CSC 415 Fall 2013

Assignment 5 – Dynamic Memory Management

Due: Friday November 22, 2013 by 11:59 p.m.
Grade: 50 points as per the rubric; late penalty 5 points per day.

Objective:
The objective of this assignment is for students to develop proficiency in, and a deeper understanding of, the challenges and trade-offs that exist with safe and efficient management of dynamic memory.

Problem Description:
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.

Assume that you are a software engineer at GF Software Solutions, Inc. Also assume that your company has been awarded a contract to develop a system for Quirky Analytics, to develop a module to perform calculations on large matrices (class **`Matrix`**) of complex numbers (class **`Complex`**). For this assignment you will incrementally develop prototypes as described below.

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

- ***Input a matrix*. Overload the >> operator.**
  The user should be able to enter a matrix 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 >> matrixA; or
  fin >> matrixA;
  ```

  where `matrix1` is a `Matrix` and `fin` is a file input stream object.

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

  ```
  matrix1 = matrix2; or
  matrix1 = value;
  ```

  where `matrix1` and `matrix2` are of type `Matrix`, and `value` is of type `Complex`.

  A statement of the form `value = matrix2;` is not reasonable and not expected.

- ***Add two matrices and store the result in a third*. Overload the + operator.**
  The system should allow statements of the form

  ```
  matrix3 = matrix1 + matrix2; or
  matrix3 = matrix1 + value; or
  matrix3 = value + matrix1;
  ```

  where `matrix1`, `matrix2`, and `matrix3` are of type `Matrix`, and `value` is of type `Complex`.

  Assume that two matrices can be added only if they are of the same size, that is, they each have the same number of rows and the same number of columns.

- ***Multiply two matrices and store the result in a third*. Overload the * operator.**
  The system should allow statements of the form

```
matrix3 = matrix1 * matrix2; or
matrix3 = matrix1 * value; or
matrix3 = value * matrix1;
```

  where `matrix1`, `matrix2`, and `matrix3` are of type `Matrix`, and `value` is or type `Complex`.

  Assume that two matrices can be multiplied only if the number of columns of the left matrix is the same as the number of rows of the right matrix.

- ***Display a matrix on the screen*. Overload the << operator.**
  The output should be formatted neatly in rows and columns, and consistently. For example, negative values should be handled appropriately.

  Thus the system should allow statements of the form

```
cout << "The matrix is: " << matrix1 << endl;
```

  where `matrix1` is of type `Matrix`.

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
Add 2 matrices of the same size	matrixA, matrixB	matrixR	Some values are incorrectly calculated.
Add a matrix and a complex number	matrixA, complexA	matrixS	matrixS

Requirements : Implementation

Implement the **`Matrix`** class using a **dynamic** two-dimensional array that holds **`Complex`** numbers. Assume that the initial size of the **`Matrix`** defaults to 3x4 (three rows, four columns) unless a size is specified by the user.

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

Create a simple Makefile to compile your program.

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 your **`Matrix`** class, 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.

Deliverables:

Submitted in Canvas under 'Assignment 5':

- 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` or `.doc` or `.docx` file.
- Complete source code for the **`Matrix`** and **`Complex`** classes.
- Script that demonstrates successful implementation.

General Instructions:

Before starting to work on this project, review, in particular, the chapters on arrays, classes, and dynamic memory management in Dale & Weems (Chapters 10.7, 10.8, 11, 12, 14.1, 14.5) and Zyante (Chapters 7 – 10), 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, and use appropriate constructs and algorithms.

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 (`Matrix`) that has a dynamic array as one of its attributes.
- Overload the four binary operators specified; you may need to overload others.
- Manipulate arrays extensively.
- Read data from user-specified files.
- Ensure that the classes you design are **reusable** and **safe**, i.e. memory management is handled appropriately to prevent memory leaks and dangling pointers. To accomplish this you will need to add some methods to the **`Matrix`** class.
- Use appropriate formatting commands to display data neatly.
- Handle errors, such as user input and incompatible matrix sizes. For example, the program should not crash if the user inputs values of incorrect data types, or the number of values exceeds the capacity of the array.
- Be compiled on the Mac or Linux operating system using g++.