CS 11 Data Structures and Algorithms

Assignment 4: Operator Overloading 2

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 4.1 [75 points]

For technical reasons, some of the requirements below may be the same as what you have done in previous assignments. If so, don't worry about it, just make sure the requirement is met, even if it means just copying code you wrote before.

Here are the **client program**, **data file**, and **correct output**.

This week you'll be making the following refinements to the class that you wrote last week. Your score on this assignment will take into consideration your work on both the previous assignment and this assignment.

Reduce Fractions [5 points]

Add a private "simplify()" function to your class and call it from the appropriate member functions. (If you write your code the way that 90% of students write it, there will be 6 places where you need to call it. But if you call it a different number of times and your class works, that's also fine.) The best way to do this is to make the function a void function with no parameters that reduces the calling object.

Recall that "simplifying" or "reducing" a Fraction is a separate task from converting it from an improper Fraction to a mixed number. Make sure you keep those two tasks separate in your mind.

For now (until you get down to the part of the assignment where you improve your insertion operator) your Fractions will still be printed as improper Fractions, not mixed numbers. In other words, 19/3 will still be 19/3, not 6+1/3. Make sure that your class will reduce ANY Fraction, not just the Fractions that are tested in the provided client program. Fractions should not be simply reduced upon output, they should be stored in reduced form at all times. In other words, you should ensure that all Fraction objects are reduced before the end of any member function. To put it yet another way: each member function must be able to assume that all Fraction objects are in simple form when it begins execution.

You must create your own algorithm for reducing Fractions. Don't look up an already existing algorithm for reducing Fractions or finding GCF. The point here is to have you practice solving the problem on your own. In particular, don't use Euclid's algorithm. Don't worry about being efficient. It's fine to have your function check every possible factor, even if it would be more efficient to just check prime numbers. Just create something of your own that works correctly on ANY Fraction.

Your simplify() function should also ensure that the denominator is never negative. If the denominator is negative, fix this by multiplying numerator and denominator by -1. Also, if the numerator is 0, the denominator should be set to 1.

Better Insertion Operator

Now modify your overloaded << operator so that improper Fractions are printed as mixed numbers. Whole numbers should print without a denominator (e.g. not 3/1 but just 3). Improper Fractions should be printed

as a mixed number with a + sign between the two parts (2+1/2). Negative Fractions should be printed with a leading minus sign.

Note that your class should have only two data members. Fractions will be stored as improper Fractions. The << operator is responsible for printing the improper Fraction as a mixed number. Also, the '+' in mixed numbers does not mean add. It is simply a separator (to separate the integer part from the Fraction part of the number). So the Fraction "negative two and one-sixth" would be written as -2+1/6, even though -2 plus 1/6 is not what we mean.

Extraction Operator

You should be able to read any of the formats described above (mixed number, negative number, whole numbers, etc.). You may assume that there are no spaces or formatting errors in the Fractions that you read. This means, for example, that in a mixed number only the whole number (not the numerator or denominator) may be negative, and that in a fraction with no whole number part only the numerator (not the denominator) may be negative. Note: You may need to exceed 15 lines for this function. My solution is about 20 lines long.

Since your extraction operator should not consume anything after the end of the Fraction being read, you will probably want to use the .peek() function to look ahead in the input stream and see what the next character is after the first number is read. If it's not either a '/'' or a '+', then you are done reading and should read no further. I have something like this:

```
int temp;
in >> temp;
if (in.peek() == '+'){
    doSomething...
} else if (in.peek() == '/'){
    doSomethingElse...
} else {
    doThirdOption
}
```

Hint: You don't need to detect or read the minus operator as a separate character. When you use the extraction operator to read an int, it will interpret a leading minus sign correctly. So, for example, you shouldn't have "if (in.peek() == '-')"

Hint: The peek() function does not consume the character from the input stream, so after you use it to detect what the next character is, the first thing you need to do is get past that character. One good way to do that would be to use in.ignore(), which ignores one single character in the input stream.

Three Files and Namespaces

Split the project up into three files: client file, implementation file, and header (specification) file. Also, place the class declaration and implementation in a namespace. Normally one would call a namespace something more likely to be unique, but for purposes of convenience we will all call our namespace "cs_Fraction". Namespaces are covered in lesson 16.15.

Add Documentation

See **Style Convention 1**, especially Style Convention 1D.

Every public member function and friend function, however simple, must have a precondition (if there is one) and a postcondition listed in the header file. Here is a two part explanation of pre- and post- conditions. You'll have to download these to view them: **part 1**, **part 2**.

The most complex of your function definitions will need additional comment in the implementation file. Most of the function definitions in the implementation file will not need a comment.

Hints about reading input files:

I suggest that you copy the text from the input file webpage(s) and paste it into a file that you have created using your IDE. The files you create when you type in your IDE are always text files (even if they don't end with .txt). If you're using Windows, you could also use Notepad, but there's no reason to open another application when you are already working in your IDE. I strongly suggest that you don't use TextEdit (Mac) or Word, because these do not store files as text files by default.

Where to save your input file so that your IDE can find it:

In Visual C++, right click on the name of the project in the solution explorer. It appears in bold there. Unless you chose a different name for the project, it will be ConsoleApplication1. From the drop-down menu, choose "show folder in windows explorer". The folder that opens up will be the correct place to save your file.

In Xcode, while on the project navigator tab (looks like a folder) you'll see a yellow folder called Products. click the expansion triangle next to it and your project executable should show up. right click it and choose show in finder. The folder it's in pops up and you can add your input file to that folder.

Here is a tutorial video created by a former student about **creating and placing input files in the right spot for Xcode to find them**. It's an alternate method. I would suggest watching this video only if the suggestion above doesn't seem to be working for you. In order for these instructions to work, you must have executed a program in your project at least once. Otherwise the folder named "debug" that the video refers to will not exist.

Extensions are part of the file name. If you want to count the words in a file named "myfile.txt", typing "myfile" or "myfile.cpp" won't work.

For Windows users, before you start this assignment, I suggest that you make sure that Windows is showing you the complete file name of your files, including the extensions. Windows hides this from you by default. In Windows 7 and 8 the procedure is as follows:

1. Choose "control panel" from the start menu
2. Choose "Folder Options" from the list of control panel items. In Windows 8 you may have to type "Folder Options" into the control panel search bar.
3. Choose the "view" tab from the Folder Options window
4. Find the checkbox that says "Hide extensions for known file types".
5. Uncheck that checkbox.

I've received two different suggestions from student about how to do this in Windows 10. One student says that the procedure is exactly the same as that outlined above, except that the "Folder Options" window is now the "File Explorer Options" window. The other student says this: go to file explorer and click "View" on the toolbar, then go to "Options" on the furthest right, click on the drop down arrow and choose "Change folder and search options". Click "View" on the top and uncheck "Hide extensions for known file types".

Please let me know if you find that this doesn't work. Also let me know if you think that one of the two Windows 10 options is clearly better.

Submit Your Work

Name your source code file(s) fraction.cpp and fraction.h. Execute the given client program and copy/paste the output into the bottom of the implementation 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.