CSE 1384 - Linked Lists
Lab 8

Objectives:

- Continue practicing past concepts
- Practice using linked lists in C++

Assignment:

For this assignment, we'll be practicing using linked lists. You'll begin the assignment with some starter files (linkedlist.cpp and linkedlist.h) and a driver file (main.cpp). The driver file will serve as a testing file; your screenshot should match mine exactly. **You should not change this file at all.** The LinkedList class has some initial functions already defined:

- Constructor
- Destructor
- Display - displays the linked list
- Append - adds an item to the end of the linked list
- Pop - removes the last item from the linked list

You should not change these functions already defined. Your job will be to complete the LinkedList class to make the driver (main.cpp) fully functional. The linked list will be a list of integers. The header file (linkedlist.h) has definitions for the functions you should be adding to the code (in the linkedlist.cpp). The return values and parameters are listed appropriately there. These functions are:

- linearSearch
    - Perform a linear search on the linked list to determine if any item is present. Return the index if present, else return -1
- sort
    - A sort of your choice -- sort the linked list *ascending*
- reverse
    - A sort of your choice -- sort the linked list *descending*
- min
    - Return the minimum number (should work on unsorted list)
- max
    - Return the maximum number (should work on unsorted list)
- mean
    - Return the average of the current numbers
- insert
    - Insert a node at the position sent. So, if position 1 is chosen, it'd be like index 1 in an array. Size should increase (AKA, the item should NOT overwrite the current)
- remove
    - Remove a node the position sent (starting at 0)

Hints:

Some of these functions will be *very* similar. Complete one and then make minor changes to get the other (sort and reverse; min and max). When sorting: consider swapping the data, not the actual nodes themselves.

Insert:

You should be able to handle the following: inserting to the head, inserting to the end of list, inserting in the middle of the list. Consider keeping track of the item before the one you're inserting and the item in the current place you're inserting to. You'll need both to not lose track of your list items.

Remove:

You should be able to handle the following: removing from the head, removing from the end of the list, and removing from the middle of the list. Consider keeping track of the item before the intending deleted item and the item after the deleted item. You'll need to link these. *Don't forget to explicitly delete the item using keyword "delete" to avoid memory leaks.*

Order of attack:

Consider implementing min/max/mean and the sort/search functions before the insert/remove. The insert/remove will more than likely be the trickiest.

Utilize commenting out portions of the driver to test as you go. Don't wait until the end to test every new function at once.

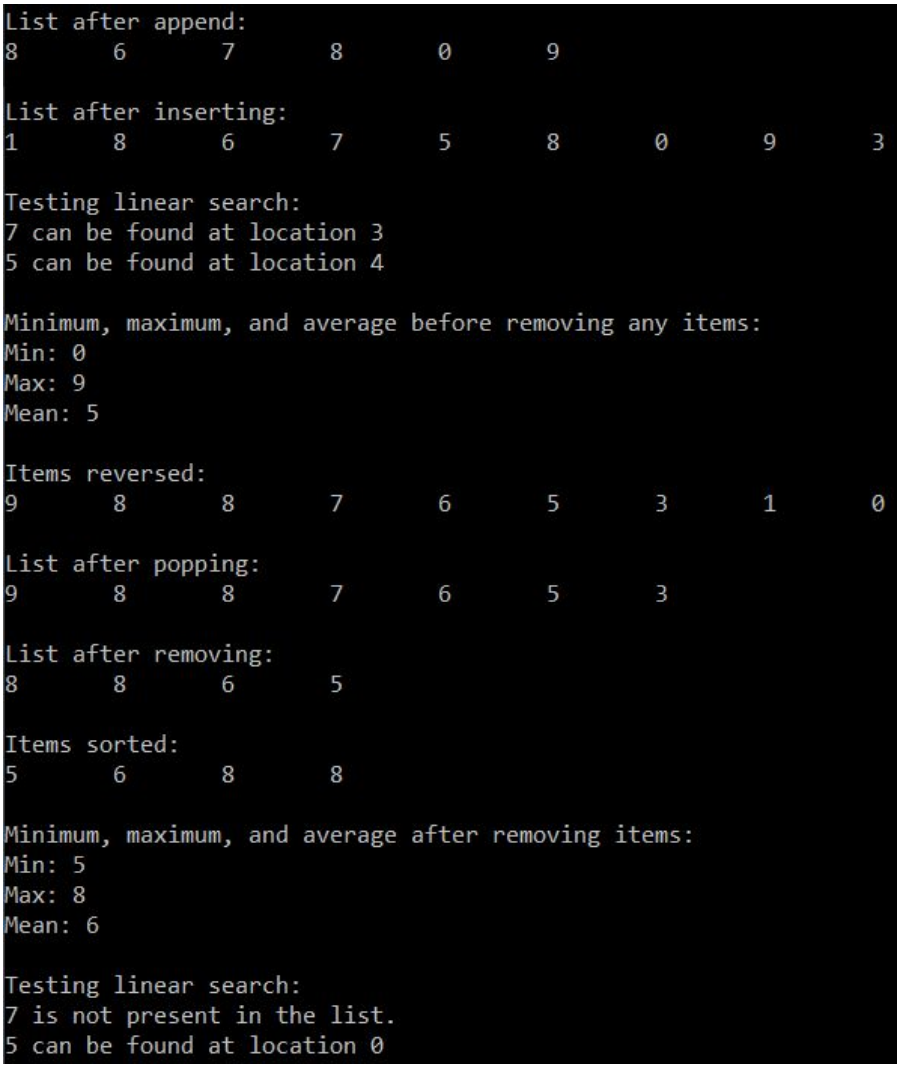
Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                      NetID: <your netID>
 Date: <current date>                   Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Execution Screenshot:



Your program (if everything is fully functional) should output this exactly.

Deliverables:

- C++ code (.cpp file)
- A document (.pdf) with a screenshot showing the program running
- Include in the PDF: which sorting algorithm you chose to use

Point Breakdown:
(100 points total)
A submission that doesn't contain any code will receive a 0.

- 10pts - pointers
    - 5pts - proper deallocation (deleting)
    - 5pts - proper allocation (creating a new instance of Node)
- 15pts - min, max, mean
    - 5pts each (correctness)
- 15pts - sorting, reversing, searching
    - 5pts each (correctness)
- 20pts - insert
    - 15pts - head, tail, middle insertion (5pts each)
    - 5pts - keeps proper track of head/tail/size when necessary
- 20pts - remove
    - 15pts - head, tail, middle removal (5pts each)
    - 5pts - keeps proper track of head/tail/size when necessary
- 10pts - execution screenshot
- 10pts - programming style *
- (penalty) -15pts - changing main.cpp OR linkedlist.h
- (penalty) -5pts - lack of sorting function identification

* Programming style includes good commenting, variable nomenclature, good whitespace, etc.

(Commenting out non-working portions of the main.cpp is the only acceptable change)