CSC 260 Fall 2012

Assignment 4 – Abstraction and Aggregation

Part I Due: Tuesday October 16, 2012 by midnight
Part II Due: Friday October 26, 2012 by midnight
Grade As Per Rubric Provided: 60 points, late penalty 10% per day

Objective:
The objective of this assignment is to internalize through practice and experience, important concepts related to software requirements modeling and implementation, such as arrays, abstraction (classes, overloading and templates) and aggregation, using UML and C++.

Requirements:
Assume that you are a software engineer at GF Software Solutions, Inc. and have been given a choice of projects to work on. Regardless of which project you choose, your program must have at least **four** well-designed classes that together

- Demonstrate aggregation,
- Have parameterized constructors,
- Overload the stream insertion and extraction operators, i.e. **<<** and **>>**,
- Overload at least two additional binary operators, like **+** and `*`,
- Demonstrate the use of template classes, and
- Implement an interesting algorithm that requires iteration and selection.

In addition, your program must

- Manipulate arrays of user-defined objects extensively, and
- Provide user interaction, including the ability to use and test all the functionality, through a separate class (user interface class).

The driver, i.e. the `main()` function should merely instantiate the objects and invoke the appropriate methods needed to demonstrate that your classes meet the above requirements.

Assume that any classes you design will be used by you or your colleagues for other projects, i.e. they should be **reusable**. In designing and implementing classes, 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. 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 in the design and/or code where relevant.

Review the projects below and select **one** to analyze, implement and test.

Project A: Be Creative.

Propose a project that interests you, that will use a reasonably complex algorithm and meet all the requirements specified above.

Submit for approval in SOCS Dropbox "Assignment 4", a coherent, typed proposal with sufficient details of the specifications by midnight **October 9, 2012**. Projects without prior approval will not be accepted.

Project B: Polynomials.

You have been assigned to design and develop a system that will enable users to manipulate different types of polynomials as defined in math.

A polynomial may be viewed as a 'list' (or array) of terms, where a term is determined by the coefficient and degree.

For example, consider the polynomials $6x^3 + 5x^2 + 2x$, $x^4 - 4x^2 + 1$, and 0.

`6x`^{3}, `5x`^{2}, and `2x` are the terms for the first polynomial, while `x`^{4}, `-4x`^{2}, and `1` are the terms for the second polynomial. `0` is the only term in the third polynomial.

In the term `6x`^{3}, the coefficient is `6` and the degree is `3`.
In the term `x`^{4}, the coefficient is `1` and the degree is `4`.
In the term `2x`, the coefficient is `2` and the degree is `1`.
In the term `1`, the coefficient is `1` and the degree is `0`.
In the term `0`, both the coefficient and degree are `0`. Typically, if the term is `0`, it is not displayed. However, since this is the only term in the third polynomial, it is displayed.

For this assignment, assume that:

- All polynomials are univariate, i.e. the polynomials are in one variable, **`x`**.
- A polynomial can have a maximum of 10 terms.
- Degrees are natural numbers, i.e. integers greater than or equal to 0.
- Coefficients can be integers, floating point (float or double), or Complex numbers.

An instance of the `Term` class stores the `coefficient` and `degree` of a single term. This class provides all operations necessary to support the operations required for the `Polynomial` class.

A `Polynomial` consists of a fixed size array of type `Term`, with the terms stored in descending order of degree. A `Term` with coefficient `0` will not be stored, except when it is the only term in a polynomial.

Some of the methods required for the Polynomial class are:

- ***<u>Input a polynomial</u>*. Overload the >> operator.**
  The user should be able to enter a polynomial from the keyboard or user-specified file. Sample input files will be posted in your project team's svn repository, but you will need to create a few of your own.

  Thus the system should allow statements of the form

  ```
  cin >> poly1; or
  fin >> poly1;
  ```

  where `poly1` is a `Polynomial` and `fin` is an input stream object.

  Don't make assumptions about the order in which terms are input, i.e. they can be entered in any order of degree. If two terms with the same degree are entered, they should be added together to get one term.

- ***<u>Replace one polynomial with another</u>*. Overload the = operator.**
  The system should allow statements of the form

  ```
  poly1 = poly2; or
  poly1 = number;
  ```

  where `poly1` and `poly2` are polynomials, and `number` is an integer, float, double or Complex number.

  A statement of the form `number = poly2;` is not expected.

- ***<u>Add two polynomials and store the result in a third</u>*. Overload the + operator.**
  The system should allow statements of the form

  ```
  poly3 = poly 1 + poly2; or
  poly3 = poly1 + number; or
  poly3 = number + poly1;
  ```

  where `poly1`, `poly2`, and `poly3` are polynomials, and `number` is an integer, float, double or Complex number.

