Lab 4 – Object-Oriented Programming in C++

Computer Science 2413 – Data Structures – Fall 2016

Due by 11:00 pm CST on Thursday, 22 September 2016

This lab is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple program to read and write merchant vessel data, with additional error checking."

Objectives:

1	Create a templated class that encapsulates dynamically resized arrays.	*5 points*
2	Create a class to hold registered ship data on a per country basis.	*5 points*
3	Create overloaded assignment and output operators.	*5 points*
►	Develop and use an appropriate design.	*5 points*
►	Use proper coding style, documentation, formatting, and unit testing.	*5 points*

Description:

This lab builds on Lab 3 by requiring the use of object-oriented programming in C++ programs. Note that Lab 3 required the use of at least one class but there is much more to OOP in C++ than using a single class. Lab 4 requires the implementation to use more sophisticated OOP structures including templates, copy constructors, overloaded assignment operators, and friend classes. Other than these additional requirements, MVP 0.4 must satisfy all requirements of MVP 0.3.

Operational Issues:

From the user's perspective, MVP 0.4 will perform exactly the same as MVP 0.3.

Implementation Issues:

In contrast to the implementation requirements for MVP 0.3 which required the use of a single class to encapsulate the resizable array used to hold country data, MVP 0.4 requires that class to be generalized such that it could be used for resizable arrays of any class. This is accomplished using templates.

Note that the templated resizable array class means that the data for each country must be stored in its own class, which must also be created with its own constructor(s), destructor, etc.

Moreover, whereas MVP 0.3 was rather lax in its requirements for mutator methods, requiring only that mutator methods exist that could add or remove data from the encapsulated array, resizing as appropriate, MVP 0.4 requires the creation of the following mutator methods.

1. A mutator method that takes an object as its parameter and adds the object to the end of the array.
2. A mutator method that takes an object and an index as its parameters and adds the object to the array at the location of the index, shifting the items by one index as necessary to make room in the array.
3. A mutator method that takes an object and an index as its parameters and replaces the object previously at the index given with the new object.

4. A mutator method that takes an index as its parameter and removes the object at that index from the array, shifting items by one index as necessary to fill the gap left in the array by the removal.

All of these methods must resize the array as necessary—doubling or halving it as appropriate. Also, for all of the methods that take an index as a parameter, if the index given is outside the bounds of the array, the method should throw an exception.

In addition, you must define an overloaded assignment operator (`=`) for the country data class and a corresponding copy constructor.

In addition, MVP0.4 must use an overloaded output (`<<`) operator for output of country data. This overloaded output operator must be defined as a friend of the class for country data.

You must thoroughly unit test all aspects of these classes.

You will create simple design document that explains and justifies your design choices. Please make this a PDF file and name it "**design.pdf**" in your submission.

The only libraries you may use for this assignment are `iostream`, `iomanip`, and `string` (`#include <iostream>`, `#include <iomanip>`, `#include <string>`).

You must use good OOP style and documentation.

You must follow C++ conventions for compilation modules, including using a header (`.h`) file for constant variables, prototypes, etc. and a source (`.cpp`) file for implementations of functions, methods, etc. Note that the C++ conventions are different for templated classes than for other classes.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

Due Date:

You must submit an electronic copy of your Visual Studio project **and your design document** to the appropriate dropbox in D2L by **11:00 pm CST on Thursday, 22 September 2016**.