

Emphasis on: **Classes**

You are to create and test a class to represent a positive fraction (with integer numerator and denominator). The numerator and denominator should not be directly accessible by a client of the class.

Write class functions to provide the following operations:

- 1) create a fraction, allowing client to initialize, or assigning default values of 0 for numerator and 1 for denominator
- 2) assign a new value for a fraction (but if client asks for a denominator of 0, assign instead a default value of 0/1)
- 3) output a fraction in the format n / d
If d is 1, as in $3/1$, output as 3. If n is 0, as in $0/2$, output as 0.
If $n > d$, as in $5/4$, you may output as $5/4$ or as $1 \ 1/4$.
- 4) add two fractions and return the fraction sum
- 5) subtract two fractions and return the fraction difference
- 6) multiply two fractions and return the fraction product
- 7) divide two fractions and return the fraction quotient
- 8) compare two fractions for equal, greater than, and less than, returning true or false
- 9) return the float equivalent for a fraction (for example, $3/4$ is equivalent to $.75$)
- 10) reduce a fraction to its lowest terms (function is provided for you; see the other side)
- 11) return the numerator value
- 12) return the denominator value

Rules for fraction operations:

$$a / b + c / d = (ad + bc) / bd$$

$$a / b - c / d = (ad - bc) / bd$$

$$a / b * c / d = ac / bd$$

$$a / b / c / d = ad / bc$$

$$a / b == c / d \text{ if and only if } ad == bc$$

$$a / b < c / d \text{ if and only if } ad < bc$$

$$a / b > c / d \text{ if and only if } ad > bc$$

You should also create a test driver application that demonstrates calling the Fraction class functions so you can test to be sure your functions are working correctly.

Requirements:

- 1) The class should define the components of only one fraction, even though some of the functions will be operating on two fractions.
- 2) Any calculated fraction displayed should be reduced to lowest terms. A fraction which is greater than 1, such as $5/4$, can be written as $5/4$ or as $1 \ 1/4$.
- 3) The client program should call class functions whenever possible; the client program should not duplicate an operation which is provided by a class function.
- 4) The only class function which does any output is the output function (#3 in the list of class operations to be provided). All other functions should return a value and let the client decide what to do with it.
- 5) You should define your Fraction in multiple files, Fraction.h containing the class definition and Fraction.cpp containing implementations of member functions. Your test driver program should contain a main function, and it should create variables of type Fraction, and demonstrate the use of the Fraction classes member functions.
- 6) Your test driver program should call fraction functions whenever possible rather than recoding an operation which can be performed by a class function.

Extra-Credit Opportunity

Allow for negative fractions to be input (there will be a leading sign, like $-1/2$) or calculated. You can represent a sign either as a separate character field or by storing the numerator value as a signed number.

Try and work ahead and overload operators to do things like add (operator+), subtract (operator-), multiple (operator*) and divide (operator/). Also comparison (operator==) and output (operator<<) could be done as overloaded operators as well.

You may use the following two functions which will reduce a fraction to its lowest terms. In order to satisfy requirement #2 above, you should reduce the resulting fraction after doing the add, subtract, multiply and divide. You might also want to reduce the fraction itself when its value is set (operation #2) or created (operation #1), to make sure that any given fraction is reduced/displayed/represented as it most reduced and simplest form.

Add these prototypes to your class specification:

```
void reduce();  
int gcd(int x, int y);
```

and add these function bodies to your class implementation:

```
// PRE: parameter has values assigned to its components  
// POST: The value returned will be the parameter fraction reduced  
// to its lowest terms. For example, for parameters 1/2, 4/8, 3/6,  
// and 5/10 the fraction 1/2 will be returned.
```

```
void Fraction::reduce ()  
{  
    int divisor;  
  
    divisor = gcd(num, den); // find greatest common divisor  
    num = num / divisor;  
    den = den / divisor;  
}
```

```
// PRE: n and d have values assigned  
// POST: returns the greatest common divisor (the largest integer  
// which can be divided evenly into both n and d). For example, if  
// n is 9 and d is 12 (for the fraction 9/12), the greatest common  
// divisor is 3 because 3 is the largest integer that divides evenly  
// into both 9 and 12. The greatest common divisor of 8/12 is 4.
```

```
int Fraction::gcd (int n, int d)  
{  
    if (d == 0)  
        return n;  
    return GCD(d, n % d);  
}
```