**Circular Buffer**

In Practice Mode

Introduction

A circular buffer, cyclic buffer or ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end.

A circular buffer first starts empty and of some predefined length. For example, this is a 7-element buffer:

[ ][ ][ ][ ][ ][ ][ ]

Assume that a 1 is written into the middle of the buffer (exact starting location does not matter in a circular buffer):

[ ][ ][ ][1][ ][ ][ ]

Then assume that two more elements are added — 2 & 3 — which get appended after the 1:

[ ][ ][ ][1][2][3][ ]

If two elements are then removed from the buffer, the oldest values inside the buffer are removed. The two elements removed, in this case, are 1 & 2, leaving the buffer with just a 3:

[ ][ ][ ][ ][ ][3][ ]

If the buffer has 7 elements then it is completely full:

[6][7][8][9][3][4][5]

When the buffer is full an error will be raised, alerting the client that further writes are blocked until a slot becomes free.

When the buffer is full, the client can opt to overwrite the oldest data with a forced write. In this case, two more elements — A & B — are added and they overwrite the 3 & 4:

[6][7][8][9][A][B][5]

3 & 4 have been replaced by A & B making 5 now the oldest data in the buffer. Finally, if two elements are removed then what would be returned is 5 & 6 yielding the buffer:

[ ][7][8][9][A][B][ ]

Because there is space available, if the client again uses overwrite to store C & D then the space where 5 & 6 were stored previously will be used not the location of 7 & 8. 7 is still the oldest element and the buffer is once again full.

[D][7][8][9][A][B][C]

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.