CS 144
Advanced C++ Programming
Spring 2019
Instructor: Ron Mak

Assignment #5

Assigned: Tuesday, February 26
Due: Tuesday, March 5 at 8:30 AM
URL: http://codecheck.it/files/19022508394veybnq2tg6gvrtwuwqge3xs8
**Canvas:** Assignment 5. Roman Numerals
Points: 160

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".

How to read Roman Numerals

1. A letter repeats its value that many times (XXX = 10+10+10 = 30, CC = 100 + 100 = 200, etc.). A letter can only be repeated three times.

2. If one or more letters are placed after another letter of greater value, add that amount.
    VI = 6 (5 + 1 = 6)
    LXX = 70 (50 + 10 + 10 = 70)
    MCC = 1200 (1000 + 100 + 100 = 1200)

3. If a letter is placed before another letter of greater value, subtract that amount.
    IV = 4 (5 – 1 = 4)
    XC = 90 (100 – 10 = 90)
    CM = 900 (1000 – 100 = 900)
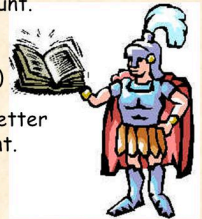
Class `RomanNumeral`

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 the corresponding decimal value (such as 1968) of each `RomanNumeral` object.
- Private member functions `to_roman` and `to_decimal` that convert between the string and decimal 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. Each constructor must ensure that a `RomanNumeral` object's `roman` and `decimal` members are correct.
- Public getter functions that return a `RomanNumeral` object's string and decimal values.
- Overloaded arithmetic operators `+` `-` `*` and `/` that enable direct arithmetic operations with Roman numerals. Each operator must return a `RomanNumeral` object and the operation must not change the value of either operand. Roman numerals do integer division (drop the fraction part of a quotient).
- Overloaded equality operators `==` and `!=` that enable direct comparisons of `RomanNumeral` objects.

1

- 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 **[***decimal***:***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?)

Tip: Do the operations using normal integer arithmetic on the decimal values and then convert to Roman strings.

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

Reference and const
Use reference parameters to minimize argument copying. Use **const** parameters and **const** member functions to maximize compile-time checking.

Test the class
Source file **RomanNumeralTests.cpp** contains two test functions.

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

Function **test2** inputs the text file **RomanNumeral.txt**:

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

http://www.cs.sjsu.edu/~mak/CS144/assignments/5/RomanNumeral.txt

The file contains simple two-operand arithmetic expressions with Roman numerals. The operator is either **+ - *** or **/** and is surrounded on each side by one or more blanks. The function should read each expression, perform the operation, and print the expression and its result:

```
[1963:MCMLXIII] + [56:LVI] = [2019:MMXIX]
[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 <u>not</u> need to do input error checking.
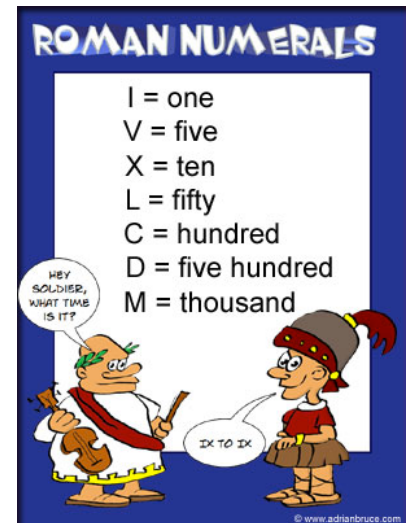
Submission into Canvas

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.

ROMAN NUMERALS

I = one
V = five
X = ten
L = fifty
C = hundred
D = five hundred
M = thousand

Rubric

Criteria	Max points		
Correct program output	**60**		
(as determined by CodeCheck)			
• Correct output from Test1:	• Test1		
○ `r1`, `r2`, `r3`, and `r4`		○	10
○ `r1 + r2/r3`		○	10
○ `r1 == r3`		○	5
○ `r1 != r4`		○	5
○ `r2 == r4`		○	5
○ `r3 != r4`		○	5
• Correct output from Test2:	• Test2		
○ `+` expression		○	5
○ `-` expression		○	5
○ `*` expression		○	5
○ `/` expression		○	5
Good class design	**90**		
• Constructor with string parameter.	• 5		
• Constructor with integer parameter.	• 5		
• Use of reference parameters and `const`.	• 10		
• Private member function `to_roman`.	• 10		
• Private member function `to_decimal`.	• 10		
• Overloaded `+` operator.	• 5		
• Overloaded `-` operator.	• 5		
• Overloaded `*` operator.	• 5		
• Overloaded `/` operator.	• 5		
• Overloaded `==` operator.	• 5		
• Overloaded `!=` operator.	• 5		
• Overloaded `>>` operator.	• 10		
• Overloaded `<<` operator.	• 10		
Good program style	**10**		
• Function declarations with comments before the main.	• 5		
• Good names and internal comments.	• 5		

It is possible ...
... to perform arithmetic directly on Roman numeral strings:
http://turner.faculty.swau.edu/mathematics/materialslibrary/roman/

Doing it on the decimal values is much easier!

Academic integrity

You may study together and discuss the assignments, but what you turn in must be your <u>individual work</u>. 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.