OOP - Comp 345: Overloaded Operators and Rational Numbers

Files: Rational.h, Rational.cpp, Test.cpp

Create a class to represent rational numbers and a test program to verify that all required functionality works.

Constructor(s):
The user of the class should be able to create an instance of a rational number passing in 0-2 arguments. If no arguments are passed in the value should be 0, for example 0/1. If one argument is passed in, it should be considered a whole number and thus be represented over one in radical form, for example 5/1. If two numbers are passed in, the first should be considered the numerator the second the denominator. Be sure to reduce the fraction completely.

Unary Operators:

- ++ post and pre increment operators. These operators should add one to the number. Be sure to reduce the resulting rational.
- -- post and pre decrement operators. These operators should subtract one from number. Be sure to reduce the resulting rational.
- +=, -=, *=, \= operators. These operators should modify the numerator appropriately according to the rValue passed in.
- () function operator. Overload this operator to return a string in the form "n/d".
- double() type cast operator. Overload this operator to return a double for the numerator divided by the denominator.

Binary Operators (may be non-members)**:**
All of the following binary operators overloaded. Each of these operators should work with long and other rational numbers. Be sure to reduce the resulting radical.

- = for assignment
- +, -, *, /
- >, < , >=, <=, ==, !=
- ^ to perform exponential operations. The r-value only needs to be a whole number. Feel free to work with rational exponents but it is not required.

Note about stream insertion and extraction:
The stream insertion << and stream extraction >> operators should get and show the value of the radical.
The stream-extraction operator is only required to get the rational from the user in this form n/d
The stream insertion operator should show the radical with a forward slash '/' appearing between the numerator and denominator with one exception. If the denominator is 1, only the numerator should be displayed.

Private Methods:
A private reduce fraction and common denominator functions will be useful. Add any others that you need.

NOTE:

- ***Watch out for negative values in both the numerator and denominator.***
- ***Do not have redundant functions.***
- ***Use const as much as possible (where appropriate). When possible pass Rationals to functions by const reference.***

Rubric:

Criteria	Points
Constructors	4
Unary Operators	10
Binary assignment operator	2
Binary arithmetic operators	10
Binary comparison operators	10
Stream insertion and extraction operators	4
Create a test program that tests **all** operators and methods of the rational class. The output of the test program should clearly show the grader that each operator functions correctly.	10
Total	**50**
Possible max penalties	-4 improper or missing const -4 redundant functionality -3 multiple returns from methods -2 missing {} -8 poor function/variable names -4 division by 0 -4 missing reductions after operations

These are just some example methods. Yours are not required to be identical to these.

```
//Rational.cpp
#include"Rational.h"
#include<sstream>

Rational::Rational(long NewNumerator, long NewDenominator)
{
       numerator = NewNumerator;
       denominator = NewDenominator;
       Reduce();
}
const Rational& Rational::operator=(const Rational & rValue)
{
       numerator = rValue.numerator;
       denominator = rValue.denominator;
       Reduce();
       return *this;
}
Rational& Rational::operator++()
{
       numerator += denominator;
       Reduce();
       return *this;
}
Rational Rational::operator++(int Garbage)
{
       Rational result = *this;
       numerator += denominator;
       Reduce();
       return result;
}
```

```
bool Rational::operator==(const Rational & rValue) const
{
	bool result = true;
	if(numerator != rValue.numerator || denominator != rValue.denominator)
	{
		result = false;
	}
	return result;
}
string Rational::operator()() const
{
	std::stringstream stream;
	stream << numerator << "/" << denominator;
	return stream.str();
}

ostream &operator<<(ostream & out, const Rational & rational)
{
	out << rational.getNumerator() << "\\" << rational.getDenominator();
	return out;
}
long Rational::getNumerator()const
{
	return numerator;
}
long Rational::getDenominator()const
{
	return denominator;
}
long Rational::LeastCommonMultiple(long x, long y) const
{
	bool Continue = true;
	long result = x;

	while (result % y != 0)
	{
		result += x;
	}

	return result;
}
long Rational::GreatestCommonDivisor(long x, long y) const
{
	long remainder = x % y;

	while(remainder != 0)
	{
		x = y;
		y = remainder;
		remainder = x % y;
	}

	return y;
}
void Rational::Reduce()
{
	long GCD = GreatestCommonDivisor(numerator, denominator);

	numerator = numerator/GCD;
	denominator = denominator/GCD;
}
```