Western Engineering

ES1036: Programming Fundamentals – **Winter 2018**

Lab06: Arrays in C++

Faculty of Engineering, Western University

In order to finish this lab on time, you must start ahead of the lab session. You must try to complete the lab early and contact your TA if you require any help. The lab is due for submission on OWL at the end of your lab section.

A. Background

An **array** is a set of **consecutive** memory locations used to store **same types of data**. This structure is useful because it allows a block of memory to be accessed without uniquely assigning a variable name to each data element. We can iterate through an **array** structure or access each element individually.

B. How to Get Full Credit for this Lab

You will get credit for your lab when you demonstrate it to your TA and get approval. During the code-demonstration, your TA will ask questions about your code or request you to perform some modifications. When you finish working on the lab, you will be required to upload your .cpp files (**no other file types will be accepted**) to OWL after changing the file name as indicated below. A zero grade will be awarded for a missing OWL submission or missing the lab demonstration.

- **Submission Instructions**

Number of files	:	**3**
Files to be submitted	:	.cpp source files for **E1, E2** and **E3**
Naming requirements	:	your_Western_user_name_lab06_q1.cpp,
		your_Western_user_name_lab06_q2.cpp,
		your_Western_user_name_lab06_q3.cpp.

C. Review

It is important that you review and understand the sections referenced below. Studying these sections will help you complete the lab.

Functions: review lecture Unit 5 and the previous lab05 assignment.

Static variable identifier: see lecture Unit 5, slides 77 through 80. (Examples on slides 79 and 80)

Arrays: see lecture handout Unit 6, slides 2 through 25. (Examples on slides 15 to 18, and 22 to 25)

C.1 –Accessing Array Elements

The following program demonstrates how an array is accessed, and how it can be passed as a parameter to a function.

```cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

void printArray(int someArray[], int size);

int main()

{

       const int size = 5;

       int myArray[size] = {8,2,3,9,10};

       cout << "This program uses a function to " << endl

              << "print the contents of an array." << endl;

       printArray(myArray, size); //The array name is passed by reference; see slide #21


} // end function main

void printArray(int inputArray[], int size)

{

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

              cout << "Content of Array Location [" << i << "] = " << inputArray[i] << endl;

}
```

C.2 – Incrementing Array Values

Examine the following code. This program passes an array to the function `incrementArray` (which simply adds 1 to each element of the array). Note that the function `incrementArray` is of type void, which means it does not return any value. Even though the function does not return a value, it is possible to change the array declared in the main function, using the function `incrementArray` (result of pass-by-reference; see slide 21, 22 and 23).

```
#include<iostream>
#include <string>
#include <iomanip>

using std::cout;
using std::cin;
using std::endl;

void incrementArray(int []);

void main()

{

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

       int B[5] = {6,7,8,9,10};

       incrementArray( B );

       for (int i = 0; i < 5; ++i)

              cout << setw( 3 ) << A[i];

       cout << endl << endl;

       for (int i = 0; i < 5; ++i)

              cout << setw( 3 ) << B[i];

       cout << endl;

}

/*setw(n) function, available in <iomanip> library is setting the width of the outputted
value.

This width is provided in terms of number of characters. It means: if setw(3) is called,
three character spaces will be provided for displaying any value followed by that function
call as shown above*/

void incrementArray(int A[])

{

       for (int i = 0 ; i < 5 ; ++i )

              A[i] += 1;

}
```

C.3 – Modifying arrays

In the following program a function `changeArray` is used to modify the contents of an array, which is passed to the function. Note that the function has a type of void since it does not return any value. However, when we modify an array that is passed to function, the original array (in the main function) is also modified (pass-by-reference; see slide 21, 22 and 23). Read the code below, which contains the `printArray` function used in the previous example. As you can see from the program output, the array that is passed from the main function is also modified by the `addArray` function.

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

void printArray(int someArray[], int size);
void changeArray(int someArray[], int size);

int main()
{
	const int size = 5;
	int myArray[size] = {8,2,3,9,10};
	cout << "This program uses a function to " << endl
		<< "add 2 to each element of an array." << endl;

	changeArray(myArray, size);
	printArray(myArray, size);

