3460:210

Programming Project: Rational

# Project: Rational numbers

In this assignment, you will create a Rational number class and implement all the required functionality for rational number operations.

# Getting Started

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

Review the contents of the project folder. You should have the following files:

* rational.hpp and rational.cpp: These files will contain the definition and implementation of your Rational class. Most of your work will go into these files.
* test\_rational.cpp: A test suite provided to check your Rational class. You won’t need to modify this, but you should understand how it tests your class. This is the driver for your program.

The provided files are downloadable, and you can rename them with the appropriate extensions. The test suite will not work initially since you need to implement the required functions.

# 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.

# Project requirements

The test\_rational.cpp represents a small set of rational number test code. 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**

# 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 it 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 Brightspace including all files, you will get a 0 on your assignment. You must submit all files to receive a grade.

|  |
| --- |
| 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.
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 Brightspace: 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

**Team Collaboration Option:**

**Students have the option to work on this project in pairs. If you choose this option, you and your partner will collaborate to submit one project representing the efforts of both. Ensure both team members understand and contribute to all parts of the project.**

**Submission Note: In your submission, it is essential to clearly indicate both team members' names and specify who did what. Detail who implemented which parts, wrote specific sections of code, or performed any other notable tasks. This clarity ensures each student receives proper credit for their contributions.**

**Remember, collaboration is about mutual understanding and effort; make sure both participants are actively involved in all stages of the project. If you prefer to work alone, that is perfectly acceptable as well.**

On Brightspace, 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 Brightspace 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 08.10.2024 by Karima Elgarroussi*

*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.*