Rational Lab Part 2

1. Introduction

This lab practices defining and enhancing class rational with operator overloading to replace the member functions for the arithmetic operations, adding relationl operations for ==, < and >, splitting the declaration and definition into multiple files. This material is covered in chapter 13 of Big C++. Friend functions are covered in a power point on Blackboard.

2. Multiple File Requirement

When we build classes, we build them because they are important and we expect to use them in multiple programs. Keeping classes in the same file as the program is a hinderance from doing that. Instead, classes should be put into two separate files: a ***header*** file with the class declaration and a ***source*** file with the class implementation. The header file ends in .h (or .hpp) and the source file ends in .cpp.

We will practice doing this with putting labrational.cpp into three files:

1. `rational.h`: header file that declares the class rational. Cut and paste the member declarations of the class rational into this file.
2. `main.cpp`: driver program that tests the class rational, this file contains your main function.
3. `rational.cpp:` source file is the implementation file for class rational. Cut and paste the implementation of ALL member functions of class rational into this file

The easiest way to create the necessary files above is with the following linux commands from within the rational_part2b directory:

```
cp ../rational_part1/Labrational.cpp rational.cpp
cp ../rational_part1/Labrational.cpp rational.h
cp ../rational_part1/Labrational.cpp main.cpp
```

Modify your new files as described above. Then make further modifications listed below.
In addition to the usual header files you include, add the following to both .cpp files:

```
#include "rational.h"
```

3. Compilation

To compile your code with all sources:

- You can use a single command to compile all .cpp files, and link them together into a.out, as follows:

```
g++ rational.cpp main.cpp
```

- Or you can compile each .cpp file separately, and then link them together:

```
g++ -c rational.cpp
## the above command just compile rational.cpp to object
```

code,

```
## rational.o

 g++ -c main.cpp
## the above command just compile main.o to object code,
## main.o

 g++ rational.o main.o
## link *.o into one executable a.out
```

4. Additional rational class Requirements

- Use default values to merge the three constructors into one. By using default values for parameters (recall the example we have seen, including **Money** class below.). Remember that our case having the numerator default to 0 is fine but denominator should NOT default to 0. What should it be to reflect whole numbers?

```
class Money
{
public:
  // Constructor definition
  Money(long dollars=0, long cents=0)
  {
    if (dollars*100+cents < 0) {
        cerr << "Error: Invalid parameters" << endl;
        exit(1);
    }
    all_cents = dollars*100+cents;
  }

 // more functions omitted

private:
    long all_cents;
};
```

1. For all parameters that are of **`rational`** type, use pass-by-reference (to avoid copying the whole object when passing parameters). If the function is NOT supposed to modify the parameter, USE **`const`** before the parameter type and **`&`** after.
2. Modify the arithmetic member function declarations **Add, Subtract, Multiply, Divide** to the corresponding friend operator function. The operators return a rational result rather than setting the invoking object. (Those would be the following: +=, -=, *=, /= set the invoking object).

   ```
    // TODO: Modify the Add member to be operator+ that returns the sum of
   the op1 + op2.
   friend rational operator +(const rational& op1, const rational& op2);
   ```

3. Modify the implementation of the arithmetic member function definitions for **Add, Subtract, Multiply, Divide** to the corresponding operator function in rational.cpp. The operators return a rational result rather than to set the invoking object. Notice that it is not a member function because of the `friend` keyword

```
 // TODO: Modify the Add member to be operator+ that returns the sum of
the op1 + op2.
rational operator +(const rational& op1, const rational& op2)
{
   rational temp;
   temp.numerator = (op1.numerator*op2.denominator +
op2.numerator*op1.denominator);
   temp.denominator = (op1.denominator * op2.denominator);
   temp.simplify();
   return temp;
}
```

4. Declare and implement a private function called **`simplify`** that simplifies the invoking object, for example, if the invoking object is stored as 4/12, the function will simplify it to be 1/3. Use the **`gcd`** function defined below.
   Hint: you will need to find the **`gcd`(greatest common divisor)** of the numerator and denominator, and then update them by dividing them by the gcd.

```
//JH: return the gcd of numbers a and b
int gcd (int a, int b)
{
   while (a!=0 && b!=0)
   {
      a = a % b;
      if (a!=0)
         b = b % a;
   }
   if (a==0)
      return b;
   if (b==0)
      return a;
    return 1;
 }
```

   And then, in all the functions that modify the invoking object's value, call **`simplify`** function in the very end to simplify its value before returning from the function.

5. Add overloaded operators for insertion (`operator <<`) and extraction (`operator >>`). Given the lack of time. Remember to use the output ostream parameter instead of cout and return the ostream reference. Do the same for the input `istream` in `operator >>`.
6. Add a `set` function that takes a new_numerator and a new_denominator. It will be unit tested along with the additional constructors.

