Lab 4 – Object-Oriented Programming in C++

Computer Science 2413 – Data Structures – Fall 2018

Due by 11:59 pm CST on Friday, 21 September 2018

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

User Request:

"Create a simple program to read, store, and write NVRA data, using new array library."

Objectives:

1	Satisfy all input/output requirements of Lab 3.	*4 points*
2	Create a templated class that encapsulates dynamically resized arrays.	*4 points*
3	Add additional mutator methods to templated array class.	*4 points*
4	Ensure all indexed array class methods throw exceptions if index is out of range.	*4 points*
5	Use proper design, coding style, documentation, and formatting.	*4 points*

Description:

This lab builds on Lab 3 by requiring the use of additional object-oriented programming structures in C++. Note that Lab 3 required the use of at least two classes and one overloaded operator defined as a friend but there is much more to OOP in C++ than that. Lab 4 requires the implementation to use two additional OOP structures: templates and exception handling. Other than these additional requirements, *VoteR 0.4* must satisfy all requirements of VoteR 0.3.

Operational Issues:

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

Implementation Issues:

In contrast to the implementation requirements for VoteR 0.3 which required the use of a specialized class to encapsulate the resizable array used to hold NVRA data, VoteR 0.4 requires that class to be generalized such that it could be used for resizable arrays of any type. This is accomplished using templates. The class for this resizable array must be called `TemplatedArray`.

Note that each NVRA record will still be an object of class `NvraRecord`, as in Lab 3. However, your templated resizable array class must be able to handle data of any primitive type or any other class, not just of class `NvraRecord`.

Moreover, VoteR 0.4 requires the creation of the following additional mutator methods:

1. A mutator method named `addAt` that takes an item and an index as its parameters (in that order) and adds the item to the array at the location of the index, shifting other items by one place as necessary to make room in the array.
2. A mutator method named `replaceAt` that takes an item and an index as its parameters (in that order) and replaces the item previously at the index given with the new item.
3. A mutator method named `removeAt` that takes an index as its parameter and removes the item 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 indices in VoteR 0.4 must be of type `unsigned long` rather than `int` (to allow for larger arrays and because there is no need for negative values in indexes).

All mutator methods that can increase or decrease the number of items in the array must resize the array as necessary—doubling or halving it as appropriate. Also, for all the methods that take an index as a parameter, if the index given is outside the bounds of the filled portion of the array, the method should throw an exception. (If `addAt` is called with an index equal to the current number of items in the array, it should add the item at the end of the filled portion of the array without shifting any items.)

You should thoroughly unit test all aspects of these classes. They will be tested in zyLabs.

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—all portions of the templated class should go in the `.h` file and there should be no corresponding `.cpp` file.

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 NVRA project to the appropriate lab in your zyBook by **11:59 pm CST on Friday, 21 September 2018**.