Lab 7

Due Mar 26 by 11:59pm **Points** 40 **Submitting** a file upload **File Types** cpp and h
Available until Mar 30 at 1pm

This assignment was locked Mar 30 at 1pm.

Fraction Program, Version 1: Ch 9

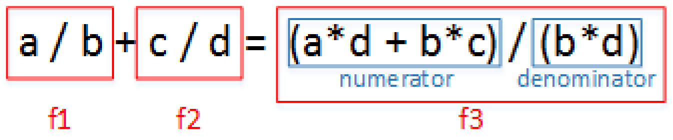
The **General Programming Lab Instructions** apply to this assignment.

A fraction is composed of two integers, a numerator and a denominator. Fractions can be added, subtracted, multiplied and divided. This is just enough detail (data and operations) to make fractions interesting demonstrations of classes and objects. Your submission will consist of three files that follows the pattern set in the **chapter 9 version of the Time example (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/progexample/Time.html)** . Together, fraction.h and fraction.cpp form a server or supplier while calc.cpp demonstrates a client program that uses the fraction class.

Fraction Functions and C++

1. You undoubtedly learned how do basic fraction arithmetic long ago, but if you need a quick review, please see **http://www.basic-mathematics.com/basic-math-formulas.html (http://www.basic-mathematics.com/basic-math-formulas.html)** for the formulas for the four basic arithmetic operations for fractions.
2. **Note that *within* a single fraction the "/" symbol and the horizontal line do NOT mean to divide - they separate the numerator and the denominator - The only time that "/" and the horizontal line mean to divide anything is when they are used as an operator *between* two fractions**
3. a, b, c, and d in the formulas represent the numerators and denominators of two separate fractions in the program - translate them into C++ object notation

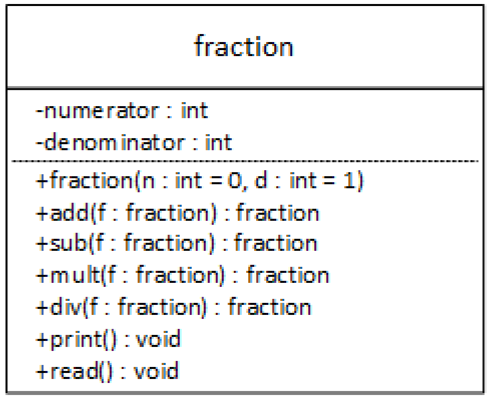
  - a and b represent the numerator and denominator of the left hand fraction
  - c and d represent the numerator and denominator of the right hand fraction
4. create a new instance of the fraction class to represent the result (for addition)
   - the numerator is a*d + b*c
   - the denominator is b*d
5. the following picture illustrates the connection between the formulas and instances of the fraction class: `f3 = f1.add(f2);`



Assignment

Lab 7 consists of three files: Together, `fraction.h` and `fraction.cpp` form a fraction server or supplier that may be used by *any client* code that needs to use fraction objects. These files illustrate class design and implementation. `calc.cpp` is a client that will use some but not all of the fraction class's features. In addition to running the calc program formed by these three files, the test bed will also create a second client program.

The test bed will link your fraction class with **frac_test_1.cpp** to test functionality not exercised by the fraction calculator. This implies that the class and function names must be just as illustrated in the UML class diagram below - remember that C++ is case sensitive. You may use this code to test your lab by temporarily commenting out the calc code and temporarily replacing it with the fraction-test code. REMEMBER TO RESTORE THE CALC CODE BEFORE SUBMITTING THE LAB.


fraction
-numerator : int
-denominator : int
+fraction(n : int = 0, d : int = 1)
+add(f : fraction) : fraction
+sub(f : fraction) : fraction
+mult(f : fraction) : fraction
+div(f : fraction) : fraction
+print() : void
+read() : void

1. Header file: name the header `fraction.h`
    a. Create a class specification for a class named fraction (the class name begins with a lower-case 'f')
    b. The class contains two private member variables, int numerator and int denominator
    c. A single public constructor that serves as a **default constructor (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/constructors.html)** , a **conversion constructor (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/constructors.html)** (int to fraction), and as a **general constructor (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/constructors.html)** . You can make one constructor serve all three roles by using **default arguments (http://icarus.cs.weber.edu/~dab/cs1410/textbook/6.Functions/default.html)** . As illustrated **here (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/constructors.html#default_args)** , you should use 0 for the numerator and 1 for the denominator (why those values?). Use an **initializer list (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/constructors.html#initializer)** to initialize the numerator and the denominator.
    d. The constructor must maintain the fraction in lowest terms (1/2 rather than 2/4, improper fractions like 5/3 are okay). Reduce each fraction to lowest terms in the constructor by finding the greatest common divisor (i.e., the biggest integer that divides both the numerator and the denominator evenly). If your initializer list is correct, you can use this (or similar) code as the body of your constructor:

```
int     common = gcd(numerator, denominator);
numerator /= common;
denominator /= common;
```

