# CS 11 Data Structures and Algorithms

## Assignment 3: Operator Overloading 1

**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

### Assignment 3.1 [15 points]

This assignment will not be graded for style issues. If your code works correctly, you'll get 15 points. Style will be graded next week.

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

Write a class to handle objects of type Fraction. In its simplest form, a Fraction is just two integer values: a numerator and a denominator. Fractions can be negative, may be improper (larger numerator than denominator), and can be whole numbers (denominator of 1).

This is the first part of a two part assignment. Next week you will be making some refinements to the class that you create this week. For example, no documentation is required this week, but full documentation will be required next week. Also, I expect you to work with just a single file this week. You should begin by copy/pasting the provided **client program** into a file and add your class to the file. The class declaration will be first in the file, followed by the definitions of member functions, followed by the client code. Next week you will divide the program up into three files.

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:**

- 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. I would simply 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).

## Submit Your Work

Name your source code file(s) according to the assignment number (a1_1.cpp, a4_2.cpp, etc.). Execute each program and copy/paste the output into the bottom of the corresponding source code file, making it into a comment. Use the Assignment Submission link to submit the source file(s). 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(s) work as required.