5. Requirements for main

Modify your main function, so that in addition to the original test code, your program should:

- Modify the prompt for an operator to include the new relational operators.

```
cout << "\nEnter operator [+, -, /, *, =, ?(==), <(less),
>(greater), c(lear), a(ccessors), q(uit)]: ";
```

- Modify the code to get another operand to include '?', '<' and '>'

```
if (oper in "+-*/?<>") cout << "\nEnter op2 (in the format of
p/q):";
```

- Modify the calls to Add, Subtract, Multiply, and Divide to use your new operators function.

```
result = result + op1; // used to be result.add(result,op1);
```

- Add cases for handing '?' (call operator==), '<' (call operator<), '>' (calls operator>).

```
'?':  cout << ((result == op1)? "Correct! Good job!" : "Oh no!
Good try!") << endl;
'<':  cout << ((result < op1)? "True, great!" : "False, try
again!") << endl
```

- Modify the input and output function calls to use operator>> and operator<< respectively. The input operator >> should validate the input and set the failbit if it fails. **(DO THIS LAST!!!)**

```
istream& operator >> (istream& in, rational& r);
ostream& operator << (ostream& out, const rational& r);
```

In the main function, you will need the following code at the bottom of the loop:

```
// Fixes cin if he last rational was invali
if (cin.fail()) {
     cin.clear(); cin.ignore();
}
```

Sample rational.h: (what's needed?)

```
/*  This rational class practice basics of defining and using class.
Start with the rational class from part1

 *  Make the required changes and leave the rest.
 */
#include <iostream>

#ifndef _RATIONAL_H
#define _RATIONAL_H
using namespace std;
class rational {

    // TODO: Modify constructor declaration to set default values for
parameters. See instructions below.
```

```
    // TODO: Overload the >> operator so it can be used to input
values of type rational

    // Precondition: If in is a file input stream, then in has already
been connected to a file

    // TODO: Overloads the << operator so it can be used to output
values of type rational

    // Precondition: If out is a file output stream, then out has
already been connected to a file

    // TODO: Modify the add member function to overload operator+
    // op1 + op2 is stored in a local rational object which is
returned
    // @return a rational with the sum




    // TODO: Modify the subtract member function to overload operator-

    // op1 - op2 is stored in a local rational object which is
returned

    // @return a rational with the difference



    // TODO: Modify the multiply member function to overload operator*
    // op1 * op2 is stored in a local rational object which is
returned
    // @return a rational with the product


    // TODO: Modify the divide member function to overload operator/
    // op1 / op2 is stored in a local rational object which is
returned
    // @return a rational with the quotient


    // TODO: Declare operator== where op1 and op2 are compared to see
if they are equal
    // @return true if equal (compare cross product of n1*d2 == n2*d1)
or false if not.
```

```
    // TODO: Declare operator< where op1 and op2 are compared to see
if op1 < op2.
    // @return true if op1 < op2 (compare cross product of n1*d2 <
n2*d1) or false if not.


    // TODO: Declare operator> where op1 and op2 are compared to see
if op1 > op2.
    // @return true if op1 > op2 (compare cross product of n1*d2 >
n2*d1) or false if not.


 private:

    // TODO: Declare simplify() member that sets the invoking object
to be a simplified rational


    // TODO: Implement the gcd function to return the greatest common
denominator, use it in simplify above.

    int gcd(int a, int b);

};

#endif // _RATIONAL_H
```

6. Sample Test Cases

- Edit All Arithmetic

```
Enter op1 (in the format of p/q): 2/5
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: +
Enter op2 (in the format of p/q): 1/3
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: =
result = 11/15
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: -
Enter op2 (in the format of p/q): 1/2
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: a
result's numerator is: 7
result's denominator is: 30
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: /
Enter op2 (in the format of p/q): 1/3
```

```
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: =
result = 7/10
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: c
Enter op1 (in the format of p/q): 1/6
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: *
Enter op2 (in the format of p/q): 3/1
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: =
result = 1/2
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: q
```

- Edit Relational Tests

```
Enter op1 (in the format of p/q): 1/5
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: ?
Enter op2 (in the format of p/q): 2/10
Correct! Good job!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: +
Enter op2 (in the format of p/q): 1/3
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: =
result = 8/15
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: <
Enter op2 (in the format of p/q): 7/15
False, try again!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: >
Enter op2 (in the format of p/q): 7/15
True, great!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: <
Enter op2 (in the format of p/q): 3/5
True, great!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: '
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: q
```

- Invalid Odd Output

```
Enter op1 (in the format of p/q): -/8
Invalid rational format!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: +
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]:
Enter op2 (in the format of p/q): 1/-8
Invalid rational format!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: -
Enter op2 (in the format of p/q): -8
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: *
Enter op2 (in the format of p/q): 1/8
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: -
Enter op2 (in the format of p/q): 1-/8
Invalid rational format!

Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: =
result = 7/8
Enter operator [+, -, /, *, =, ?(==), <(less), >(greater), c(lear),
a(ccessors), q(uit)]: q
```