    The gcd function is included at the end of the assignment.
    e. Add function prototypes to the class for the functions defined in step 2 below.
2. Functions file: program functions are defined in a file named `fraction.cpp`
    a. Define four arithmetic member functions named `add`, `sub`, `mult`, and `div`. (It's important that you use these names because I will link your fraction class to my code for testing.) Each function must accept one **explicit fraction object (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/organization.html#implicit_explicit)** as an argument **passed by value (http://icarus.cs.weber.edu/~dab/cs1410/textbook/6.Functions/value.html)** and should return the result as a new fraction object returned by value.
        i. Do not change the values stored in either of the two original fraction objects.

  ii. Do not create a temporary fraction object in any of the arithmetic functions (see **Return Efficiency (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/return.html)**)
  iii. Avoid statements like `a = numerator;` or `d = f2.denominator;` as they needlessly clutter the code and look unprofessional.

b. Define a member function named `print`, which is a void function and has no **explicit arguments (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/organization.html#implicit_explicit)** but prints the **implicit argument (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/organization.html#implicit_explicit)** in the form `numerator/denominator`

c. Define a member function named `read` that reads a numerator and denominator from the keyboard into the **implicit argument (http://icarus.cs.weber.edu/~dab/cs1410/textbook/9.Classes_And_Objects/organization.html#implicit_explicit)** . The function shall
  i. Have a return type of void
  ii. Read a single (one) fraction
  iii. Assign the parts separately: one cin >> for the numerator first followed by one cin >> for the denominator
  iv. Each arithmetic operation requires two fraction objects. So, in calc.cpp, you will need to call the read function twice, once for each fraction
  v. Your read function is not required to reduce the faction to lowest terms (but it's not hard to do if your constructor is correct)

d. Include the gcd function at the bottom of this file, but do NOT make it a member of the fraction class

3. Client program: the client program uses the fraction class and replaces the driver in the Time example. Create a simple fraction calculator by modifying the **calc.cpp** program; change "double" to "fraction" and make any other changes needed (e.g., replacing the operators +, -, *, /, <<, and >> with the corresponding fraction functions: change +, -, *, / to add, sub, mult, div respectively; >> to read, and << to print)
  a. Do not alter the order of data input and output in calc.cpp - if you write your own calc.cpp file, you must follow the given input and output order
  b. The program should loop, printing the menu below showing the four arithmetic operations and the option of exiting:

```
                A       add
                S       subtract
                M       multiply
                D       divide
                E       exit
```

  a. If the user selects the exit option, the program must terminate without additional output

b. If the user selects one of the arithmetic operations, the program the prompts for two fractions (numerator first and then the denominator), carries out and displays the results of the operation. (The prompt for the numerator and denominator follows the selection of the operation.)

Euclid's Algorithm

Euclid's algorithm may be used to find the largest integer that divides two other integers (i.e., the greatest common divisor).

1. Copy and paste the "gcd" function code at the bottom of `fraction.cpp`
2. Add a prototype for "gcd" above the fraction class in `fraction.h`.

```
// Euclid's Algorithm for finding the greatest common divisor

int gcd(int u, int v)
{
        u = (u < 0) ? -u : u;                   // make u non-negative
        v = (v < 0) ? -v : v;                   // make v non-negative

        while (u > 0)
        {
                if (u < v)
                {
                        int t = u;              // swap u and v
                        u = v;
                        v = t;
                }

                u -= v;
        }

        return v;                               // the GCD of u and v
}
```

Grading

Upload three files (`fraction.h`, `fraction.cpp`, and `calc.cpp`) to Canvas for grading. **I will link your fraction class to my own code, so insure the class name and the operations are named as specified.**

Lab 7 Scoring

Criteria	Ratings	Pts
fraction Class Name Class name is spelled correctly (including beginning with a lower case letter).		1.0 pts
File Names All file names (fraction.h, fraction.cpp, and calc.cpp) are correct (it's okay if Canvas adds a -1, -2, etc. to the file name).		3.0 pts
Single Constructor Acts As Three The program has one and only one constructor. The constructor serves (i.e., works) as a default, a conversion, and a general constructor. Don't include loops or if-statements, don't use the conditional operator.		5.0 pts
Constructor Reduces Fraction The constructor reduces the fraction to lowest terms (improper factions are okay).		4.0 pts
add The function's name, argument list, and return type are correct. The function does not change either of the original fraction objects. The function produces and returns the correct result that is reduced to lowest terms by calling the constructor at the correct time.		4.0 pts
sub The function's name, argument list, and return type are correct. The function does not change either of the original fraction objects. The function produces and returns the correct result that is reduced to lowest terms by calling the constructor at the correct time.		4.0 pts
mult The function's name, argument list, and return type are correct. The function does not change either of the original fraction objects. The function produces and returns the correct result that is reduced to lowest terms by calling the constructor at the correct time.		4.0 pts
div The function's name, argument list, and return type are correct. The function does not change either of the original fraction objects. The function produces and returns the correct result that is reduced to lowest terms by calling the constructor at the correct time.		4.0 pts
print The function's name, argument list, and return type are correct. The function displays the implicit fraction argument in the correct format.		3.0 pts

Criteria	Ratings	Pts
read The function's name, argument list, and return type are correct. The function correctly reads the numerator and denominator (two separate cin statements) and stores the result in implicit fraction argument.		3.0 pts
calc calc.cpp is fully modified to use the fraction class and its functions, but the original behavior (i.e., the menu, order of data input, etc.) remains unchanged.		5.0 pts
Total Points: 40.0		