Contents

Overview

This is an example that we will work through in class time of building a linked list in C. This example will have us thinking about:

1. Design by contract (preconditions, postconditions, invariants).
2. Build tools (`make`).
3. Automated testing.
4. Debugging (with `lldb`).

Permission to use

The code

`list.h`

This the interface we will be building. This isn't a very good list because we can only insert to front and ask about size, but it's enough for us to review the concepts we want to review.

This interface is designed to hide the implementation of the list from the code using this library (we do not reveal any `struct` in this interface). That means that we could (in theory) build multiple versions of the list (e.g., arrays and linked lists) without having to change code that's relying on this list.

`list.c`

This is an empty file that we will fill in with an implementation of a linked list.

`main.c`

This is a mostly empty file that we will fill in with an implementation of an automated test.

`Makefile`

This is... a `Makefile` (it is what it says on the tin). This `Makefile` uses a few different things that you may have not seen before:

- Implicit Rules

  This is where I've written things like `main: list.o` without specifying how to build either `main` or `list.o`. Different versions of `make` know how to build C programs and `.o` files.

- Variables

  Implicit rules are built up using different system variables. This `Makefile` defines values for variables like `CC` (the C compiler to use).

The `Makefile` also has a set of `CFLAGS` specified that you should use. One of these flags (`-D_FORTIFY_SOURCE=3`) is there to help with things like doing basic bounds checking on arrays. It's not able to catch all programming mistakes, but it's able to help catch many.

Building

You can build this code using `make`:

```
make
```

Running

You can run the automated tests (there are none here right now!) by running:

```
./main
```