scope (for example, those declared within a function) are left uninitialized. This means that none of its elements are set to any particular value; their contents are undetermined at the point the array is declared.

But 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 };
```

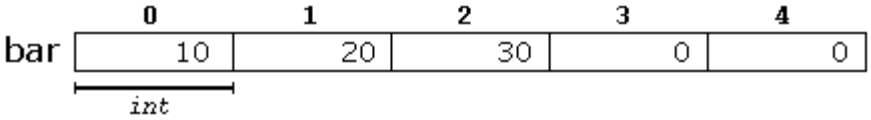
This statement declares an array that can be represented like this:



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:

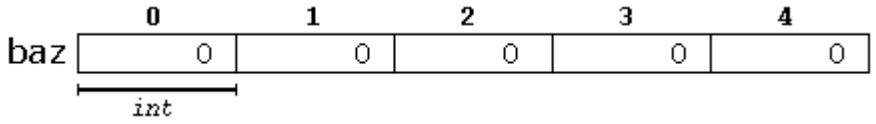


The initializer can even have no values, just the braces:

```
int baz [5] = { };
```

This creates an array of five int values, each initialized

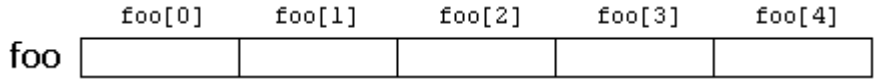
with a value of zero:



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:

name[index]
Following the previous examples in which foo had 5 elements and each of those elements was of type int, the name which can be used to refer to each element is the following:



For example, the following statement stores the value 75 in the third element of foo:

```
foo [2] = 75;
```

and, for example, the following copies the value of the third element of foo to a variable called x:

```
x = foo[2];
```

Therefore, the expression foo[2] is itself a variable of type int.

Examples: the following examples displays the sum of elements of array.

Example	OUTPUT

`#include <iostream>`
`using namespace std;`
 	
`int foo [] = {16, 2, 77, 40, 12071};`
`int n, result=0;`
 	
`int main ()`
`{`
`  for ( n=0 ; n<5 ; ++n )`
`  {`
`    result += foo[n];`
`  }`
`  cout << result;`
`  return 0;`
`}`
	`12206`

The following example shows how to Find the Highest and Lowest Values in a Numeric Array

HIGHEST	LOWEST

`int count;`
`int highest;`
 	
`highest = numbers[0];`
`for (count = 1; count < SIZE; count++)`
`{`
`  if (numbers[count] > highest)`
`  highest = numbers[count];`
`}`
	`int count;`
	`int lowest;`
	
	`lowest = numbers[0];`
	`for (count = 1; count < SIZE; count++)`
	`{`
	`  if (numbers[count] < lowest)`
	`  lowest = numbers[count];`
	`}`

Implicit Array Sizing:

It's possible to define an array without specifying its size, as long as you provide an initialization list. C++ automatically makes the array large enough to hold all the initialization values. For example, the following definition creates an array with five elements:

```
double ratings[] = {1.0, 1.5, 2.0, 2.5, 3.0};
```

Printing the Contents of an Array:

Suppose we have the following array definition:

```
onst int SIZE = 5; int numbers [SIZE] = {10, 20, 30, 40, 50};
```

You now know that an array's name is seen as the array's beginning memory address. This explains why the following statement cannot be used to display the contents of array :

```
cout << numbers << endl; //Wrong!
```

When this statement executes, cout will display the array's memory address, not the array's contents. You must use a loop to display the contents of each of the array's elements, as follows.

```
for (int count = 0; count < SIZE; count++)

cout << numbers[count] << endl;
```

Two-Dimensional 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 threeD[5][10][4];
```

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 –

	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.

Initializing Two-Dimensional Arrays

Multidimensioned 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];
```

Example:

Example

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

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

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

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

   return 0;
}
```

OUTPUT

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Lab Tasks:

- Task1: Write a program that asks for the number of hours worked by six employees. It stores the values in an array.

- Task2: Write a program that asks the user to type 10 integers of an array. The program must compute and write how many integers are greater than or equal to 10.

- Task3: Write a C++ program to swap first and last element of an integer 1-d array.

- Task4: Write a C++ function that multiplies two matrices using arrays.

For $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ (2×2) and $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$ (2×2):

$$AB = \begin{bmatrix} (1 \cdot 5 + 2 \cdot 7) & (1 \cdot 6 + 2 \cdot 8) \\ (3 \cdot 5 + 4 \cdot 7) & (3 \cdot 6 + 4 \cdot 8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

- Task5: Write a C++ function to add two matrices using multidimensional arrays.

For $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$:

$$A + B = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

- Task 6: Write a program in C++ to find the transpose of a matrix.

Matrix:

$$B = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} \quad (2 \times 3)$$

Transpose:

$$B^T = \begin{bmatrix} 5 & 8 \\ 6 & 9 \\ 7 & 10 \end{bmatrix} \quad (3 \times 2)$$

- Task 7: Write a function in C++ to find out the highest number in a 2 Dimensional array.

Think??

- Correct the errors in the following program

```
int main(){
   int SIZE=5;
   int arr[SIZE];

   // to store value 1 4 9 16 25 in arr
   for (int i=1;i<=5;i++)
        arr[i]=i*i;

   cout<<"arr5="<<arr[5];

   return 0;
}
```