Samuel Whittenberger
COP3530
9/21/2018

Programming Assignment 1 Commentary

Computational Complexities:
InsertEnd - O(1)
Insert - O(n)
Delete - O(n)
Edit - O(n)
Search - O(n)
Print - O(n)

Thoughts on Linked List Use:

The use of a linked list for this project is appropriate due to the many insertions and deletions that take place in the middle of the list. Insertions and deletions from a linked list are quick and efficient since the list does not need to be shifted to accommodate the change in data. The next and previous pointers of the surrounding nodes can be easily changed to exclude or include a new node. If the line editor only supported insertions and deletions from the end of the list then an array based implementation would be preferable since there would be no need to shift all the entries of the array since the entries would be contiguous in memory. There might be a case to be made for an array based implementation over a linked list due to arrays being very fast for accessing data, but I still believe the pointer based linked list implementation is preferable.

What did I Learn/What Would I do Differently:

While implementing the linked list for this project, I learned how to properly add and remove nodes from a doubly linked list without leaving null pointers in the middle of the list. I also learned how to properly utilize a pointer to iterate through a list in order to find particular indices. Additionally, I decided to use the std::shared_ptr object rather than a basic C++ pointer due to the automatic memory management that shared pointers possess. Finally, I learned how to effectively manage user input from the console using C++.

I wouldn't do much differently if I had to complete this project again, but I would consider using a singly linked list implementation over a doubly linked list since singly linked lists are slightly less complicated. They also use half as many pointers which means less memory per list. Overall, I was happy with how my project was built and how it performs.