CS 132

Project 3: "Rational Number Class" - *individual project, no partners*

Name: David Landry

A Rational Number is any number which can be written in the form p/q, where p and q are both integers, and q is not **zero**. In C++, Rational Numbers can be approximated as floating point types, but are often inaccurate. It would be nice to have a data type which could accurately represent a Rational Number in its p/q form. That is what this project should accomplish.

You will then both create and implement a new class: CRational (*defined in a new namepace*). It should contain *at least* the following private data members (*properties*):

- *m_numerator* (of type *long)*
- *m_denominator* (of type *long*)

It should also contain *at least* the following public member methods:
NOTE: all of these methods, including the overloaded operators, should be ***defined as members*** of the class
NOTE: all of these methods that do not change the object should be declared as ***const***.
ALSO: There should be **NO** friend functions declared *(including operator<< and operator>>)*

- at least one default constructor. This (*these*) Constructor(*s*) must handle the cases of 0, 1, or 2 ***long*** parameters. Defaults should be set as ***0L*** for the numerator and ***1L*** for the denominator. It should call the *setProperForm* method described below. Should throw an ***invalid_argument exception*** for *divide-by-zero*.
- a constructor that takes in one ***int*** as the parameter.
- a conversion constructor that takes in a ***double*** as the parameter.
- Two accessors, designed to get either the numerator or denominator.
- **NOT** two mutators, designed to change either the numerator or denominator. Again...**do not** include these in the class.
- a *void print* method. Should take in an ostream reference. Should be defined to allow printing a fraction in the form p/q. If the *denominator* is 1, then no denominator should be printed.
- another *void print* method that takes no parameters. It should simply call print(cout)
- an *invert* method. Should change the current object to be its own *multiplicative inverse*, and return itself, *i.e.*, given an object with a value of p/q, *the object should change to have the value of* q/p (*returns the changed object*). Should throw an ***invalid_argument exception*** for *divide-by-zero*.
- A *toThePower* method. Should take in a *long* parameter. Should return the a *CRational* which is equal to the current object raised to that power, Should be calculated using repeated multiplication (**do not** use the ***pow*** function). Should throw an ***invalid_argument exception*** for *zero-to-the-zero-power*.
- A *compareTo* method *(for two CRationals)*. Should return a *long* which is less than 0 if the first < the second, a 0 if the two are equal, and a *long* which is greater than 0 if the first > the second. (Simply returns the *product-of-the-extremes* minus the *product-of-the-means)*.
- A * operator *(for two CRationals).* Should return a CRational that is created by multiplying the two rational numbers and using *setProperForm (could be done implicitly by calling the constructor).*
- A / operator *(for two CRationals).* Should return a CRational that is created by dividing the two rational numbers and using *setProperForm (could be done implicitly by calling the constructor). Should throw an* ***invalid_argument exception*** *for divide-by-zero.*
- A + operator *(for two CRationals).* Should return a CRational that is created by adding the two rational numbers and using *setProperForm (could be done implicitly by calling the constructor).* Should call the LCD method described below to find the common denominator before adding, *in order to help keep intermediate values from going out of range.*
- A - operator *(for two CRationals).* Should return a CRational that is created by subtracting the two rational numbers and using *setProperForm (could be done implicitly by calling the constructor). Could be done by calling the binary+ and unary- operations.*

- A - operator *(for **one** CRational).* Should return the *additive* inverse *(negative)* of the current rational number by multiplying it by -1 and calling *setProperForm (could be done implicitly by calling the constructor).*
- A *= operator *(for two CRationals).* Should assign the product of the two rational numbers to--and return-- the current object..
- A /=operator *(for two CRationals).* Should assign the quotient of the two rational numbers to--and return-- the current object. Should throw an ***invalid_argument exception*** for *divide-by-zero*.
- A += operator *(for two CRationals).* Should assign the sum of the two rational numbers to--and return-- the current object.
- A -= operator *(for two CRationals).* Should assign the difference of the two rational numbers to--and return-- the current object.
- Two overloaded ++ operators *(for one CRational).* Should be both pre- and post-increment for a rational number. The resulting number should have a value equivalent to the result of +=1 *(e.g. if a is 1/4, ++a would make a be 5/4).*
- Two overloaded -- operators *(for one CRational).* Should be both pre- and post-decrement for a rational number. The resulting number should have a value equivalent to the result of -= 1 *(e.g. if a is 5/4, --a would make a be 1/4).*
- Six relational operators:
    - An = = operator *(for two CRationals).* Should return true if the two rational numbers are equal (returns *the product of the means = = the product of the extremes*).
    - A **!** = operator *(for two CRationals).* Should return true if the two rational numbers are not equal (returns !(*left = = right*)).
    - A < operator *(for two CRationals).* Should return true if the first rational number is less than the second (returns *the product of the extremes < the product of the means*).
    - A <= operator *(for two CRationals).* Should return true if the first rational number is less than or equal to the second (returns !(*left > right*)).
    - A > operator *(for two CRationals).* Should return true if the first rational number is greater than the second (returns *the product of the extremes > the product of the means*).
    - A >= operator *(for two CRationals).* Should return true if the first rational number is greater than or equal to the second (returns !(*left < right*)).
- A **!** operator *(for one CRational).* Should return true if the numerator is 0.
- A ~ operator *(for one CRational).* Should just return *the multiplicative inverse* of the current object. It should not change the current object. Should throw an ***invalid_argument exception*** for *divide-by-zero*.
- a cast operator to bool. Should return true if the numerator is not 0
- a cast operator to double.
- a cast operator to std::string.

