CSE2421 Lab 4 (4 points)

Pointers, Strings, and Structs

In this lab, you will need to implement a doubly linked list with pointers and structs, and then implement a command line interface to operate the linked list. I already provide a skeleton of the code.

Step 1: Doubly linked list

I assume you have learned it in your prior class, but if you forget, you can refer to the wiki page (https://en.wikipedia.org/wiki/Doubly_linked_list)

1.1. Each node contains an integer value. You should define node members in struct ListNode (in linked_list.h)

1.2. You should define members in struct LinkedList (in linked_list.h).

You should implement five functions in linked_list.c:

1.3. **init_list** should initialize the linked list. If you find there is nothing to do, you can leave it as blank.

1.4. **destroy_list** releases everything in the linked list. If you find there is nothing to do, you can leave it as blank.

1.5. **print_list** prints the integer value of each node, from the first node to the last node.

1.6. **insert_node** inserts a node at a given position marked by index. If index is 0, it means inserting the value as the first node in the list.

1.7. **remove_node** removes a node at a given position.

For both insert_node and remove_node, you need to check the validity of index: obviously index should not be negative; any other constraint?

Step 2: Command line interface

Then you will need to implement a command line interface to operate a linked list. I have provided a skeleton of the code in lab3main.c. You should take a look at it first.

This time we are not going to use scanf since it is hard to perform error checking and hard to take input with different formats. In our sample code, we use the getline function to read a whole line from stdin (i.e., keyboard), and then you can perform whatever you like on the line.

The command line interface should implement five commands, and actually the sample code already implements two:

exit: quits this tool

help: print all commands

print: print all values in the linked list

add <i> <value>: add value as the ith element

delete <i>: delete the ith element

"print" should be easy to implement.
To implement "add" and "delete", you need to break a line into multiple tokens. You can use the **strtok** function for this purpose. Read its manual and example code carefully before you start. You also need to convert a string to an integer. You can use the **strtol** function for this purpose. Similarly, read its manual and example code carefully before you start.

This time, you should do error checking with your best effort and give meaningful output when an error happens. For example, if the user types a command that is not any of the above five, you should print something like "Invalid command XXX"; if the user types a command without the expected number of arguments, you should print something like "add needs two arguments but you only give one".

Test cases: we will test everything through the command line interface

```
Case 1:
add 0 0
print           Should get 0

Case 2:
add 0 0
add 1 1
print           Should get 0->1

Case 3
add 0 0
add 1 1
add 2 2
add 3 3
print           Should get 0->1->2->3
delete 1
print           Should get 0->2->3
delete 0
print           Should get 2->3
delete 1
print           Should get 2

Case 4
haha            Error

Case 5
add 0           Error

Case 6
add 0 0
delete 2        Error
```

We will also use valgrind to check memory leak.

Grading criteria:

1. We will unzip your zip file, and run “make”. If it reports any errors, you will lose all points
2. Test cases with valid commands (2.5 points)
3. Test cases with invalid commands (1 points)
4. Your program should have no memory leak (0.5 points)

Suggestions: this is a relatively bigger project, with functions that you might not be familiar with. My suggestion is that if there is anything that you are not certain, write a small program to test it first. For example, if you are not sure how to use strtok or strtol, write a small program to test just these two functions. This will help you narrow down the problem if something is wrong.