San José State University
Department of Computer Engineering

CMPE 180A
Data Structures and Algorithms in C++

Spring 2018
Instructor: Ron Mak

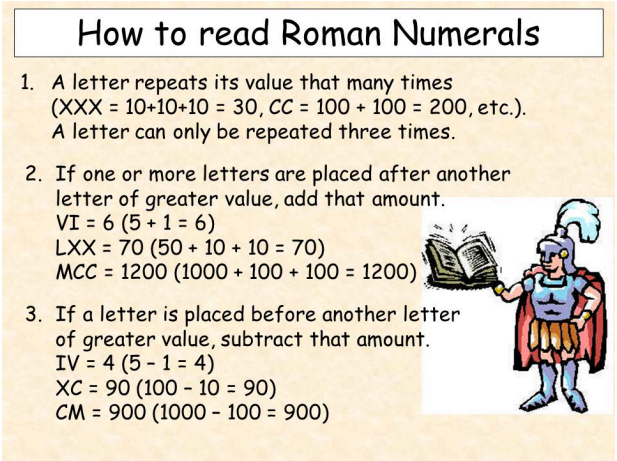
Assignment #5

Assigned: Friday, February 23
Due: Thursday, March 1 at 5:30 PM
URL: http://codecheck.it/files/17091917124r0yck5moj6w8j7rckda27s98
Canvas: Assignment 5. Roman Numerals
Points: 100

Roman numerals

You will practice creating a class that has operator overloading and friend functions, and that uses 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`

Design and implement a C++ class **`RomanNumeral`** that reads, writes, and performs arithmetic operations on Roman numerals. This class must have:

- Private member variables **`string roman`** and **`int decimal`** that store the Roman numeral string (such as **`"MCMLXVIII"`**) and its integer value (such as 1968).
- Private member functions **`to_roman`** and **`to_decimal`** that convert between the string and integer values of a **`RomanNumeral`** object and thereby set the values of member variables **`roman`** and **`decimal`**.
- Public constructors, one that takes an integer parameter and another that takes a string parameter.
  - One constructor creates a **`RomanNumeral`** object from an integer value.
  - The other creates a **`RomanNumeral`** object from a string value.
- Public getter functions that return an object's string and integer values.

- Overloaded arithmetic operators **+ - * /** and that enable direct arithmetic operations with Roman numerals.
  - Roman numerals do integer division (drop the fraction part of a quotient).
- Overloaded equality operators **==** and **!=** that enable direct comparisons of **`RomanNumeral`** objects.
- Overloaded I/O 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 for this assignment that the values of the Roman numerals range from 1 through 3999. (Did the ancient Romans have a zero or negative numbers?)

Separate compilation

In CodeCheck, you will complete header file **RomanNumeral.h** and implementation file **RomanNumeral.cpp**.

Test the class

You are provided source file **RomanNumeralTests.cpp** that contains two tests.

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

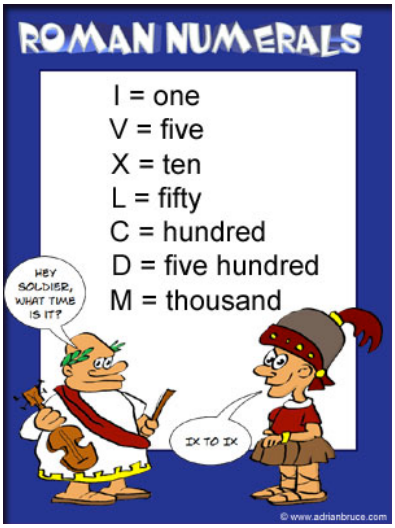
```
MCMLXIII + LIII
MMI - XXXIII
LIII * XXXIII
MMI / XXXIII
```

Function **`test2`** inputs the text file **RomanNumeral.txt**:
http://www.cs.sjsu.edu/~mak/CMPE180A/assignments/5/RomanNumeral.txt

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 input errors. Therefore, for this assignment, you do not need to do input error checking.



Submission into Canvas

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 #5. Roman Numerals.

Rubric

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 `to_roman`. • Private member function `to_decimal`. • Overloaded `+` operator. • Overloaded `-` operator. • Overloaded `*` operator. • Overloaded `/` operator. • Overloaded `==` operator. • Overloaded `!=` operator. • Overloaded `>>` operator. • Overloaded `<<` operator.	**50** • 3 • 3 • 10 • 10 • 3 • 3 • 3 • 3 • 3 • 3 • 3 • 3
Good program style • Consistent formatting: indentations, placement of { and }, etc. • Good comments.	**10** • 5 • 5

Academic integrity

You may study together and discuss the assignments, but what you turn in must be your individual work. Assignment submissions will be checked for plagiarism using Moss (http://theory.stanford.edu/~aiken/moss/). **Copying another student's program or sharing your program is a violation of academic integrity.** Moss is not fooled by renaming variables, reformatting source code, or re-ordering functions.

Violators of academic integrity will suffer severe sanctions, including academic probation. Students who are on academic probation are not eligible for work as instructional assistants in the university or for internships at local companies.