Advanced Programming

Homework Assignment 3

Operator Overloading

General guidelines:

a. Maximize readability and ensure indentation.
b. Do exactly as required in the questions.
c. Every class in every question must be submitted in two seperate files – a header file (.h) and an implementation file (.cpp).
d. Add functions and helper methods as necessary to ensure readability.
e. Submission must be done according to the submission guidelines – a document published in the course website – read them carefully!
f. Use relevant names.
g. Use comments to document your functions and complex code parts. **Add an output example at the end of your file!**
h. Individual work and submission – paired submission is not allowed.

Important note: Unless otherwise specified, ever homework has no more than one week for submission.
Open submission boxes past the deadline are not permission for late submission.

Question 1:

Note: From this exercise onward it is required to use const wherever relevant!!

Part A:

Define and implement a class called Rational that represents rational numbers. (You may base your solution on your code from homework 1. The purple marked instructions are identical to homework 1; all the rest is new).

Note: The denominator must not be 0 or negative.

The class must include the following fields:

- numerator
- denominator

The class must also include the following methods:

- Constructors:
  - An empty constructor that initializes both fields to 1.
  - An assignment constructor. If the denominator argument is 0, the value 1 should be put instead.
  - A copy constructor.
- For every field:
  - A setter.
    The denominator cannot be 0, so if the argument given is 0 set the denominator to 1. The denominator cannot be negative, so if the argument given is negative, reset the numerator accordingly: If it was positive make it negative and if it was negative make it positive.
  - A getter.
- A method print for printing the rational number. The printing is to be in the format: numerator/denominator (For example: 1/2, 3/4, 54/56 etc.) according to

the original values and not the values after reduction. If the denominator equals 1 print only the numerator.

- A **helper** method for reducing the fraction:
  - Function header: void reduce();
  - The function is to reduce the calling object. This means, for example, that given the following code:

```
Rational rat(2,4);
rat.reduce();
```

The resulting numerator for rat will be 1 and its denominator will be 2. After reduction, if the numerator is 0, the method is to make the denominator 1.

Part B:

So as to make the use of the Rational numbers as intuitive as possible, add to the Rational class the operators:
+, -, *, /, ++ (prefix and postfix), -- (prefix and postfix), ==, !=, >=, <=, >, <

Expected behaviour:

- The + operator will be similar to the add method from homework 1. The operator must reduce the result.
  For example, after the following code:

```
Rational r1(1,6);
Rational r2(1,3);
Rational r3;
r3 = r1 + r2;
```

  The values will be:
  r1 will equal 1/6
  r2 will equal 1/3
  r3 will equal 1/2
- The behavior of the other binary arithmetic operators (-, *, /) will be similar to the behavior of the + operator.
- You may assume no division (/) by zero. It is not required to check validity.
- The unary operators (++, --) will not reduce the result.
  For example, after the following code:

```
Rational r1(2,6);
Rational r2
r2 = r1++;
```

  The values will be:
  r1 will equal 8/6
  r2 will equal 2/6
- The equality operator (==) will be similar to the equals method from homework 1, except that now it returns true if their values are actually equal.
  For example, after the following code:

```
Rational r1(2,6);
Rational r2(1,3);
```

  The value of r1 == r2 will be true.

Note: The equality operator (==) requires two calls of reduce – one for each operand. Since the operator is supposed to be const this is problematic. A simple way to bypass the problem is to copy the operands into a temporary variables and apply reduce to them instead.

- Similarly, the inequality operator (!=) will return true if their values are actually inequal.
  For example, after the following code:

  ```
  Rational r1(2,6);
  Rational r2(1,3);
  ```

  The value of r1 != r2 will be false.

  Note: So as to eliminate duplicate code make use of a call to the equality operator.

Use the provided main function to test your class. The function inputs two Rational numbers in the format numerator/denominator with the prompt "enter two rational numbers:" and computes and prints the results of the required operations.

Question 2:

Define and implement a class called MyDate to represent dates.

