Final Project Tips

What are Unit Tests?

In computer programming, unit tests are a way of testing small, individual pieces of your code. This way, you can narrow down issues with your program to a specific section. If you instead write the entire program and then attempt to run it all at once, it is much harder to find which lines of code are causing issues.

https://en.wikipedia.org/wiki/Unit_testing

When working on the final project, you break up your program into many smaller functions, and test them one at a time as you write them.

For example, let's say we have a function called "adder" in another file that is supposed to add two integers. You can write a separate unit test function to make sure that "adder" works correctly. We will call this testing function from main to run the test.

```
void test_adder(void) {
    // Print results of various tests
    print("%d\n", adder(5, 6));
    print("%d\n", adder(0, 5));
    print("%d\n", adder(-1, 5));
    print("%d\n", adder(-1, -5));
}
```

To run the test, simply call `test_adder()` from the main function. Then you would verify that what is printed is what you expect. Notice we tested a variety of test cases with positive, negative, and 0 numbers...not just one type of test.

Document Setup

1. Create your main, .c, and .h files. The assignment description requires these files be named `RootTable.c`, `MyLinkedList.h`, and `MyLinkedList.c`.
2. Write a simple program where the main function in `RootTable.c` calls another function in `MyLinkedList.c` to print a "Hello World" message. This is just to test that we can compile the files correctly
3. Create a makefile, then run the simple test program.

Creating Nodes

1. Define a node struct in your header file. It needs to contain both a number (value), and a pointer to the next node (next). Watch the lecture from class on linked lists for more help with how they work if

needed.

2. **Unit Test:** Write a unit test that creates a new node, then prints the value from the node
3. **Unit Test:** Write a unit test that creates two nodes, then link them together using the "next" property of the first node. Your test should be able to print the value from the second node by accessing the "next" property of the first node.

The Linked List

When working with a linked list, your main function will store the pointer to the head of the list. **Any function in MyLinkedList.c that modifies the list should accept the head pointer as a parameter, and return the pointer to head when finished.** This is because head might change if a node is inserted at the beginning of the list.

Watch the lecture from class on linked lists for more help with how they work if needed.

1. Write a function that accepts an integer input, creates a node and returns a pointer to that node (call this function "add").
2. **Unit Test:** Test the function, and make sure you can print the value from the node.
3. Modify the "add" function so that if you call it several times, it will create a linked list by linking the new node after the previously created node. Remember, your function should also be accepting "head" as a parameter, and returning a pointer to "head".
4. **Unit Test:** Test that the new function can create a linked list by printing the value of each node in the list, starting from the first.
5. Consider writing a "print_list()" function that will print the linked list. You'll be doing this a lot to test the program
6. Modify the "add" function again to insert the new value in the correct sorted location in the list instead of always putting it at the end.
7. **Unit Test:** Test adding nodes with values out of order. Watch out for the special case of adding a node that should end up being placed at the beginning of the list, or a currently empty list.

File I/O

1. Write the function that reads a list of integers from a file, then simply print the values as they are read.
2. **Unit Test:** Write a unit test that passes a file name to your function, and make sure the integers are printed.
3. Modify your function to add the integers to a linked list instead of printing them.
4. **Unit Test:** Modify the previous unit test to print the linked list that should have been created.

5. Write the function that will output the values in the linked list to a file along with the required calculations.

6. **Unit Test:** Write a unit test that will pass the input and output file names to the function you just write. Look at the output file to ensure this works.

Finishing Up

Let's finish up by adding code to the main function!

1. Validate that the file names passed into argv[] exist, then call the appropriate functions you've already wrote to read and write to the files.

2. Make sure you've followed all the program requirements including correct formatting, using malloc and delete, and accepting file names as command line parameters.