**Sieve**

## Introduction

Use the Sieve of Eratosthenes to find all the primes from 2 up to a given number.

The Sieve of Eratosthenes is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e. not prime) the multiples of each prime, starting with the multiples of 2. It does not use any division or remainder operation.

Create your range, starting at two and continuing up to and including the given limit. (i.e. [2, limit])

The algorithm consists of repeating the following over and over:

* take the next available unmarked number in your list (it is prime)
* mark all the multiples of that number (they are not prime)

Repeat until you have processed each number in your range.

When the algorithm terminates, all the numbers in the list that have not been marked are prime.

The wikipedia article has a useful graphic that explains the algorithm: https://en.wikipedia.org/wiki/Sieve\_of\_Eratosthenes

Notice that this is a very specific algorithm, and the tests don't check that you've implemented the algorithm, only that you've come up with the correct list of primes. A good first test is to check that you do not use division or remainder operations (div, /, mod or % depending on the language).

## 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.