CSE2050 - Programming in a Second Language
Assignment 1: Rational Numbers

September 4, 2017

1 Submission

- Due Date: **September 18th, 11.59pm**
- Deliverable: **rational.cpp** (case sensitive)
- Submit on **Canvas**.

2 Description

The goal of this assignment is to implement a `C++` program that can perform basic mathematical operations on rational numbers (represented as fractions in the form $\frac{a}{b}; b \neq 0$) using a number of functions (described below). These operations include:

- Adding two fractions.
- Subtracting one fraction from another.
- Multiplying two fractions.
- Dividing one fraction by another.
- reducing a fraction to its simplest form (i.e., when the *greatest common divisor* (GCD) of its numerator and denominator is 1).

3 Important Notes

Here are some points that we consider when grading your submission..

- Submit only your source code (rational.cpp). No other files are needed (or graded!).

- Make sure your code runs on our server (**code01.fit.edu**). Your program will be compiled using the following flags:

```
g++ -Wall -g -std=c++11 -o rational rational.cpp
```

- To help the automated grading system to correctly read the results of your code, please make sure you follow the format shown in the examples below (including extra spaces and/or extra characters in the output will cause your results to be flagged as incorrect, which will then have to be re-graded manually). Please do not print out any prompts (like "Enter a number",.. etc) or other comments (like "the result is",..etc).

- Please make sure your program handles the input correctly. The problem description for each assignment provides details about the method to use (e.g., from a file, or from the command-line,..etc) and the format of that input. If, for example, the input is to be provided from the command-line (as in this project), but your program can only read input from standard input, then it will fail our tests (will affect your grade).

4 Implementation details

4.1 Input

As shown in the examples below, the program receives 5 integers from the command line as follows:

```
./rational n1 d1 n2 d2 op_code
```

The first two integers (n1 and d1) represent the first fraction ($(\frac{n1}{d1}$). The second pair (n2 and d2) represent the second fraction ($(\frac{n2}{d2}$). The last integer (*op_code*) determines the operation to be performed on those two fractions. *op_code* can have one of the following values:

op_code	operation
1	Addition $((\frac{n1}{d1} + \frac{n2}{d2}))$
2	Subtraction $((\frac{n1}{d1} - \frac{n2}{d2}))$
3	Multiplication $((\frac{n1}{d1} \times \frac{n2}{d2}))$
4	Division $((\frac{n1}{d1} \div \frac{n2}{d2}))$

For details on how to read and handle command line arguments, you can refer to cplusplus.com

4.2 Output

The program prints out a fraction (in the lowest terms) that represents the result of the requested operation, followed by the new line character (as shown in the examples below).

Please note that there are no spaces preceding or contained within this fraction. To pass the automated test cases, your output should match the examples character-by-character (i.e., don't print other things, just the fraction).

4.3 Suggested functions

The functions you need to implement this assignment might include the following:

1. ```
   int GCD(int a, int b)
   ```

   This function calculates the **gcd** of **a** and **b** using Euclid's algorithm (recursively). Other (non recursive) methods to calculate the **gcd** might not receive full mark (one of the goals of this assignment is to practice recursion).

2. ```
   int LCM(int a, int b)
   ```

   Calculates the least common multiple (LCM) of the two numbers a and b.

3. ```
   void simplify(int n, int d)
   ```

   Reduces the numerator and the denominator (n and d) to the lowest possible values using their **gcd**. But, can we change the values of function parameters?

4. ```
   bool is_in_lowest_terms(int n, int d)
   ```

   Returns **true** if the fraction $\frac{n}{d}$ is already in the lowest terms, returns **false** otherwise.

5. ```
   string rational_to_string(int n, int d)
   ```

   Returns a string representation of the fraction in the form n/d. Only add the "/" and the denominator if the latter is not 1. The return value does not contain any spaces or other characters.

6. ```
   string rational_addition(int n1, int d1, int n2, int d2)
   ```

   Adds two fractions ($\frac{n1}{d1} + \frac{n2}{d2}$) (you can call the LCM() function to calculate the LCM of the denominators d1 and d2). Returns the result as a string in the required format (can we return the result as two separate integers instead?).

7. ```
   string rational_subtraction(int n1, int d1, int n2, int d2)
   ```
   ```

Subtracts two fractions ($\frac{n1}{d1} - \frac{n2}{d2}$) and returns the result as a string. Can we utilize the rational_addition() function to calculate this?

8.
```
string rational_multiplication(int n1, int d1, int n2, int d2)
```

Multiplies two fractions ($\frac{n1}{d1} \times \frac{n2}{d2}$). Returns the result as a string.

9.
```
string rational_division(int n1, int d1, int n2, int d2)
```

Divides two fractions ($\frac{n1}{d1} \div \frac{n2}{d2}$). Returns the result as a string.

10.
```
int main()
```

Reads and parses command-line arguments, calls the appropriate method to perform the requested operation, and prints the result on the standard output device (cout). The program terminates after printing this output.

5 Sample input/output

Each of the following examples shows the command used to execute the program (with the input values) on the first line, and the result printed by the program for that input on the second line.

```
./rational 3 4 1 2 2
1/4
```

```
./rational 2 5 1 10 1
1/2
```

```
./rational 2 5 1 10 3
1/25
```

```
./rational 2 5  10 20 2
-1/10
```

```
./rational 40 100 5 2 3
1
```

```
./rational -3 5 1 9 3
-1/15
```

```
./rational 4 1 2 2 4
4
```

```
./rational 3 6 -2 5 1
1/10
```

6 Rubric

Criterion	Possible points	Excellent (max. points)	Satisfactory (partial points)	Unsatisfactory (no points)
Delivery	10%	on time, using correct file name (see top)		wrong file name
Compiles	10%	compiles on code01.fit.edu with no errors		does not compile
Runs	10%	runs on code01.fit.edu with no run-time errors (missing files,..etc)		does not execute
Completion	25%	all functions implemented		
Calculating GCD	10%	implemented Euclid's algorithm (recursively)	implemented another non-recursive algorithm	
Output format	10%	correct output in the correct format	correct output in the wrong format (not simplified, contains extra characters, ...etc)	wrong output
Test cases	25%	passes all test cases	passes some test cases	fails all test cases