San José State University
Department of Computer Engineering

CMPE 180-92
Data Structures and Algorithms in C++

Fall 2016
Instructor: Ron Mak

Assignment #6B

Assigned: Friday, September 30
Due: Friday, October 7 at 11:59 PM
URL: http://codecheck.it/codecheck/files/1610020846371n7r467ipl8kuqwhhhzur2q
Canvas: Assignment 6.b. Roman Numerals
Points: 100

Roman numerals

In this assignment, you will practice creating a class that has friend functions, operator overloading, and separate compilation.

For a refresher on Roman numerals, see https://en.wikipedia.org/wiki/Roman_numerals
Read up to but not including the section **Alternate forms**.

Class `RomanNumeral`

Define a C++ class **`RomanNumeral`** that implements arithmetic operations on Roman numerals, and reading and writing Roman numerals. This class must have:

- Private member variables **`string roman`** and **`int decimal`** store the Roman numeral string (such as **`"MCMLXVIII"`**) and its integer value (such as 1968).
- Private member functions **`toRoman`** and **`toDecimal`** convert between the string and integer values of a **`RomanNumeral`** object and set **`roman`** and **`decimal`**.
- One constructor has an integer parameter, and another has a string parameter.
  - Construct a Roman numeral object either from an integer or a string.
- Public getter functions to return the object's string and integer values.
- Override the arithmetic operators **`+ - * /`** and **`/`**
  - Roman numerals do integer division.
- Override the equality operators **`==`** and **`!=`**
- Override the stream operators **`>>`** and **`<<`**
  - Input a Roman numeral value as a string, such as **`MCMLXVIII`**
  - Output a Roman numeral value in the form **[***integer value***:***roman string***]** such as **`[1968:MCMLXVIII]`**

You may assume that the values of the Roman numerals range from 1 through 3999. (Did the ancient Romans have a zero?)

Separate compilation

In CodeCheck, you will complete source files **RomanNumeral.h** and **RomanNumeral.cpp**.

Test the class

Source file **RomanNumeralTests.cpp** will be provided.

Function **`test1`** performs arithmetic and equality tests on **`RomanNumeral`** objects.

Function **`test2`** inputs the text file **RomanNumeral.txt**:
http://www.cs.sjsu.edu/~mak/CMPE180-92/assignments/6B/RomanNumeral.txt

```
MCMLXIII + LIII
MMI - XXXIII
LIII * XXXIII
MMI / XXXIII
```

The file contains simple two-operand arithmetic expressions with Roman numerals. The function reads each expression, performs the operation, and prints the expression and its result:

```
[1963:MCMLXIII] + [53:LIII] = [2016:MMXVI]
[2001:MMI] - [33:XXXIII] = [1968:MCMLXVIII]
[53:LIII] * [33:XXXIII] = [1749:MDCCXLIX]
[2001:MMI] / [33:XXXIII] = [60:LX]
```

You may assume that all the Roman numerals in the input are in upper case, and that there are no errors. Therefore, for this assignment, you do not need to do error checking of the input.

Rubrics

Criteria	Maximum points

Correct program output (as determined by CodeCheck)
• Correct output from test1:
  ○ **`r1`**, **`r2`**, **`r3`**, and **`r4`**
  ○ **`r1 + r2/r3`**
  ○ **`r2 == r4`**
  ○ **`r1 == r3`**
• Correct output from test2
  ○ **`+`** expression
  ○ **`-`** expression
  ○ **`*`** expression
  ○ **`/`** expression
	40
	• test1:
	  ○ 5
	  ○ 5
	  ○ 5
	  ○ 5
	• test2:
	  ○ 5
	  ○ 5
	  ○ 5
	  ○ 5

Good class design • Constructor with string parameter. • Constructor with integer parameter. • Private member function **`toRoman`**. • Private member function **`toDecimal`**. • Overloaded **`+`** operator. • Overloaded **`-`** operator. • Overloaded **`*`** operator. • Overloaded **`/`** operator. • Overloaded **`==`** operator. • Overloaded **`!=`** operator. • Overloaded **`>>`** operator. • Overloaded **`<<`** operator.	**50** • Constructor string parm: 3 • Constructor integer parm: 3 • **`toRoman`**: 10 • **`toDecimal`**: 10 • **`+`** operator: 3 • **`-`** operator: 3 • **`*`** operator: 3 • **`/`** operator: 3 • **`==`** operator: 3 • **`!=`** operator: 3 • **`>>`** operator: 3 • **`<<`** operator: 3
Good program style • Consistent formatting: indentations, placement of { and }, etc. • Good comments.	**10** • Formatting: 5 • Comments: 5

You can submit as many times as necessary to get satisfactory results, and the number of submissions will not affect your score. When you're done with your program, click the "Download" link at the very bottom of the Report screen to download the signed zip file of your solution.

Submit the signed zip file into **Canvas: Assignment 6.b. Roman Numerals**.