Lab 4: Arrays and Vectors

Arrays

To this point in the course, variables have only been able to store a single value at a time.

```
int count = 1;
while ( count <= 5)
{
   cout << count << endl;
   count++;  // increment count by 1
}
```

However, arrays are able to store multiple values of the same data type. Begin by declaring an empty array of integers with "10 elements":

```
int array_name [10];
```

The "10 elements" are represent by the [10] in the declaration.

** Note - the array will be filled with random/meaningless values until properly initialized, similiar to a regular variable. **

Like a regular variable, arrays can be declared and initialized at the same time. The syntax used to declare and initialize an array is a little different from the syntax used to declare and initialize a scalar (i.e. regular) variable like the variables you have been using so far, due to the assignment of values to multiple array elements.

```
int array_name[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
```

This is how the array looks in memory:

10	20	30	40	50	60	70	80	90	100	array elements (values)
0	1	2	3	4	5	6	7	8	9	element index

** Note - The index starts at 0 and not at 1. **

The number of array elements, also referred to as the array's size, does not need to be declared, if initializing the array with elements. The array size must be a constant intead of a variable, so that older compilers will be able to compile the source code.

```
string MONTHS[] = { "January", "February", "March", "April", "May",
                  "June", "July", "August", "September", "October",
                  "November", "December"};
```

**Note -The compiler will assume the size of the array by the number of values.

Once declared, an element can be individually accessed using the index. Here are a few examples:

```
//assigns a value of 5 to index 0
array_name[0] = 5;

//assigns the value of index 0 to index 1
array_name[1] = array_name[0];

//adds two different array elements and assigns the sum to array index [0]
array_name[0] = array_name[1] + array_name[2];
```

Iterating Over an Array

Loops are useful and necessary to traverse arrays.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   // Note the const int SIZE throughout the program
   // to represent the size of the array.  To adjust the size of
   // the array change SIZE's value.

   const int SIZE = 5;

   char array[SIZE];

   // Here we allow the user to fill an array using a FOR loop
   cout << endl << "Enter five characters" << endl;
   for(int i = 0; i < SIZE; i++)
   {
      cin >> array[i];
   }

   // Now we print the array back to the screen using another FOR loop

   for(int i = 0; i < SIZE; i++)
   {
      cout << array[i] << " ";
   }
   cout << endl;
   return 0;
}
```

Going Out of Bounds of an Array

It is easy to crash a program with a segmentation fault by attempting to access data outside of your array, or even worse, it may allow you to access a memory location not allocated to the array.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   // What error is received when running the following code?
   double array_2 [10];
   cout << "Testing Segmentation fault. Creating an array with 10 elements.\n";

   // notice that this will likely run even though it is outside of your array
   array_2[10] = 100;
   cout << "array element 11 " << array_2[10] << endl;

   // This will almost certainly cause an access violation
   array_2[1000000] = 100;
   cout << "array element 1000001 " << array_2[1000000] << endl;
   return 0;
}
```

Understanding the Index of an Array

The index of an array is simply an integer. Therefore the index can be saved in a variable, to reference or modify that element later.

```
#include <iostream>
#include <fstream>

using namespace std;
int main()
{
   // an array of strings containing the months:
   const string MONTHS[] = { "January", "February", "March", "April", "May",
                             "June", "July", "August", "September", "October",
                             "November", "December"};
   // If the user indicated that their birthday was in June, instead of saving
   // the string "June" we can instead save the index to June which is 5.
   int birthday_index = 5;
   // now when we want to print out their birthday month, we can use
   // the following:
   cout << "My birthday is in " << MONTHS[birthday_index] << endl;
   // Notice how the index is the variable "birthday_index".
   return 0;
}
```

Array Example 1

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   // Here is a simple algorithm that searches an array for the number 10.
   const int SIZE = 5;

   int array_4[SIZE] = {4, 9, 10, 32, 44};  // array to be searched

   int searchValue = 10; // the value to be found within the array

   int foundIndex = -1;  // -1 signals "not found".

   // search array for value
   for(int i = 0; i < SIZE; i++)
   {
      // if the value being searched is located in the array, then assign
      // i to "foundIndex" to trigger a later if statement to output the
      // results

      if(searchValue == array_4[i])
      {
         foundIndex = i;
      }
   }

   // print result of search
   if (foundIndex != -1)
   {
      cout << "Found value at index " << foundIndex << endl;
   }
   else
   {
      cout << "Value not found" << endl;
   }

   return 0;
}
```

Array Example 2 - File I/O Fixed Size

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   // here the code will fill an array using input from a file and send
   // it to an output file.
   // *IMPORTANT*
   // We use ARRAY_SIZE constant to specify the size of the array.
   const int ARRAY_SIZE = 5;

   int array_5 [ARRAY_SIZE];

   ifstream fin;
   fin.open("input.txt");

   if(!fin)
   {
      cout << "Error - file not found" << endl;
      return 1;
   }

   // Pull ARRAY_SIZE number of data entries into your
   // array. Note that if there are fewer entries in your
   // input file this code will not work properly.

   for (int i = 0; i < ARRAY_SIZE; i++)
   {
      fin >> array_5[i];
   }

   fin.close();
   cout << "Your data has been written to output file - output.txt\n";

   // Send our data to an output file

   ofstream fout;
   fout.open("output.txt");

   for(int j = 0; j < ARRAY_SIZE; j++)
   {
      fout << array_5[j] << " ";
   }

   fout.close();

   return 0;
}
```

Array Example 2 - File I/O Uncertain Size

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   // This code pulls data from a file into an array, but this time it is
   // uncertain of the number of elements inside the file. Therefore, we use
   // a variable 'count' to keep track of how many items found within the file.

   const int ARRAY_SIZE_2 = 100;
   int array_3 [ARRAY_SIZE_2];
   int count;

   ifstream fin;
   fin.open("input.txt");

   if(!fin)
   {
      cout << "Error - file not found" << endl;
      return 1;
   }
   count = 0;
   // while the array is not full and the file is not empty,
   // pull data from file and put it into the next array element.
   while(count < ARRAY_SIZE_2 && fin >> array_3[count])
   {
      count++;
   }

   fin.close();
   // print array to screen
   for(int k = 0; k < count; k++)
   {
      cout << array_3[k] << " ";
   }
   cout << endl;
   // Now lets send our data to an output file
   ofstream fout;
   fout.open("output.txt");
   for(int j = 0; j < count; j++)
   {
      fout << array_3[j] << " ";
   }

   fout.close();

   return 0;
}
```