**Roman Numerals**

In Practice Mode

Introduction

Write a function to convert from normal numbers to Roman Numerals.

The Romans were a clever bunch. They conquered most of Europe and ruled it for hundreds of years. They invented concrete and straight roads and even bikinis. One thing they never discovered though was the number zero. This made writing and dating extensive histories of their exploits slightly more challenging, but the system of numbers they came up with is still in use today. For example the BBC uses Roman numerals to date their programmes.

The Romans wrote numbers using letters - I, V, X, L, C, D, M. (notice these letters have lots of straight lines and are hence easy to hack into stone tablets).

1 => I

10 => X

7 => VII

There is no need to be able to convert numbers larger than about 3000. (The Romans themselves didn't tend to go any higher)

Wikipedia says: Modern Roman numerals ... are written by expressing each digit separately starting with the left most digit and skipping any digit with a value of zero.

To see this in practice, consider the example of 1990.

In Roman numerals 1990 is MCMXC:

1000=M 900=CM 90=XC

2008 is written as MMVIII:

2000=MM 8=VIII

See also: http://www.novaroma.org/via\_romana/numbers.html

Getting Started

Make sure you have read the "Guides" section of the [C track](https://exercism.io/my/tracks/c) on the Exercism site. This covers the basic information on setting up the development environment expected by the exercises.

Passing the Tests

Get the first test compiling, linking and passing by following the [three rules of test-driven development](http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd).

The included makefile can be used to create and run the tests using the test task.

make test

Create just the functions you need to satisfy any compiler errors and get the test to fail. Then write just enough code to get the test to pass. Once you've done that, move onto the next test.

As you progress through the tests, take the time to refactor your implementation for readability and expressiveness and then go on to the next test.

Try to use standard C99 facilities in preference to writing your own low-level algorithms or facilities by hand.