CIS 278 (CS1) Programming Methods: C++

Assignment 11: Classes 3

Skip to Main Content

Learning Objectives

After the successful completion of this learning unit, you will be able to:

- Define syntactically correct overloaded operators in accordance with good programming practice
- Implement friend functions

Assignment 11.1 [45 points]

For this assignment you will be building on your Fraction class. However, the changes will be significant, so I would recommend starting from scratch and using your previous version as a resource when appropriate. You'll continue working on your Fraction class for one more week, next week. For this week you are not required to provide documentation and not required to simplify Fractions.

Please keep all of your code in one file for this week. We will separate things into three files for the next assignment. Your class will go first, then your class member function definitions, then main().

Here are the **client program** and **correct output**.

Your class should support the following operations on Fraction objects:

- Construction of a Fraction from two, one, or zero integer arguments. If two arguments, they are assumed to be the numerator and denominator, just one is assumed to be a whole number, and zero arguments creates a zero Fraction. Use default parameters so that you only need a single function to implement all three of these constructors.

  You should check to make sure that the denominator is not set to 0. The easiest way to do this is to use an assert statement: assert(inDenominator != 0); You can put this statement at the top of your constructor. Note that the variable in the assert() is the incoming parameter, not the data member. In order to use assert(), you must #include <cassert>

  For this assignment, you may assume that all Fractions are positive. We'll fix that next week.

- Printing a Fraction to a stream with an overloaded << operator. Next week we will get fancy with this, but for now just print the numerator, a forward-slash, and the denominator. No need to change improper Fractions to mixed numbers, and no need to reduce.

- All six of the relational operators (<, <=, >, >=, ==, !=) should be supported. They should be able to compare Fractions to other Fractions as well as Fractions to integers. Either Fractions or integers can appear on either side of the binary comparison operator. You should only use one function for each operator.

- The four basic arithmetic operations (+, -, *, /) should be supported. Again, they should allow Fractions to be combined with other Fractions, as well as with integers. Either Fractions or integers can appear on either side of the binary operator. Only use one function for each operator.

  Note that no special handling is needed to handle the case of dividing by a Fraction that is equal to 0. If the client attempts to do this, they will get a runtime error, which is the same behavior they would expect if they tried to divide by an int or double that was equal to 0.

- The shorthand arithmetic assignment operators (+=, -=, *=, /=) should also be implemented. Fractions can appear on the left-hand side, and Fractions or integers on the right-hand side.

- The increment and decrement (++, --) operators should be supported in both prefix and postfix form for Fractions. To increment or decrement a Fraction means to add or subtract (respectively) one (1).

Additional Requirements and Hints:

- You will not be graded on documentation on this assignment. You'll be working on the documentation next week.
- The name of your class must be "Fraction". No variations will work.
- Use exactly two data members.
- You should not compare two Fractions by dividing the numerator by the denominator. This is not guaranteed to give you the correct result every time, because of the way that double values are stored internally by the computer. I would cross multiply and compare the products.
- Don't go to a lot of trouble to find the common denominator (when adding or subtracting). Simply multiply the denominators together.
- The last two bullets bring up an interesting issue: if your denominators are really big, multiplying them together (or cross multiplying) may give you a number that is too big to store in an int variable. This is called overflow. The rule for this assignment is: don't worry about overflow in these two situations.
- My solution has 20 member functions (including friend functions). All of them are less than 4 lines long. I'm not saying yours has to be like this, but it shouldn't be way off.
- Do not use as a resource a supplementary text or website if it includes a Fraction class (or rational or ratio or whatever).

Getting Started
Here are some suggestions for those of you who have trouble just figuring out where to start with assignment 1. Remember to use iterative development. That means start with the smallest, simplest subset of the final product that you can, make sure it works, and then start adding things to it one at a time (preferably the simple things first, if possible).

Start with just a default constructor and a stream insertion operator. For now, don't even worry about mixed numbers, just write the stream insertion operator so that it works with proper fractions. Test this out with a client program something like this:

```
int main(){
        Fraction f1;

        cout << f1;
}
```

(You should get output of "0/1" because you should have initialized the fraction to 0/1 in your constructor.)

If you have trouble getting this far, be sure to let me know ASAP so I can help!

Submit Your Work

Name your source code file according to the assignment number (a1_1.cpp, a4_2.cpp, etc.). This source code file should include your class declaration at the top, followed by the definitions of your class member functions, followed finally by the provided client program. Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. Use the Assignment Submission link to submit the source file. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the program works as required.

Keep in mind that if your code does not compile you will receive a 0.