CSC 415 Spring 2014

Assignment 4 – Aggregation and Operator Overloading

Due: Tuesday April 1, 2014 by 11:59 p.m.
Grade: 50 points as per the rubric; late penalty 5 points per day.

You must design and develop the solution for this assignment **individually**. General questions about design and programming **concepts**, requirements or compile error messages may be asked publicly in class or on Piazza. Students are encouraged to respond to such questions without waiting for my intervention. Specific questions about your code should be addressed to me directly, either through **private** posts on Piazza, after class, or during my office hours. Seeking or providing inappropriate assistance from other students is a violation of TCNJ's Academic Integrity policy.

Objective:
The objective of this assignment is for students to develop proficiency in, and a deeper understanding of, aggregation and operator overloading.

Problem Description:
Assume that you are a software engineer at GF Software Solutions, Inc., and that your company has been awarded a contract to develop a system for Quirky Analytics to perform various forms of analyses on large amounts of data. You have been assigned to develop a prototype for a class called **`Set`** that will be a "container" for different types of data, as per the specifications described below.

A 'set' is a collection of unique entities that share some common properties. Elements in a set are typically not sorted.

For the prototype, assume that the **`Set`** class uses an array to store elements. Also assume that the data stored will be of type **`Complex`**, and that the maximum number of elements in a **`Set`** is 100.

Some of the methods required for the **`Set`** class are:

- ***Input a set*. Overload the >> operator.**
  The user should be able to enter a set from the keyboard or user-specified file. Sample input files will be posted on Canvas, but you will need to create a few of your own.

  Thus the system should allow statements of the form

  ```
  cin >> set1; or
  fin >> set1;
  ```

  where `set1` is a `Set` and `fin` is a file input stream object.

- ***Replace one set with another*. Overload the = operator.**
  The system should allow statements of the form

  ```
  set1 = set2; or
  set1 = value;
  ```

  where `set1` and `set2` are of type `Set`, and `value` is of type `Complex`.

  A statement of the form `value = set2;` is not reasonable and not expected.

- ***Find the union of two sets and store the result in a third*. Overload the + operator.**
  The union of two sets results in a set that contains all the elements in both sets, while retaining the properties of a set. The system should allow statements of the form

  ```
  set3 = set1 + set2;
  ```

  where `set1`, `set2`, and `set3` are of type `Set`. Ensure that there are no duplicates. You must handle the case where the resulting set has more than 100 elements.

- ***Add an element to the set*. Overload the + operator.**
  The system should allow statements of the form

  ```
  set1 = set1 + value; or
  set1 = value + set1;
  ```

  where `set1` is of type `Set`, and `value` is of type `Complex`. Ensure that there are no duplicates. You must handle the case where the resulting set has more than 100 elements.

- ***Find the intersection of two sets and store the result in a third*. Overload the * operator.**
  The intersection of two sets results in a set that contains elements that are common to both sets, while retaining the properties of a set. The system should allow statements of the form

  ```
  set3 = set1 * set2;
  ```

  where `set1`, `set2`, and `set3` are of type `Set`, and `value` is of type `Complex`.

- ***Remove an element from the set*. Overload the – operator.**
  The system should allow statements of the form

  ```
  set1 = set1 - value;
  ```

  where `set1` is of type `Set`, and `value` is of type `Complex`. You must handle the case where `value` is not in `set1`.

- ***Display a set on the screen*. Overload the << operator.**

  The elements of the `Set` should be displayed with proper notation and formatted neatly. For example, a `Set` that contains `4.3+2.1i`, `2.6—8.9i`, `6.5`, `9.7i`, and `–16+6i` is displayed as

  ```
  {4.3+2.1i, 2.6—8.9i, 6.5, 9.7i, -16+6i}
  ```

  The system should allow statements of the form

  ```
  cout << "The set is: " << set1 << endl;
  ```

  where `set1` is of type `Set`. You are not expected to write the set to a file.

- ***Check if a set is empty*.**
  This method returns `true` if the set is empty and `false` otherwise.

- ***Check if a set is full*.**
  This method returns `true` if the set is full and `false` otherwise.

