Lab 4 – Object-Oriented Programming in C++

Computer Science 2413 – Data Structures – Fall 2017

Due by 11:59 pm CST on Thursday, 28 September 2017

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

User Request:

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

Objectives:

1	Create a templated class that encapsulates dynamically resized arrays.	*5 points*
2	Create a class to hold eclipse data on a per eclipse basis.	*5 points*
3	Create overloaded assignment and output operators.	*5 points*
4	Develop and use an appropriate design.	*5 points*
5	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, *EclipseR 0.4* must satisfy all requirements of EclipseR 0.3.

Operational Issues:

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

Implementation Issues:

In contrast to the implementation requirements for EclipseR 0.3 which required the use of a single class to encapsulate the resizable array used to hold eclipse data, EclipseR 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 `ResizableArray` and must reside at `EclipseR/src/ResizableArray.h` in your project folder.

Note that the templated resizable array class means that the data for each eclipse must be stored in its own type, which in this assignment must be a class with its own constructor(s), destructor, etc. The class for eclipse objects must be called `Eclipse` and must reside at `EclipseR/src/Eclipse.cpp` and `EclipseR/src/Eclipse.h` in your project folder.

Moreover, whereas EclipseR 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, EclipseR 0.4 requires the creation of the following mutator methods.

1. A mutator method that takes an item as its parameter and adds the item to the end of the array. This method must be named `Add`.
2. A mutator method that takes an item and an index as its parameters and adds the item to the array at the location of the index, shifting the items by one index as necessary to make room in the array. This method must be named `AddAt` and the second parameter must be the index.
3. A mutator method that takes an item and an index as its parameters and replaces the item previously

at the index given with the new item. This method must be named `ReplaceAt` and the second parameter must be the index.

4. A mutator method 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. This method must be named `RemoveAt`.

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 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.)

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

In addition, EclipseR 0.4 must use an overloaded output (<<) operator for output of eclipse data. This overloaded output operator must be defined as a friend of the `Eclipse` class.

You must thoroughly unit test all aspects of these classes.

You will create simple design document that explains and justifies your design choices. For your design document, don't include UML or detailed, step-by-step descriptions of how the code works. The UML is more formal than we need (and some students may not know UML) and the step-by-step description of the code should be clear to someone who looks at the code itself (because it should have descriptive variable names, comments, etc.). Instead, provide no more than one page of high-level description and justification of your design choices. For example, for storing the individual eclipses, did you decide to make an eclipse class or an eclipse struct or an array of strings or a single string or something else? If you chose to use a class, what fields and methods did you create? If you chose to use a struct, what fields did you create? Why did you choose what you did?

Once you have created your design document as a PDF document called "**design.pdf**", make a "doc" folder inside your top-level folder for the project (which should be called "EclipseR") and place "design.pdf" inside "doc." That way it will be included in the zip file that you submit for your project.

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—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 Eclipse project **and your design document** to the appropriate dropbox in Canvas by **11:59 pm CST on Thursday, 28 September 2017**.