RBE104TC C/C++ Programming Language
Assignment 2

Contribution to the Overall Marks	60%
Issue Date	Sept. 11th 2023
Submission Deadline	Nov. 10th, 23:59, Beijing Time

<u>Assignment Overview</u>:

This assignment is geared towards assessing fundamental coding concepts in C/C++ and initiating the process of code development using the software development process (SDP) discussed in relevant lectures.

In composing the SDP report **[in English]**, we request you to fulfil the following criteria:

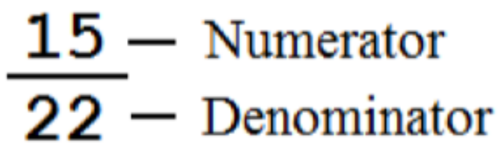
- Problem Formulation: Clearly outline the problem and assigned task.
- Analysis: Identify inputs, outputs, and any additional requirements.
- Design: Specify a sequence of steps (illustrated through flowcharts) necessary to accomplish the given task.
- Implementation: The C code must be submitted in a separate file. Please indicate the file name and provide instructions for user operation.
- Testing: Describe how you tested and validated your code for required functionalities, as well as its robustness.

You are expected to apply this approach to each of the provided simple exercises. Important notes to consider:

- Incorporate clear comments **[in English]** within your code to facilitate understanding.
- Detail your testing procedure and observations made during testing. Provide **screenshot images** along with comprehensive explanations to showcase actual implementation output.

<u>Exercise 1 (40%):</u>

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



The data members of the class Fraction are two integers, top and down, 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 0/1
            Fraction b(3,4);                   // represents 3/4
            Fraction c(5);                     // represents 5/1
```

2. Fractions should be able to act just like other numbers.
   Add, subtract, multiply, and divide;
   Compare based on values;
   Input and output.

Part 2: Advanced requirements

3. Fractional number is normalized to ensure that only the numerator can be negative and the value is in the 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.

<u>Exercise 2 (30%):</u>

Derive a sub-class iFraction from the base class Fraction, which was designed by you in <u>Exercise 1</u> of Assessment 2. The iFraction class represents mixed fractions like $1\frac{2}{3}$;

Part 1: Design the sub-class iFraction:

- It should have a constructor for initialisation;
- It should contain a method to display the mixed fraction on screen;

Part 2: Design an external function convertF (not function member of class Fraction and iFraction) to convert the improper fractions to mixed fractions.

Hint: convertF can be the friend function of those two classes (Fraction and iFraction).

<u>Exercise 3 (30%):</u>

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

- Write appropriate constructors for this class;
- Complex numbers should be able to add, subtract, and multiply.

What should be submitted?

You should submit the followings:

- A concise report (with text spanning a few pages) accompanied by C source codes. This report should delve into the specifics for each question:
    - Detail SDP steps 1 to 3 within the report (Problem Specification + Analysis + Design), accounting for 40% of the evaluation.
    - Encompass SDP step 4 (Implementation + Robustness) with your source code, incorporating comments. Implementation contributes 35% while Robustness adds 5%.
    - Elaborate on SDP step 5 (Testing), elucidating how you validated the correctness, robustness, and thoroughness of your codes. Employ screenshots and ample explanations as verification, encompassing 20% of the evaluation.
- For a comprehensive grading scheme, please consult the Marking Guidelines available on the Learning Mall system.
- The report must be saved in *.pdf format. As for C/C++ source code files, they should be compressed into a single archive file. Your final submission should comprise:
    - The report
    - Zipped source code files.
- The naming convention for the submitted Report and source code files should adhere to the subsequent format:
    - StudentID_LastName_FirstName_AssignmentID.pdf
    - StudentID_AssignmentNumber_ExerciseID.c/cpp
    - StudentID_LastName_FirstName_AssignmentID.zip

As an illustration:

The report and C source file would be denoted as follows:

- 1234567_Albert_Einstein_1.pdf
- 1234567_1_1.c
- 1234567_2_2.cpp

The ultimate zipped submission file would be named:

- 1234567_Albert_Einstein_1.zip

How the work should be submitted?

Submission should take place electronically through the Learning Mall system, enabling us to execute your software during the assessment. Additionally, feedback will be provided via the same Learning Mall system.