3460:210

Programming Project: Rational

# Project: Rational numbers

In this assignment, you will implement a Rational number class and finish writing a small rational number calculator.

# Getting Started

Go to the Project folder on Springboard, and locate the content for the project (this will align with the specific project number that we are working on).

Investigate the contents of that directory. There’s a lot there. You should have the following files:

• rational.hpp and rational.cpp — The definition of your Rational class and its associated functions will go here. Most of your work will be within these files.

• test\_rational.cpp — A small test suite for your Rational number class. You will not need to modify this file, but get very familiar with it. This is our test driver.

These are downloadable files that you will want to convert to the appropriate named (extension) file. You can try building the project, but it won’t work. The test suite and the calculator are written in terms of functions that you must provide.

# Class requirements

The Rational class must have the following:

1. A default constructor that initializes the rational number to 0.
2. A constructor that takes a single integer value n and initializes the rational number so that it is equal to n / 1.
3. A constructor that takes a numerator and denominator and guarantees that the resulting object holds a valid rational number (i.e., the denominator cannot be 0). Note that these numerator and denominator may need to be adjusted during initialization in order to normalize the representation to support other operations. (such as 2/4 -> 1/2; 1/-2 -> -1/2; 6/3 -> 2/1; 8/6 -> 4/3 etc)
4. Overloads of equality operators (== and !=) to compare rational numbers. The == operator must return true when two rational numbers are the same (1/2 is the same value as 2/4).
5. Overloads of ordering operators (<, >, <=, and >=) to order rational numbers. These must perform the expected orderings (e.g., 1/4 is less than 1/2).
6. Overloads of arithmetic operators (+, -, \*, and /). These must perform the expected numeric operations (e.g., 1/2 + 1/2 == 1/1) and the result must be normalized.

Overloads of streaming operators (<< and >>) have been provided for you.

Define the Rational class and declare all associated functions in rational.hpp. Define all of those functions in rational.cpp.

# Calculator requirements

The function rc.cpp represents a small rational number calculator and so contains a (mostly) functional calculator for rational numbers. After you have defined the Rational class, this should be able to take any two rational numbers and an operator and compute the result. Sample input might be:

**> 1/2 == 2/4**

**true**

**> 1/2 \* 1/4**

**1/8**

**> 1/4 < 1/2**

**true**

You need to finish the calculator by invoking all of the various operators you overloaded for the Rational class. See the comments in the rc.cpp function.

# Going the extra mile

A good Rational number class should interoperate with ints. For example, we expect the following to work:

**Rational r1 = 1;**

**cout << (r1 == 1); // prints true**

**cout << (2 == r1); // prints false**

**cout << (r1 < 2); // prints true**

**cout << (0 < r1); // prints false**

**cout << (r1 + 2); // prints 3/1 (or just 3)**

To do this, you must provide overloads of all operators that take an int as one of the arguments, and a Rational as the other. This will increase the size of your program considerable.

As you do this, consider how this will affect the streaming operators (<< and >>). Whenever you print a rational who’s a denominator of 1, you should just print that value. And when you read a value, you should be able to differentiate between a rational number and just an integer.

Note that correctly extending the >> operator to accept integers as rational numbers will allow the calculator to accept input like this:

**> 3 + 4**

**7**

**> 1/2 + 2**

**3/2**

# Program and Submission Details

## Grading basis

If your homework is not in Springboard including all files, you will get a 0 on your assignment. You must submit all files to receive a grade. Do not submit the cdp file.

|  |
| --- |
| The total is out of 100 points.  • 0 You submitted code without any effort  • 40 You submitted complete code but you put minimal effort into doing the work  • 50 You submitted complete/solid code that does not compile  • 75 You submitted complete/solid code that compiles, but crashes or has errors when executed  • 90 You submitted a program that compiles and runs with no errors  • 100 You’ve gone the extra mile (see above), and your source code is well documented |

1. We will be testing for different starting numbers of rats and other parameters so you should rigorously test your code and manually check your results to see if they are correct.
2. Complete code means all required ‘to do’ coding was done, solid means it was done coherently.
3. Name your variables appropriately. Don’t use more variables than you really need. *Do* use named constants for the percentages.
4. ***This is not a group project***. If you are having difficulty consult with your professor, the teaching assistant, or the tutor only.
5. Refer to the Programming Rubric for details located on Springboard: Table of Contents -> main() -> { Course Orientation ->RubricProgrammingforAssigments
   1. Use a header comment with your name and the description of the program.
   2. Use pre and post condition comments in each member function.
   3. Validate all data.
   4. Include all your code in the appropriate files for the program.
   5. Use appropriate code comments, member function and Class names, variable names, constants...
   6. Do not use the STL or other resources (unless prescribed for the problem).
   7. Use good coding practices (i.e., spacing, indentation, etc.).

Submission Instructions – for projects

On Springboard, go to the matching Assignments for the **Project #**, where # is the appropriate number of the project that is assigned (eg., Project 1), and submit the programs (cpp) and implementation files (hpp) to Springboard and upload. Make sure it is the program only and **not** a project (cbp) file. Make sure all of the *separate compilation files* required for this project are included or you will not receive any points. You may use any suitable name for your cpp/hpp files of your choice.

**Projects will not be graded after 11:59 p.m. on the due date.**

*Last Updated 12.27.2016. Based on original material by Andrew Sutton.*

*Be aware that programming falls under all of the rules of plagiarism. Be careful when using any coding found in the outside world that is not your own. Any evidence of plagiarism is subject to sanctions like forfeits, suspension, and even ejection, as determined by the Department of Student Conduct and Community Standards.*