The class must include the following fields:

- day (int)
- month (int)
- year (int)

The class must also include at least the following methods:

- Constructors:
  - An assignment constructor. If not all arguments are receives the date should default to 1/1/1582. The constructor should check the validity of the date according to the rules described here. The following table lists the months and the number of days there can be in each month:

Number	Name	No. Days
1	January	31
2	February	28 (29)
3	March	31
4	April	30
5	May	31
6	June	30
7	July	31
8	August	31
9	September	30
10	October	31
11	November	30
12	December	31

The number of days in February in a leap year is in parenthesis.
A year is a leap year if it divides by 4 but not by 100, or if it divides by 400.

The program input is in the format dd/mm/yyyy where dd, mm and yyyy are positive integers that represent the day, the month and the year respectively.
The allowed ranges are:
Days: 1 ≤ dd ≤ 31
Months: 1 ≤ mm ≤ 12
Years: 1582 ≤ yyyy ≤ 9999
The validity check must not only assert that the date is in a valid range but that the date itself is valid.
Example: 31/4/2000 is not valid because April has 30 days.
If an invalid date is received the program is to print "Error date" and set the date to 1/1/1582 (day 1, month 1, year 1582).

- A copy constructor.

- A method called setDate for updating the date. The method receives an argument for each field. If the combined arguments constitute a valid date the date is set according to them, otherwise no change is done **in any field** and no output is printed.
- A method called print that prints the date in the format dd/mm/yyyy, where the day and the month can be either one digit or two and the year is 4 digits.
- Copy assignment (operator=). Performs a shallow copy of the fields.
- **Prefix** increment (operator++). Changes the values of the fields to the date of the next day. Make sure to set correctly the day after the end of a month to the beginning of the next month and the day after the end of a year to the beginning of the next year.
- **Postfix** increment (operator++). Changes the values of the fields to the date of the next day. Make sure to set correctly the day after the end of a month to the beginning of the next month and the day after the end of a year to the beginning of the next year.
- operator>. Returns true if the left date is later than the right date. Otherwise returns false.
- operator<. Returns true if the left date is earlier than the right date. Otherwise returns false.
- operator==.

  The expected output for boolean values is "true" or "false" (and not their numeric value.

Implement a main program that does the following operations:

- Prints "Enter a date" and inputs a date (**note the expected input format**).
- Initializes an instance of MyDate with the inputted values.
- In a loop:
  - Prints "Enter a code" and inputs an action code.
  - Performs the correct action (see description below).
  - Prints the value of the object.
- The program ends when code 0 (zero) is inputted.

Note: So that the results of ++ will be printed it is required to assign them into another variable and then print the new variable's value.
For example (prefix increment):

```
MyDate d1, d2;
d1 = ++d2;
d1.print();
cout << endl;
d2.print();
cout << endl;
```

Action codes:

- 1 - Update date: Prints Enter a date, inputs a new date, calls setDate, prints the updated date.
- 2 - Print the result of prefix increment (see example below).
- 3 - Print the result of postfix increment (see example below).
- 4 - Print Enter a date, input a date into a new object, assign the new object into the old object using the assignment operator (=). Print the old object (its new values).
- 5 - Print Enter a date, input a date into a new object. Print the symbol > between the old and the new dates (old on the left), print a colon (:), print the result (true or false).
- 6 - Print Enter a date, input a date into a new object. Print the symbol < between the old and the new dates (old on the left), print a colon (:), print the result (true or false).
- 7 - Print Enter a date, input a date into a new object. Print the symbol == between the old and the new dates (old on the left), print a colon (:), print the result (true or false).
- 0 - End the loop and the program.

Output example (don't print the comments!):

```
Enter a date
-5/1/2012//input
Error date
1/1/2012//prints the assigned value
Enter a code
1
Enter a date
5/7/2010//input
5/7/2010//prints the date after the change
Enter a code
2
6/7/2010//prints prefix increment
6/7/2010
Enter a code
3
6/7/2010// prints postfix increment
7/7/2010
Enter a code
4
Enter a date
14/7/2010
14/7/2010// input through assignment
Enter a code
5
Enter a date
14/7/2010
14/7/2010 > 14/7/2010 : false
Enter a code
7
Enter a date
14/7/2010
14/7/2010 == 14/7/2010 : true
Enter a code
0
```

Good Luck!