- ***<u>Multiply two polynomials and store the result in a third</u>*. Overload the `*` operator.**
  The system should allow statements of the form

  ```
  poly3 = poly 1 * poly2; or
  poly3 = poly1 * number; or
  poly3 = number * poly1;
  ```

  where `poly1`, `poly2`, and `poly3` are polynomials, and `number` is an integer, float, double or Complex number.

- ***<u>Evaluate a polynomial and return the result</u>*.**
  The program should allow the user to provide a value for the variable `x`, and then compute the polynomial when that value is substituted for `x`. Thus if the user enters the value `2`, the polynomial $x^4 - 4x^2 + 1$ would evaluate to

  $$1*2^4 - 4*2^2 + 1 = 16 - 16 + 1 = 1$$

- ***<u>Display the polynomial in descending order of degree</u>*. Overload the << operator.**
  The output should be formatted neatly and consistently, for example negative coefficients should be handled appropriately. Use ^ to represent exponentiation.

  Thus the system should allow statements of the form

  ```
  cout << "The polynomial is: " << poly1 << endl;
  ```

  where `poly1` is a Polynomial.

  The polynomial

  $$x^4 - 4x^2 + 1$$

  should be displayed as

  ```
  x^4 – 4x^2 + 1
  ```

  The system should also provide the capability to write the polynomial to a file in the format required for a polynomial to be read in.

<u>PART I</u>

Formalize and document your analysis using the following UML diagrams:

- Detailed **analysis 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**.

Design a set of test cases that will effectively demonstrate **all** the functionality and error handling. A sample format for the test case design is shown below:

Functionality Tested	**Inputs**	**Expected Output**	**Actual Output**

<u>Clearly specify the polynomials you will use for testing.</u>

Implement the `Polynomial` and `Term` classes for integer coefficients. If you chose Project A, implement a "container" class, equivalent to the `Polynomial` class, and a "contained" class, equivalent to the `Term` class. Do not implement the template classes for this part.

Also implement a simple driver to test that all the desired operations work as per the specifications.

CSC 260 Fall 2012

PART II

Implement the `Polynomial` and `Term` classes with coefficients of **any** data type, including but not limited to integers, floats, and Complex numbers. Be sure to complete and test the `Complex` class definition before you attempt to use it for this assignment. If you chose Project A, implement a "container" class, equivalent to the `Polynomial` class, and a "contained" class, equivalent to the `Term` class, that work for data of **any** data type, including at least one user-defined data.

General Instructions:
Before starting to work on this project, review Chapters 1 – 12 and 16 in Dale & Weems, with a special focus on Chapters 11, 12 and 16. Also, review Chapters 4 – 8 in Pressman.

Use an appropriate tool to draw the UML diagrams. For details and instructions, refer to the wiki page at http://tcnjcsc340.pbworks.com/w/page/59147045/UML%20Tools. Export all diagrams in **`pdf`** format for submission.

Implement the program in C++ following best practices for software development. See the "Guidelines for programming assignments" at http://tcnjcsc340.pbworks.com/w/page/34555300/Guidelines-for-programming-assignments.

Compiling and Testing:
Create a '`make`' file to compile your code. For details and instructions refer to the wiki page at http://tcnjcsc340.pbworks.com/w/page/59486008/Make%20And%20Other%20Build%20Tools.

The executable should be named `assign4`***`INIT`***. Replace ***`INIT`*** with your initials.

The assignment must be completed on time and the program compiled on the iMac operating system (machines in Holman 370), since your executable will be tested by one or two of your classmates later this semester. Use the naming conventions specified in this document. Additionally, submit a `readme`***`INIT`***`.txt` file with instructions for correct execution and the input files you used for testing your implementation.

Deliverables:
__PART I:__ Zip together and upload to SOCS in the Dropbox folder 'Assignment 4 Part I' by **October 16**:

- Analysis class diagram modeling the structure of the system.
- Statechart modeling the behavior of the system.
- Statechart modeling the behavior of the operators + and `*`.
- Test case design.
- Script showing successful compilation and execution of the program without `templates`.
- Source code for the program without `templates`.
- Script showing successful testing of the user-defined data type (e.g. `Complex` class).
- Source code for the user-defined data type (e.g. `Complex` class).
- Text files used to input data.
- Approved proposal with any modifications, if you chose Project A.

__PART II:__ Zip together and upload to SOCS in the Dropbox folder 'Assignment 4 Part II' by **October 26**:

- Script showing successful compilation and execution of the complete program.
- Source code for the complete program, including definitions for all classes used.
- Updated analysis documents and test case design, if applicable.
- "`make`" file to compile your code.
- Text files used to input data.
- Approved proposal with any modifications, if you chose Project A.

Check into your team's `svn` repository in the folder '`assign4`' by **October 26**:

- Executable of your program compiled on Mac OS X.

DO NOT POST SOURCE CODE OR DESIGN DOCUMENTS IN SVN.