COP 3331 Summer 2017: Programming Assignment 3

Due: Tuesday, 13 June, 11:55 pm

Please include the following files in a zipped folder and submit the zipped file via the assignment link on Canvas. The zipped file should have the name "proj3-xxx.zip" where xxx is your NetID.

- Operator Overload Exercise (50 pts)
  - Class File (15 pts) – Fraction.h
  - Specification File (20 pts) – FractImp.cpp
  - Driver Program (15 pts) – FractDriver.cpp

- README file: A plain text including instructions on how to compile and run your code in the IDE you used. This file should include any special instruction/information that the TA should know to be able to run your code.

Operator Overload Exercise

Write a program that performs basic math operations on fractions. In particular, your program should create objects that represent fractions and overload the +, -, * and / operators so that they compute the results of the fractions correctly.

To make things interesting, **you will enter and display fractions by using overloaded << and >> operators!** (Don't worry, this part has been done for you. Additional details on overloading >> and << operators will be added to next week's slides.)

You will need:

1. A class that contains:
   a. Two private data members:
      i. numerator
      ii. denominator
   b. The following prototypes:
      i. A constructor
      ii. A mutator
      iii. Operator prototypes for +, - * and /
      iv. Operator prototypes for << and >> (included)

2. An implementation file that contains the function definitions for:
   a. The constructor, which should take no parameters but initializes numerator and denominator to 0 and 1 respectively.

   b. A mutator function that can update the members with information provided by the user.
   c. Operator functions for +, -, *, /
   d. Operator functions for << and >> (included)

3. A driver program that tests the class and operator overloads. Your driver should:
   a. Create two Fraction objects
   b. Accept input (and display output) in fractions form

Sample output is shown below:

Sample Output 1:

```
Enter the first fraction in the form a / b:   1 / 4

Enter the second fraction in the form a / b:   2 / 5

Fraction 1 = 1 / 4
Fraction 2 = 2 / 5
1 / 4 + 2 / 5 = 13 / 20
1 / 4 - 2 / 5 = -3 / 20
1 / 4 * 2 / 5 = 2 / 20
(1 / 4) / (2 / 5) = 5 / 8
```

Sample Output 2:

(Pay attention to the addition and subtraction results when the denominator is in common):

```
Enter the first fraction in the form a / b:   1 / 10

Enter the second fraction in the form a / b:   2 / 10

Fraction 1 = 1 / 10
Fraction 2 = 2 / 10
1 / 10 + 2 / 10 = 3 / 10
1 / 10 - 2 / 10 = -1 / 10
1 / 10 * 2 / 10 = 2 / 100
(1 / 10) / (2 / 10) = 10 / 20
```

Extra Credit (10 pts)

Note that the sample output above does not simplify the fractions. Write a private member function that simplifies the fractions after performing the operations. A sample output (with the function) is shown below:

```
Enter the first fraction in the form a / b:   1 / 4

Enter the second fraction in the form a / b:   2 / 5

Fraction 1 = 1 / 4
Fraction 2 = 2 / 5
1 / 4 + 2 / 5 = 13 / 20
1 / 4 - 2 / 5 = -3 / 20
1 / 4 * 2 / 5 = 1 / 10
(1 / 4) / (2 / 5) = 5 / 8
```

(Hint: look at the simplify function in the FeetInches example to see how it is implemented. Of course the logic for this function definition will be much different than the one in the example.)

Overloaded >> and << Functions

Include these prototypes in your class file:

```
   friend ostream& operator<< (ostream&, const Fraction&);
   friend istream& operator>> (istream&, Fraction&);
```

Include these functions in your implementation file:

```
//overload the operator <<
ostream& operator <<  (ostream& os, const Fraction& fraction)
{
   //note that we print out a / as it is simply easier to do so!
   os << fraction.numerator << " / " << fraction.denominator;
   return os;
}

//overload the operator >>
istream& operator>> (istream& is, Fraction& fraction)
{
   char ch;

   is >> fraction.numerator;      //get the numerator
   is >> ch;                      //read and discard the '/'
   is >> fraction.denominator;    //get the denominator

   return is;
}
```