Practical 2: Roman Numbers

Task

How would life be different if the Roman Empire was still alive in the computer age? For a start, computer programs would have to deal with input and output of numbers using a rather different scheme than the Arabic system we use today.

Your task is to write a C++ program, **roman**, that reads numbers specified in Roman form and then writes the numbers translated into Arabic.

The program should read from standard input and write to standard output. Input will consist of a series of lines each containing a number to be converted. For each input line, the program should write a line of output that consists of the converted number terminated with a newline. The program should continuously read input and write output until all input has been processed.

SVN check out

Before you begin, you will need to check out the practical directory from the topic repository. This can be achieved by selecting Get from Version Control on the CLion welcome screen. In the window that appears, select Subversion from the dropdown list. If you have not previously connected CLion to the repository, click the add button (**+**) and paste the following URL into the text field (replacing **FAN** with your FAN):

```
https://topicsvn.flinders.edu.au/svn-repos/COMP2711/FAN
```

If you are prompted to login, use your University FAN and password. Expand the repository listing and select the **roman** directory. Click Check Out then select a location to store your project (preferably in a practicals directory) and click Open. From the destination list, choose the second option to create a project directory named **roman** and then click OK to complete the check out. A dialogue may appear confirming the subversion working format; if so, 1.8 is fine. Finally, when prompted if you would like to open the project, click Yes.

Automated marking

This practical is available for automatic marking via the quiz **Practical 2**, which is located in the Assessment module on FLO. To assess your solution, first commit your changes to the topic repository. This can be completed within CLion via the Commit option under the VCS menu (or alternatively with ⌘+K on macOS or Ctrl+K on Windows).

You should adhere to good development habits and enter a commit message that describes what you have changed since your last commit. When ready, click Commit to upload your changes and receive a revision number. Enter this revision number into the answer box for the relevant question (level).

There are no penalties for incorrect solutions, so if you do not pass all test cases, check the report output, modify your solution, commit, and try again. Remember to finish and submit the quiz when you are ready to hand in. You may complete the quiz as many times as you like—your final mark for the practical will be the highest quiz mark achieved. If you do start a new quiz attempt, ensure you reassess any levels you have previously completed.

Background

Like the familiar Arabic scheme, Roman numbers are written in decimal form. A Roman number contains thousands, hundreds, tens, and units 'digits'. However, Roman numbers use strings of letters to represent each digit, with different letters used for the thousands, hundreds, tens, and units. Moreover, although Roman numbers are always written so that the thousands digit comes first and the units digit last, there is no way to write zero; you simply leave the digit out.

		M or m:	one thousand
D or d:	five hundred	C or c:	one hundred
L or l:	fifty	X or x:	ten
V or v:	five	I or i:	one

Roman digits are written using the following letters, with each letter standing for a certain value:

	Thousands	Hundreds	Tens	Units
1	M	C	X	I
2	MM	CC	XX	II
3	MMM	CCC	XXX	III
4		CD	XL	IV
5		D	L	V
6		DC	LX	VI
7		DCC	LXX	VII
8		DCCC	LXXX	VIII
9		CM	XC	IX

To represent digit values other than those above, digit letters are repeated. As a special case, digits 4 and 9 (and their corresponding higher place values) are written by prefixing the '5' and '10' letters with the 'I' letter. The table below lists all valid Roman digits. Note that the highest valid Roman number using this scheme is 3999 or MMMCMXCIX.

Level 1: Units-only numbers

Begin by writing a program that converts a single Roman number in the range I (1) to IX (9) to Arabic form. The program should read a single string from standard input and print the corresponding value to standard output.

> **Hint:** use an array of strings to represent the Roman units digits, organised so that the string index corresponds to the digit value. For example, the array would have the string `"IV"` at index position 4 and the string `"VII"` at index position 7. You can then convert the input string by searching through the array until you find a match. The index where you find the match will be the digit value you need. Note that for reasons explained below, it is easiest to begin at the '9' end of the array and work downwards.

Extend the program so that it works correctly when the input consists of either lower case or upper case Roman letters. The simplest approach is to convert each character in the input word into uppercase before trying to find a match. Run a loop over the characters in the string using the index/subscript operator (`[]`) to access each character and use the `toupper` function (you will need to include the `cctype` header file) to get the corresponding uppercase value.

Level 2: Multiple numbers

Extend the program so that it can deal with single digit numbers of any value. A single digit number is one that consists only of thousands, hundreds, tens, or units. Thus LXX (70) and CD (400) are single digit numbers, but XIV (14) and MC (1100) are not. Use the same approach as for units digits, but with 4 different arrays, one each for the thousands, hundreds, tens, and units digits. Try looking for thousands digits first, then for hundreds, and so on. When you find a match in one of the arrays, print the corresponding value and stop.

Modify the program so that it reads and converts all input numbers until end of file (eof) on standard input. You will probably be able to do this by simply adding an appropriate *reading loop* around the code that reads a single line.

Level 3: Multi-digit numbers

Extend the program so that it can handle multi-digit numbers. The idea is to look for prefixes of the Roman number that correspond to valid Roman digits, beginning with the thousands digits and working from the '9' down to the '1' end. If you find a matching prefix, remove it and add the corresponding value to a progressive total, then move on to the next digit position. Since each digit is unique, and since larger digits are represented by longer strings than smaller ones, this process will always find the correct value. When you have considered all possible digits, the progressive total is the answer you need.

Make sure the program behaves correctly even if the input does not consist of a valid Roman number. In such cases, convert the longest valid prefix and ignore any remaining characters. If no prefix characters are valid, display zero.

Level 4: Convert either way

Extend the program so that if the input is an Arabic number, the output is the corresponding Roman number. Use the first character in the input word to decide which way to convert. If it is an Arabic digit, convert as much as possible of the word to Roman. If it is a Roman digit letter, convert as much as possible to Arabic. If it is neither, output zero. Use the same digit arrays to convert both to and from Roman numbers.