Lab 3 – Classes in C++

Computer Science 2413 – Data Structures – Fall 2016

Due by 11:00 pm CST on Thursday, 15 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	Satisfy all requirements of Lab 2, except for the requirement to use a function.	*5 points*
2	Create a class that encapsulates dynamically resized arrays.	*10 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 2 by requiring the use of an important building blocks for C++ programs—classes. Lab 2 does this by substituting a new implementation requirement for Lab 3 for an old one from Lab 2. This requirement substitution defines MVP 0.3. Other than this requirement substitution, MVP 0.3 must satisfy all requirements of MVP 0.2.

Operational Issues:

From the user's perspective, MVP 0.3 will perform exactly the same as MVP 0.2.

Implementation Issues:

In contrast to the implementation requirements from Lab 2 which required the use of a function to modularize the process of doubling the size of the array used to hold country data, Lab 3 requires that functionality to be encapsulated within a class. This must be a fully formed class with at least two constructors (a no argument constructor that starts at a default size of 10 and a constructor that takes a starting size as an argument), a destructor, accessor and mutator methods, and a display method. The mutator methods must allow for addition to the data stored (which must automatically double the size of the allocated memory when necessary) and removal from the data stored (which must automatically halve the size of the allocated memory if it ever becomes less than half full). This class must be tested and shown to work correctly using unit tests that you must also develop and code, as well as work in the final, integrated program. In particular, you must be sure to test both constructors, the destructor, and the mutators that add and remove data, ensuring that these latter automatically double or halve the size of the allocated memory, respectively.

In addition to these specific requirements, you must carefully consider how to organize the data such that it is easy to manipulate (e.g., using looping constructs), easy to understand (by treating similar data in similar ways), and helps to preserve data integrity (e.g., by grouping related data together in structured ways). **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 programming style and documentation, including making your code modular, using explanatory comments for each section of code, using meaningful variable and method names, using consistent indentation, etc.

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.

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, 15 September 2016**.