StaticArray Template, Lab Assignment 2

Re-submit Assignment

Due Sep 13 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** cpp and h
Available after Sep 3 at 8am

This is a two-part assignment. You may wait to submit the files for both parts when you are done with both, or submit part 1 first and part 2 later. It should work just like **class Array** from Lab Assignment 1, but with the versatility of a C++ template, and using a different data type for the values stored -- something other than whole numbers. Also, it should allow the main program to specify capacity.

PART 1

Write And Test An Array Class [StaticArray.h *and* **StaticArray.TestDriver.cpp]**
Write a data structures template. The resulting template can be used in any program in place of a C++ array, without having to copy/paste the class code the way you did with **class Array** in Lab Assignment 1.

Requirements. Develop **StaticArray.h** as you write **StaticArray.TestDriver.cpp** with **class StaticArray**, defined and *fully tested*. Write the public interface exactly as specified below -- do not add to, or change the public interface as specified.

1. Write the template for an array of values of unspecified type.
2. Let the capacity be set as part of the template specification, as modeled in the reading.
3. Include a square bracket getter and a setter, both with index range-checking, returning whatever value you wish, if out of range.
4. Include a getter named **StaticArray::capacity( )** to return the data structure's capacity.
5. Initialize *all* data members to their default values in the constructor. That includes setting the array elements to their default data type.
6. Do tests with `int`, `double`, or `char`. Also do tests with an object, like `string`.

Do *NOT* write any other functions in the public interface.

PART 2

Write An Array Application [MyStaticArray.cpp]
Write **MyStaticArray.cpp** using your **StaticArray template**. Use your already-tested and verified H file from part 1.

Exactly as in Lab Assignment 1's **MyArray.cpp**, *this* app lets its user enter as many values as they like, and when that process is completed, lets the user look up values by matching index. Except that these values are *doubles*. Use a **StaticArray** object of capacity 100 to track the values.

In a loop, the app should prompting the user to enter a *pair of numbers* on the same line: a whole number *index* and its corresponding ***floating point*** *value*. Do *not* validate input in the app, because your template should handle out-of-range indexes, and it should allow overwriting an already-entered index. Quit the loop when an *uppercase* or *lowercase* Q is entered for *either* the index or the value. Indexes can be entered in any order -- they don't have to start with zero and go up by one thereafter. It's whatever the user enters.

Your app should keep track of which indexes got entered. Use a *bool* **StaticArray** for that.

After all data entry is complete, the app should:

1. output how many (unique) indexes got entered,
2. output the list of all used indexes and their values, per the example below, and
3. implement an event-controlled loop that prompts for an index value and outputs whether the index is in use or not, and if in use, what is the value stored for that index. Loop until the user elects to stop by entering uppercase or lowercase Q.

Here's a sample of how this should work (user input in blue):

```
Input an index and a value [Q to quit]: 33 1.2
Input an index and a value [Q to quit]: 4 0
Input an index and a value [Q to quit]: 5 300
Input an index and a value [Q to quit]: x 1.7
Input an index and a value [Q to quit]: 33 120
Input an index and a value [Q to quit]: -1 23.4
Input an index and a value [Q to quit]: 2000 -999
Input an index and a value [Q to quit]: q

You stored this many values: 4
The index-value pairs are:
  0 => 1.7
  4 => 0
  5 => 300
  33 => 120

Input an index for me to look up [Q to quit]: 33
Found it -- the value stored at 33 is 120
Input an index for me to look up [Q to quit]: 38
I didn't find it
Input an index for me to look up [Q to quit]: 0
Found it -- the value stored at 0 is 1.7
Input an index for me to look up [Q to quit]: -100
I didn't find it
Input an index for me to look up [Q to quit]: 1000
I didn't find it
Input an index for me to look up [Q to quit]: Q
```

Design the prompts and the output formatting as you like.

And remember -- even H files need an identification comment code block.

Lab Assignment Rubric

Criteria	Ratings							Pts
Fully accurate results, following all specifications view longer description	Works the first time. 70.0 pts	Works on the 2nd try 65.0 pts	Works on the 3rd try 60.0 pts	Works after 4 or more tries. 50.0 pts	Doesn't work after 2 weeks. Partial credit. 20.0 pts	Not submitted within two weeks of the due date. 0.0 pts	Work is not original -- appears to be a marked-up copy of the work of another or previous student. 0.0 pts	70.0 pts
Submits all work on time, fully complete if not fully correct. view longer description	Submitted on time 20.0 pts	Submitted on time, but one or more files are missing or not correctly named. 16.0 pts	Submitted on time, but with missing identification in one or more submitted CPP or H files. 15.0 pts	Submitted on time but not fully complete. 10.0 pts	Late or wholly incomplete! 0.0 pts			20.0 pts
Well-organized and professional quality code. view longer description	Fully meets expectations 10.0 pts	Mostly meets expectations, just needs to be a bit more careful. 8.0 pts	Many areas are well done, but there are a lot of areas that need work. 6.0 pts	Getting there, but needs to be a lot better. 3.0 pts	Needs a lot of work. See the instructor for guidance. 0.0 pts			10.0 pts
								Total Points: 100.0