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.

Program Description

1. In a file named fraction.h
   a. Create a class named fraction in with two private member variables, an int denominator and an int numerator
   b. Add function prototypes for the functions defined in step 2 below
2. Define the fraction functions in a file named fraction.cpp (see p. 236)
   a. Define four arithmetic functions named add, sub, mult, and div (see Exercise 12, p. 129 for the formulas for the four basic arithmetic operations on fractions - a, b, c, and d represent the numerators and denominators of two separate fractions - a and b represent one fraction and c and d represent a second fraction). Each function should accept one explicit fraction object as an argument passed by value and should return the result as a new fraction object returned by value. The implicit object is the left hand operand and the explicit object is the right hand operand.
   b. Define a function named print, which has no explicit arguments but prints the implicit argument in the form num/denom
   c. Define a function named read that reads a numerator and denominator from the keyboard into the implicit argument. The function should read the values separately (2 separate prompts and 2 separate reads)
3. Create a public constructor
   a. taking two integer parameters, the numerator first, followed by the denominator
   b. use default arguments (see default.cpp in the chap 5 examples) to set the denominator to 1 if a denominator is not specified (this implies that it is possible to convert an integer n to the fraction n/1), and the numerator to 0 if a numerator is not specified (this will allow the constructor to serve as a default constructor).
   c. use the initializer list notation (see pp. 229 - 230).
4. Fractions should be maintained in lowest terms: 1/2 rather than 2/4, but improper fractions (5/3) are okay. Reducing a fraction to lowest terms is most conveniently done at the end of a constructor and involves finding the greatest common divisor (i.e., the biggest integer that divides both the numerator and the denominator evenly). The gcd function is included at the end of the assignment.
   ```
   int common = gcd(numerator, denominator);
   numerator /= common;
   denominator /= common;
   ```
5. Create a simple fraction calculator by modifying the calc.cpp program accompanying the assignment to use your fraction class

a. 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

b. If the user selects the exit option, the program must terminate (the exit function may be used if desired).

c. 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.

Euclid’s Algorithm

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

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

```cpp
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 Blackboard for grading. Make sure that your menu accepts the commands outlined in 5.a. Please do not zip the files.