CS 1410 Introduction to Computer Science – CS2
Section 1: MWF 10:30 a.m. – 11:20 a.m.
Section 2: MWF 1:00 p.m. – 1:50 p.m.
Instructor: Xiaojun Qi
Assignment #3

Given: Wednesday, January 29, 2014
Due: 11:59 p.m. Saturday, February 8, 2014
Total Points: 30 points

Introduction: A rational number is one that can be represented as a fraction whose numerator and denominator are both integers. For example, 1.6875 can be represented as 27/16. The square root of 2, on the other hand, cannot be represented as the ratio of two integers and thus is not rational. Rational numbers can be useful in some computations. For example, the C++ statements

```
float x = 1.0 / 3.0;
float y = 3.0 * x;
```

might not yield a value of 1.0 for y because of the limitations on the accuracy of floating point numbers (there are only so many bits allocated to hold the floating point number 0.33333...). Storing x and y as rational numbers, with numerators and denominators stored separately, provides a means of calculating 1/3 * 3/1 so that the product is *exactly* 1, not approximately 1.

This assignment involves creating a class called **Rational** that performs several **representative** arithmetic operations with fractions. This class uses integer variables to represent the private data members, namely, the numerator and the denominator. It also uses a string variable to represent the private data member as "numerator/denominator" (where both numerator and denominator are integers). **This Rational class must have only the following public methods and overloaded operators.** The class, obviously, contains other private data and members.

```
// Constructors and copy constructor
Rational() ;
Rational(int);
Rational(int, int);
Rational(string) ;
Rational(const Rational &);
// You may use the "stringstream" streaming class to help with the conversion from the two
// integers to the string. Its use will be reviewed in class.
// For the implementation of Rational(string) function, please check out Table 12-8 (page 809) to
// find the appropriate member functions to do the task. Sample functions are: find and size.

// Assignment operator
Rational &operator=(const Rational &);
Rational &operator=(const int) ;  // convert an int data to its Rational representation
Rational &operator=(const string) ;

// Arithmetic binary operators
Rational operator+(const Rational &);
Rational operator+(int) ;
Rational operator+(string) ;
```

```
/****
Compound assignment operators.  Make sure you only use two lines of code (i.e., two
statements) to implement each of the compound assignment operators.  In other words, you
can directly use the overloaded operator inside your function to remove a lot of duplicated
code.
****/
Rational &operator+=(const Rational &);
Rational &operator+=(int) ;
Rational &operator+=(string) ;

// Relational operators
bool operator==(const Rational &);
bool operator>(const Rational &);

// Prefix and Postfix operators
Rational operator ++ () ;      // prefix ++
Rational operator ++ (int) ;   // postfix ++

operator double();  // convert a Rational object to a double
```

Additionally, you must implement both << (insertion) and >> (extraction) streaming operators. For the >> streaming operator, the input must be a '/' separated sequence of numerator and denominator. For the << streaming operator, the output must be a '/' separated sequence of numerator and denominator in legal ranges. When the denominator equals 1, the output should be numerator itself. Some functions outside of the class may also be added to solve the problem. Please carefully read the following examples where some parts are highlighted to catch your attention.

1. Constructors: New Rational objects are initialized to zero by default. A Rational object may also be initialized with a given numerator; or numerator and denominator;or a string. For example, the following are valid Rational object declarations and initializations.

```
Rational a;                          // default to 0/1
Rational b(a) ;                      // 0/1
Rational c(3), d(2, 3), e("2/4");    // 3/1, 2/3, 1/2
```

Note: **the constructor always stores the Rational object in the simplified form.** For example, the fraction 2/4 is stored in an object as 1/2. The denominator cannot be 0's at any time.

2. Rational objects may be added by another Rational object or an int value, or a string object. For example, the following are valid arithmetic operations.

```
b = a + c ;   a = 2 + c ;  d = e + 4 ;   a = "1/2" + b ;   d = c + "4/10" ;
```

3. Rational objects may be input and output using >> and << operators. A Rational object is always input and output as a "/" separated sequence of numerator and denominator. For example,

```
cout << "Input a rational number";
cin >> a;                                  // user enters 6/8,
cout << a << endl ;                        // output is 3/4
a += 2 ;
cout << "That rational number plus 2 is: " << a << endl;       // output is 11/4
b = 3 + a;    // Hint: You need to implement one outside function to accomplish the task
cout << "That rational number plus 3 is: " << b << endl;       // output is 23/4
cout << b++;                               // output is 23/4
```

```
        cout << b << endl;                          // output  is 27/4
```

4. Rational objects may be assigned an **int** number. For example,

```
        b = 12;
        cout << b << endl;                          // output is 12
```

5. A Rational object assigned to a **double** variable results in the equivalent double number. For example,

```
        double x = b;  // where b = 3/2.
        cout << b << " is equivalent to " << x << endl;     // output is: 3/2 is equivalent to 1.5
```

6. Rational objects may be compared using the standard relational operators. For example,

```
        if (a > b)
           cout << "a is greater than b";
```

Some comments about the above

- The + operator does not change the value of the current instance of the Rational.
- The "+=" operator does change the value of the current instance.
- For the operators with the parameter of (int), the integer variable represents the numerator in the rational number.
- Do not deviate in any manner from the syntax and naming conventions noted above. The grader will be running a test driver program versus your class and if your class is defined differently, it won't compile...and you don't want that :)

In addition to the Rational class, write some driver code (in the main function) that fully demonstrates every operator in your class works correctly. Remember, it is your responsibility to prove to the grader your code works, the grader should not have to guess and label your output.

You might find the Simplify and ComputeGCD functions helpful.

```
void Simplify(int &num, int &denom)
{
        int gcd=ComputeGCD(num,denom);

        num=num/gcd;
        denom=denom/gcd;
}

int ComputeGCD(int num, int denom)
{

        int remainder= denom % num;
        while (remainder != 0)
        {
                denom = num;
                num = remainder;
                remainder= denom % num;
        }

        return num;
}
```