**List Ops**

In Practice Mode

Introduction

Implement basic list operations.

In functional languages list operations like length, map, and reduce are very common. Implement a series of basic list operations, without using existing functions.

The precise number and names of the operations to be implemented will be track dependent to avoid conflicts with existing names, but the general operations you will implement include:

* append (*given two lists, add all items in the second list to the end of the first list*);
* concatenate (*given a series of lists, combine all items in all lists into one flattened list*);
* filter (*given a predicate and a list, return the list of all items for which predicate(item) is True*);
* length (*given a list, return the total number of items within it*);
* map (*given a function and a list, return the list of the results of applying function(item) on all items*);
* foldl (*given a function, a list, and initial accumulator, fold (reduce) each item into the accumulator from the left using function(accumulator, item)*);
* foldr (*given a function, a list, and an initial accumulator, fold (reduce) each item into the accumulator from the right using function(item, accumulator)*);
* reverse (*given a list, return a list with all the original items, but in reversed order*);

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.