EEE102 C++ Programming and Software Engineering II

Assessment 2

Classes and Objects

Assessment Number	2
Contribution to Overall Marks	15%
Submission Deadline	March 30, 23:55

How the work should be submitted?

SOFT COPY ONLY !

(MUST be submitted through ICE so that the TAs can run your programs during marking.)
Make sure your name and ID are printed on the cover page of your report.

Assessment Overview

This assessment aims at testing some basic concepts of C++ programming and initiates the routine of code development using the software development process (**SDP**), namely the five main steps of the software development process:

1. Problem statement: formulate the problem.
2. Analysis: determine the inputs, outputs, variables, etc
3. Design: define the list of steps (the algorithm) needed to solve the problem.
4. Implementation: the C++ code has to be submitted as a separate file. Just indicate here the name of the file.
5. Testing: explain how you have tested and verified your C++ program.

You will need to apply this methodology to each one of the following simple exercises.

What should be submitted?

A short ***report*** (up to a few pages of texts plus C++ source codes) detailing for all the questions of the assignment. The answer for each question should follow the SDP method:

a) SDP steps 1 to 3. (30% of the total marks for that question)
b) SDP step 4 (implementation): your C++ source code including the comments. (50%)
c) SDP step 5 (testing): you will explain how you have tested the correctness of your C++ program and will include some sample runs of your C++ Programs. (20%). **Testing result must be shown by screenshot.**

The report in Microsoft Word format **(.DOCX file)** and **C++ source code (with comments)**, for all questions should also be zipped into ***a single file***. (It is a good practice to include comments in your code stating the aim of the program, what are the inputs, what are the outputs, which algorithm is used, who is the author and so on.)

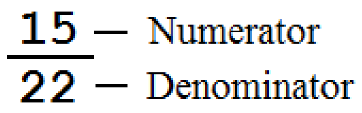
EXERCISE 1 (5 POINTS OUT OF 15)

Define a class called **Season** that is an abstract data type for a season. Your class will have one member variable of type **int** to represent a season (1 for Spring, 2 for Summer, and so forth). Include all the following methods:

1. A constructor to set the season using the first three letters in the name of the season as three arguments;
2. A constructor to set the season using an integer as an argument (1 for Spring, 2 for Summer, and so forth);
3. A default constructor;
4. An input function that reads the season as an integer;
5. An input function that reads the season as the first three letters in the name of the season;
6. An output function that outputs the season as the whole name of the season (Spring, Summer, and so forth) ;
7. and a member function that returns the next season as a value of type **Season**. Test your class appropriately.

EXERCISE 2 (5 POINTS OUT OF 15)

Design a new class to represent a fraction (a ration of two integer values).



The data members of the class **Fraction** are two integers top and bottom, denoting the numerator and denominator, respectively.

```
class Fraction
{
private:
   int top;        // Numerator
   int bottom;     // Denominator
public:
   . . . . . .
};
```

Part 1: Fundamental requirements for the class Fraction.

1. Fractional numbers can be declared with both a numerator and denominator, or simple numerator:
```
    Fraction a;           // represents 1/1
    Fraction b(4,3);      // represents 3/4
    Fraction c(5);        // represents 5/1
```
2. Fractions should be able to act just like other numbers.
   Add, subtract, multiple and divide;
   Compare based on values;
   Input and output.

Part 2: Advanced requirements (Optional, not counted in marking)

3. Fractional number is normalized to ensure that only the numerator can be negative and the value is in least common denominator form:
   2/-3 would be converted into -2/3 automatically;
   15/21 would be converted into 5/7 automatically.
4. Write methods to convert between decimals and fractions.

EXERCISE 3 (5 POINTS OUT OF 15)

Design a composite class represents complex numbers whose real and imaginary parts are Fractions.

1. Write appropriate constructors for this class;
2. Fractions should be able to add, subtract, multiple and divide.