CSE 410: Systems Programming

Programming Assignment 0: Pointer Practice — Linked Lists

Introduction

This assignment will give you a little bit of practice with pointers and refresh your memory on how references and linked data structures work. You will also have to write some tests. Later assignments in this class will require many of the skills you will use in this assignment.

You have been given a partial implementation of a linked list library using a C structure for its list node representation. Several of its functions have no implementation. You must complete those implementations and write tests to verify the correctness of your linked list library.

1 Getting Started

You should have received a GitHub Classroom invitation for this assignment. Follow it, then check out your repository.

Look at `src/list.h` and `src/list.c` in the project sources. The header defines the interface for the linked list structure you will be creating, and *you should not change it for this project*. The C source is a partial implementation of the given API that you may use for reference and should incorporate into your implementation. The given functions that have implementations are correct and complete, and need not be modified to receive full credit on this assignment.

You will also want to look at the `Makefile` in the top-level directory, as you will have to modify it for this project. It contains many comments and examples of recipes that you will find useful, either directly or modified to suit your requirements.

2 Requirements

For this project, you must:

1. Complete the linked list implementation as defined in the given `src/list.h`.
   - Your implementation must be *complete* and *correct*.
   - You must manage memory cleanly. There should be no memory leaks, no access of invalid pointers, *etc.*
   - The specific functions you must implement are:
     - `list_append`
     - `list_remove`
     - `list_free`
2. Create tests to validate your implementation.
   - You have been given several tests, but you must create *at least two additional tests* that **fail on the given code** and **pass on a correct and complete implementation** when invoked with no arguments.
   - Each test should be a *single `.c` file* added to the `tests` subdirectory.
   - Each test `.c` file must compile to an executable of the same name as the source file, without the trailing `.c`, just as the included tests do. For example, the included `tests/test_prepend.c` compiles to `tests/test_prepend`.
   - The tests should return `0` from `main` if the test *passes*, and non-zero if the test *fails*.
   - The compiled test executable names (including `tests/`) must be included in the `NEWTESTS` Makefile variable.

- Ensure that your two additional tests are compiled by `make test`.
- Use the given tests and the given Makefile as an example of how to design and compile your new tests. *Note that your tests will almost certainly fail to compile without changes to the Makefile!*

You may write as many additional tests as you wish, but you will not receive direct credit for writing more than two. Any tests that you do not wish the grader to check should *not* be included in the value of `NEWTESTS`.

You must submit your code to Autograder by creating a *tar file* of your completed code. Tar is an archive program (the **t**ape **ar**chiver), and we will use it to bundle several files into one. The Makefile included with the project repository will create an archive file named `submission.tar` when you run the command `make submission`, and this is the file that you should submit to Autograder.

In order for your code to be tested properly, you *must* put your tests in the `tests` subdirectory and include them in the `NEWTESTS` Makefile variable as described above.

3 Grading

Your submission will be awarded points as follows:

- Correctness of the required functions (1 point each, 3 points total):
  - `list_append`
  - `list_remove`
  - `list_free`
- Correctness of memory management (1 point)
- Additional tests (1 point each test, 2 points total)

The testing apparatus will assume that the functions with given implementations *remain correct*. If you break those functions (intentionally or inadvertently), you may find that some tests fail unexpectedly.

Note that not all of these features are tested completely in the given source or in the autograder testing. Your additional tests should cover some of the additional features! This means that **your final score on this project may be lesser than that reported by Autograder**.

No submission will be graded unless it follows the required submission instructions. Autograder should inform you if your submission is invalid due to failure to follow the submission instructions.