The following member method should be ***static***, so that it could be called directly from the class.

- *GCF (for two **longs**).* This should return the *greatest common factor* of the 2 ***longs***. It should probably use ***Euclid's Algorithm***, which is as follows :

  First, if either number is 0, the answer is 0. After checking that, take the absolute value of the two numbers.(*integer*)divide the larger number by the smaller number. If the *remainder* is greater than zero, divide the previous *divisor* by the *remainder*. Repeat this process until the *remainder* is zero. The *greatest common factor* of the two original numbers is the last *remainder* before the zero.

  For example, if the fraction is *4/12*, you would take *12 / 4*. Since the *remainder* is 0, the *greatest common factor* is *4*. Dividing both *4* and *12* by *4* would give you *1* and *3*, or *1/3*.
  If the fraction is *8/12*, you would take *12 / 8*. Since the *remainder* is *4*, you would take *8 / 4*. Since the remainder this time is zero, the *greatest common factor* is *4*. Dividing both *8* and *12* by *4* would give you *2* and *3*, or *2/3*.

  P.S.: this is a good function to write recursively!

The class should also contain *at least* the following private member methods:

- *setProperForm (void)*. This should:
  - make sure there is no zero in the denominator-- *throw an exception* if it is 0.
  - make sure there is no negative in the denominator
    e.g., -3/-4 should become 3/4; 3/-4 should become -3/4.
  - reduce the fraction to simplest form. To do this, you will need to call the GCF method. Also,
    - A fraction with a 0 *numerator* should be set to 0/1
- *LCD (for two CRationals)*. This should return the *least common denominator* of the 2 fractions. It should probably use the following algorithm:

  Divide one *denominator* by the *greatest common factor* of the two *denominators*, then multiply by the other *denominator (multiplying first could provide an intermediate value that is out of range)*

  however, if either denominator is 0, the lcd is 0

Outside of the class:

- an overloaded << operator should call the *print* method.
- an overloaded >> operator should be defined to allow inputting a *CRational* using an istream. should input a fraction in the form p/q *(or even p / q)*, an integer or a **real** number with a decimal point.

The driver program for this class should declare objects of this type, and test **all** of the methods. You may want to use the CComplex Driver as a *partial* model...you will need to write additional code.

WARNING: THIS CODE, BY ITSELF, DOES NOT CONSTITUTE A SUFFICIENT TESTING OF THE **CRational class**. You must still write a **FULL** test plan, sufficiently testing **each** method of the class.

Points Possible: 100

Deliverables:

Physical:

The Project should be turned in inside a clear plastic file folder. This folder should have a simple flap to hold paper in place--NO buttons, strings, Velcro, etc. Pages should be in order, not stapled.

- Assignment Sheet (printed pdf from the web), with your name written on it, as a cover sheet.
- Printed Source Code with Comments *(including heading blocks -- a file header for each file plus a function header for each function. Describe parameters, any input or output, etc., no line wrapping). Print in portrait mode, 10 - 12 point font.*

Electronic:

- All .h, .cpp, .exe(Release Version) zipped together. Do not use rar or any archive format other than zip. Rename the file: "<YourName>_p3.zip".
- Sample Output (as .rtf -- run the program, copy the window using <*Alt*|*PrtScn*>, paste into Paint, invert colors *(*<*Ctrl*|*Shift*|*I*>), copy, open Wordpad, save.)
- A simple test plan including explanations of any discrepancies and reasons for each test. Show actual input and ALL values output as well as ALL expected output. Test each possible action. Save as .xls, xlsx, .doc or .docx file
- Submit this single zip file by going to canvas, select this class, select the Assignment tab on the left, select Assignment 3, select the submission tab at the top right, find the file, and Submit.

Due:

Section A: *Monday, February 29, 2016 9:30 a.m. (beginning of class)*
Section B: *Tuesday, March 1, 2016 2:50 p.m. (beginning of class)*