Be sure to include any other methods that you might need for this prototype to work correctly.

Requirements : Design Diagram

Using correct UML syntax, formalize and document your design using the following diagrams:

- Detailed **design class** diagram to model the **structure** of the system.
- **Statechart** to model the overall **behavior** of the **system**.
- Detailed **statecharts** to model the **behavior** of the **overloaded + and * operators**.

Consider how you can appropriately encapsulate data to facilitate information hiding and reuse. Your class diagram must show the relationships between the classes correctly. Be sure to label the diagram with your name and assignment number.

Use LucidChart or a similar UML-aware tool to draw the diagrams. Export the diagram as a `pdf` document. No other format will be accepted. If you have not already done so, you can sign up to join my LucidChart account at https://www.lucidchart.com/e/pulimood.tcnj.edu.

Requirements : Test Case Design

Apply the insights and knowledge you gained about testing methodologies in Assignment 2 to design a set of test cases that will effectively demonstrate successful execution of **all** the functionality and error handling features. A sample format for the test case design is shown below:

Functionality Tested	Inputs	Expected Output	Actual Output
Find the union of 2 sets where the result has less than 100 elements.	setA, setB	setC	setC
Find the union of 2 sets with some duplicates and where the result has more than 100 elements.	setA, setB	setC with duplicates removed, and upto 100 elements stored	setC with additional elements stored outside the array (Note that this would be an error situation.)
Etc.			

Requirements : Implementation and Testing

Implement the **`Set`** class that uses a **fixed size** array that can hold up to 100 **`Complex`** numbers.

Clean up and use the **`Complex`** class that you developed in the class and homework exercises.

Create a simple Makefile to compile your program. See the page on Build Tools on Canvas.

Create a script file to demonstrate that your Makefile works, and the program compiles without errors or warnings. Use the driver provided and the Test Case Design you developed, to demonstrate all the functionality and error-handling capabilities of the **`Set`** and **`Complex`** classes, and that the program executes without runtime errors. Some input files will be provided for testing purposes, but you should create a few new ones of your own.

Bonus 4 points for using a testing tool, like Google Test, to test your program. See the page on Testing Tools on Canvas.

Deliverables:

Submitted in Canvas under 'Assignment 4':

- Design class diagram as a `.pdf` file.
- Statechart to model the overall behavior of the system as a `.pdf` file.
- Detailed statecharts for the overloaded + and * operators as `.pdf` files.
- Test case design document as a `.pdf` file.
- Zip file containing
  - The complete source code for the **`Set`** and **`Complex`** classes.
  - Input files you used to test your program.
  - Script that demonstrates successful implementation.
  - A "readme" text file with a brief description of your program, and how to use it.

General Instructions:

Before starting to work on this project review, in particular, the chapters on arrays and classes in Zyante (Chapters 7 and 8) and the chapters on requirements and design modeling in Pressman (Chapters 6 and 7.3), the related slides, related class and homework exercises, as well as the guidelines below and posted on Canvas.

It is not sufficient for the program to just "work". Rather, it must meet all the requirements, handle errors gracefully, implement efficient algorithms, and use appropriate C++ constructs.

There should be some thought given to why a particular approach would be more efficient or practical than another approach. Document this in the design and/or code where relevant.

Your program must:

- Be implemented in C++.
- Follow the principles of information hiding, encapsulation and responsibility-driven design, i.e. all functionality and error handling must be provided by the appropriate classes.
- Include appropriate documentation in every source code file submitted:
  - Specify identification information at the top;
  - Document every class definition and method;
  - Name identifiers well so that your code is "self-documenting"; and
  - Provide adequate comments for complicated code fragments or algorithms.
- Implement a class (`Set`) that has a fixed size array as one of its attributes and manipulate arrays extensively.
- Overload the binary operators specified; you may need to overload others.
- Read data from user-specified files.
- Ensure that the classes you design are **reusable**.
- Use appropriate formatting commands to display data neatly.
- Handle errors, such as user input, out-of-bound issues and buffer overflow. For example, the program should not crash if the user inputs values of incorrect data types, or the number of elements exceeds the capacity of the array.
- Be compiled on the Mac or Linux operating system using g++.