} // end function main
void changeArray(int inputArray[], int size)
{
	for (int i = 0; i < size; ++i)
		inputArray[i] = inputArray[i] + 2;
}
void printArray(int inputArray[], int size)
{
	for (int i = 0; i < size; ++i)
		cout << "Content of Array Location [" << i << "] = " << inputArray[i] << endl;
}
```

E. Lab Questions

You must complete all three of the following questions to get full marks.

E.1 Improvements to Dice Game

Requirements:

Recall the dice game we developed in the previous lab. You will now implement additional specifications as listed below. The user will now **choose to roll 0, 1, or 2 dice at a time** to reach a total dice value of 50 and **keep track of the total number of dice rolled**.

Specifications:

1. Modify the function *rollDice* that will now take an integer number of dice to roll as a parameter and returns an integer result of counting the rolled dice. For example, if the parameter is 2, you will generate two random numbers between 1 and 6 inclusive, and return their sum. The function will have the following prototype:

   ```
   int rollDice(int numberOfDice);
   ```

2. The function *rollDice* must now record the total number of dice that have been rolled so far since the game started. Every time that *rollDice* is called, the total number of dice that have been rolled so far will increase by 0, 1, or 2, dependent on the parameter passed. You will have a cout statement inside this function responsible for informing the user how many dice have been rolled so far in the game. **This value must be counted using a variable declared inside *rollDice*; you are NOT allowed to use any global variable for this question.**
3. Inside your main function, you will tell the user the goal score of the game (50) and tell them their current score (starts at 0 and gets larger the more dice they roll).
4. Ask the user to choose how many dice they wish to roll (0, 1, 2) and validate their input is one of these choices.
5. Call the *rollDice* function and add the returned result to the user's current score. You should also display the result returned from the *rollDice* function in the main with a print statement. For example,

   ```
   cout << "The result from rolling the dice is: " << result;
   ```

   See the sample output below for a more detailed explanation of required outputs.
6. Your game will loop until the user chooses to roll 0 dice or their score has exceeded 50.
7. When the main game loop ends (user chooses 0 or 50 is exceeded):
   1. If their score is under 50, print their score.
   2. If their score is greater than 50, print a message indicating the user rolled too many.

Sample Output:

```
The goal is a score of: 50
Your current running score is: 0
How many dice would you like to roll (0,1,2): 2
Total number of dice rolled so far: 2
The result of the current roll is: 10

The goal is a score of: 50
Your current running score is: 10
How many dice would you like to roll (0,1,2): 1
Total number of dice rolled so far: 3
The result of the current roll is: 2

The goal is a score of: 50
Your current running score is: 12
How many dice would you like to roll (0,1,2): 2
Total number of dice rolled so far: 5
The result of the current roll is: 5

The goal is a score of: 50
Your current running score is: 17
How many dice would you like to roll (0,1,2): 2
Total number of dice rolled so far: 7
The result of the current roll is: 5

The goal is a score of: 50
Your current running score is: 22
How many dice would you like to roll (0,1,2): 2
Total number of dice rolled so far: 9
The result of the current roll is: 10

The goal is a score of: 50
Your current running score is: 32
How many dice would you like to roll (0,1,2): 2
Total number of dice rolled so far: 11
The result of the current roll is: 11

The goal is a score of: 50
Your current running score is: 43
How many dice would you like to roll (0,1,2): 1
Total number of dice rolled so far: 12
The result of the current roll is: 1

The goal is a score of: 50
Your current running score is: 44
How many dice would you like to roll (0,1,2): 1
Total number of dice rolled so far: 13
The result of the current roll is: 5

The goal is a score of: 50
Your current running score is: 49
How many dice would you like to roll (0,1,2): 0
Total number of dice rolled so far: 13
The result of the current roll is: 0

Game over!
Your score was :49
```

E.2 Grade Calculator with Arrays

Requirements:

Write a program (*according to the specifications outlined below*) that calculates a weighted grade of 4 assignments.

Specifications:

1. A final course grade will be computed based on the user inputs of 4 assignment grades and weights, and will be shown to the user as demonstrated in the sample output, where Final Grade = grade1*grade1 weight + … + grade4*grade4 weight, and '*' sign represents the multiplication operator.

2. You must validate all real number grades to ensure they are between 0 and 100 inclusive. This validation will be accomplished by calling a *isValidGrade* function which will take the real number grade input by the user and return a Boolean value indicating if the input was valid or not. You will ask the user to input a value, pass it to this function, and keep asking the user for an input until the function returns true. The function *isValidGrade* will have the following prototype:

   ```
   bool isValidGrade(double grade);
   ```

3. Ensure that all real number decimal grade weights are between 0 and 1 inclusive and all four weights sum to 1. If they do not sum to 1, ask the user to re-enter all the grade weights until a sum of 1 is obtained.

4. An array of size 4 will be declared in the main function to store grades as double-type data.

5. A second array of the same type and size will be declared to store assignment weights. **Hint: Use the same index in each array to correspond to the same assignment, for example, index 0 in each array will be the grade for the first assignment and weight of the first assignment in the respective (but separate) arrays.**

6. **No marks will be awarded for this question if grades and grade weights are NOT stored in arrays.**

Sample Output:

```
Input the first grade (0-100):
90
Input the first grade weight (0.0-1.0):
0.2
Input the second grade (0-100):
102
Input the second grade weight (0.0-1.0):
0.3
Incorrect grade input!
Input the second grade (0-100):
100
Input the second grade weight (0.0-1.0):
0.3
Input the third grade (0-100):
80
Input the third grade weight (0.0-1.0):
0.2
Input the fourth grade (0-100):
60
Input the fourth grade weight (0.0-1.0):
0.3
The course grade is: 82
Program ended with exit code: 0
```

E.3 Modifying an Array of Integers

Requirements:

Write a program that allows the user to input a set of desired real number values to store in an array, and perform operations on the array.

Specifications:

1. Declare a double-type array with size 10 in the main.
2. Create a function responsible for populating the array. This function will have a void return type and will take the double-type array and an integer-type size as parameters (code writing hints: combination of slide 23 and 17). It will contain all cout and cin statements required (shown in sample output) and will be called from the main. In a loop structure inside the function, ask the user to input 10 real numbers and store them in the passed array. **No marks will be awarded for the question if an array is not used.**

   ```
   void populateArray(double array[], const int size);
   ```

3. After populating the array in the *populateArray* function, you will create an array called arrayToDisplay of same size and data-type. Once this back up array is declared, call the *displayMenu* function from the main. This function will print the following options to the console for the user to select:
   1. Threshold
   2. Reverse
   3. Print
   4. Exit

   Ask the user to input a choice, validate that the choice is a valid menu option, and return the integer-type choice. If the input integer is invalid, keep asking for an input until a valid one is entered. You must write the *displayMenu* function to fit the following prototype:

   ```
   int menuChoice ();
   ```

4. If the user selects option 1, a function *thresholdArray* will be called from the main. Inside this function, ask the user to input a threshold and set all elements of the array that are less than the threshold to 0. The function will have the following prototype:

   ```
   void thresholdArray(double array[], double array2Dispay[], const int size);
   ```

   where, in the parameter list, array[] is the one you have populated, array2Diplay is the one you will store the revised items.

5. Option 2 will require calling the function *reverseArray*. This function will reverse the order of the elements in the passed parameter. This function will have the following prototype:

   ```
   void reverseArray(double array[], double array2Dispay[], const int size);
   ```

   where, in the parameter list, array[] is the one you have populated, array2Diplay is the one you will store the revised items.

6. Lastly, if the user chooses option 3, a function *printArray* will be called that has the following prototype:

   ```
   void printArray(double array[], const int size);
   ```

   Here, you will pass the array you would like to display.

These options will be repeated in an infinite loop, until the user decides to exit by choosing option 4. See the sample output below for more details.

Sample Output

```
Please populate the array with 10 real numbers.
Enter number 1: 12.5
Enter number 2: 11.23
Enter number 3: 10.98
Enter number 4: 49.9
Enter number 5: 98.3
Enter number 6: 76.75
Enter number 7: 74
Enter number 8: 80
Enter number 9: 84
Enter number 10: 87

******************************
** Please choose an option: **
******************************
1. Threshold
2. Reverse
3. Print
4. Exit
Option: 1
Please enter a threshold: 50

******************************
** Please choose an option: **
******************************
1. Threshold
2. Reverse
3. Print
4. Exit
Option: 3
0, 0, 0, 0, 98.3, 76.75, 74, 80, 84, 87

******************************
** Please choose an option: **
******************************
1. Threshold
2. Reverse
3. Print
4. Exit
Option: 2

******************************
** Please choose an option: **
******************************
1. Threshold
2. Reverse
3. Print
4. Exit
Option: 3
87, 84, 80, 74, 76.75, 98.3, 0, 0, 0, 0

******************************
** Please choose an option: **
******************************
1. Threshold
2. Reverse
3. Print
4. Exit
Option: 4
Program ended with exit code: 0
```

F. Additional questions for practice

1. Create a function called **uppercaseString** that converts all the letters in a string to uppercase (e.g. converting “pRograMMing” to “PROGRAMMING”) and a function called **lowercaseString** that converts them to lowercase (e.g. “CAnaDa” to “canada”).

Hint: A string can be considered as an array of characters. You can access any character in a string with help of the specific index. For example, if a string has been declared as **string str = “Hello!”**, then str[0] = ‘H’, str[1]= ‘e’, and so on. The length of the string can be obtained by calling the function *STRINGNAME*.**length()**. For example, str.length() returns a value of 6.