CSCI 301 – COMPUTER SCIENCE II
Assignment 1 - Fraction Class
Due Date: 11:59 pm on September 4, Thursday

Objectives:

1. Students will apply basic constructs of programming languages to write programs.
2. Students will write correct, will-documented and readable programs in a reasonable amount of time.

Problem Description:

Declare and define a class for a fraction number. A fraction in mathematics is defined as **`a/b`**, where **`a`** and **`b`** are integers and called numerator and denominator.

Requirements

Task1

- Define a fraction class that has the following member functions:
    - constructor that initializes the fraction by default arguments.
    - set function that sets the numerator of the fraction.
    - set function that sets the denominator of the fraction.
    - get function that returns the numerator of the fraction.
    - get function that returns the denominator of the fraction.
    - a function that displays the fraction.
- Write the class in **header and implementation files**, and compile it separately from the client program. Name the files as *fraction.h* and *fraction.cpp.*
- Document the class following the example of the ***point*** class posted on D2L.
- Write a test program to show all your member functions work before you move to Task2. Name your program as *project1_task1.cpp*.

Task2

- Add the following **nonmember** functions in your fraction header file and implementation file following the example of modified ***point*** class in the file "*newpoint.h*" and "*newpoint.cpp*":
    - A function that returns the sum of two fractions.
    - A function that returns the difference of two fractions.
    - A function that returns the product of two fractions.
    - A function that returns the quotient of two fractions.

  [Note] To make your implementation easier, no need to simplify the calculated result.

  Name the modified files as *newfraction1.h* and *newfraction1.cpp*.

- Write another test program to show all the operations work correctly before you move to Task 3. Name the program as *project1_task2.cpp*.

  A run of this test program might look like this:

```
>a.out
Enter the first fraction: numerator denominator
3 5
Enter the second fraction: numerator denominator
2 3
The two fractions entered are
f1 = 3/5
f2 = 2/3

The arithmetic operations on these two fractions:
f1 + f2 = 19/15
f1 - f2 = -1/15
f1 * f2 = 6/15
f1 / f2 = 9/10
```

Task 3

Redo the Task 2 using operators.

- Use operator overloading to define the following operations for the fraction class:
  - Sum: + as a member function
  - Difference: - as a member function
  - Product: * as a non member function
  - Quotient: / as a non member function
  - Output: << as a non member function
  - Input : >> as a **friend** function of the class fraction
- Following the example of the ***point*** class for all the documentation.
- Write and document the class in **header and implementation files**, and compile it separately from the client program. Name the files as *newfraction2.h* and *newfraction2.cpp*.
- Write a program that performs all the operations defined above. Name the program as *project1_task3.cpp*.

Other requirements for all three tasks

- For each program, add the following information at the top of the file:
  - Description of the problem to solve
  - Your name
  - Your startID
  - Due Date
  - Instructor
- Add Javadoc style comments in the class definitions and implementations following the **point** class example.
  - For more information about Javadoc style comment, please refer to Appendix C: C++ Documentation Systems from the textbook.

What to Hand In

- Submit all source programs to your class account in **GitHub** and test well.

- Submit the following documents to the drop box **Project1 on D2L**:
  - ✓ *fraction.h, fraction.cpp*, *project1_task1.cpp,* and **the script file of the running result on GitHub**.
  - ✓ *newfraction1.h, newfraction1.cpp*, *project1_task2.cpp* and **the script file of the running result on GitHub**.
  - ✓ *Newfraction2.h, newfraction2.cpp* and *project1_task3.cpp* and **the script file of the running result on GitHub**.

  ** Don't list your program source code in the script file!!!

How to create a script file

You get this by the Linux command `script` which causes everything that passes over the screen to be recorded in the file called **typescript**. Here is how it is done.

1. start the script utility with the following commands:
```
script
g++ project1_task1.cpp fraction.cpp
./a.out
……..(follow the instruction to run the program)
```
2. type `ctl-d` to end the script session.
3. type `ls` and you will see the script file "`typescript`" has been created, where all the running results should be recorded.

Or You can name the script file by
```
script   script_file_name
```
Chose any method to create a script file.

Grading

Requirements	**points**
Javadoc style comments in the program	10
Program correctness for Task1	20
Script file from several test runs on **GitHub**	10
Program correctness for Task2	20
Script file from several test runs on **GitHub**	10
Program correctness for Task3	20
Script file from several test runs on **GitHub**	10
TOTAL POINTS	**100**

Javadoc style comments in the program	10
Program correctness for Task1	20
Script file from several test runs on **GitHub**	10
Program correctness for Task2	20
Script file from several test runs on **GitHub**	10
Program correctness for Task3	20
Script file from several test runs on **GitHub**	10
TOTAL POINTS	**100**

No submission made

Javadoc style comments in the program	0
Program correctness for Task1	0
Script file from several test runs on **GitHub**	0
Program correctness for Task2	0
Script file from several test runs on **GitHub**	0
Program correctness for Task3	0
Script file from several test runs on **GitHub**	0
TOTAL POINTS	**100**