Text Editor

The data structure used is commonly referred to as a doubly linked list. The difference between a singly linked list and a doubly linked list is, with the former there is usually only one pointer pointing to the next node in the list, where the latter has two pointers with one pointing to the next node in the list and the other pointing to the previous node (with the end cases being null pointers in both implementations). The singly linked list is more memory efficient, but more performance inefficient. Double linked lists are the opposite, being more memory inefficient and more performance efficient.

Below is a table showing all of the time complexities for each of the methods in the program. The overall time complexity for the implementation turns out to be O(n).

Function	**Worst Case**	**Average case**
`del()`	O(1)	O(1)
`insert()`	O(1)	O(1)
`insertEnd()`	O(1)	O(1)
`find()`	O(n)	O(n)
`edit()`	O(n)	O(n)
`print()`	O(n)	O(n)
`parseString()`	O(1)	O(1)
`parseLineNum()`	O(1)	O(1)
`check()`	O(n)	O(n)

Using a linked list seemed to work fine as a text editor from a functional standpoint. It works great for instances where there aren't a lot of lines to include. Using a linked list is advantageous as a text editor over using an array if we don't want to limit the number of insertions the user can make. Additionally, inserting new information into a linked list is less expensive than it would be for an array since an array would require moving all the data from the point of insertion up one index. On the other hand, performance may take a hit with a high number of insertions (particularly with the search and edit commands). Further, accessing elements randomly in a linked list isn't allowed since we need to sequentially move through the list in order to reach the index of interest. Also, from a user standpoint it, seemed to be a little awkward and time-consuming writing commands and getting the notation right. If a GUI were successfully implemented, it would be a nice application to use as a grocery list, a to do list or… anything that involves a list. Although, a binary search tree would probably be a better choice when creating a list (using int key as the item number and string data as the content). It appears that gap buffers are most commonly used in modern text editors.

Doing this project has taught me a great deal on how to successfully implement a linked list. Most notably, I've learned a great deal on how to traverse a linked list while making sure that I'm not trying to invoke a NULL pointer. There were several times where I needed to reconsider my approach due to attempting to access a pointer that essentially didn't exist. I also learned how to consider end cases when creating my program, which weren't too obvious when writing my functions. In the future, it would benefit me to consider end case scenarios while sketching an outline of my program. This would have saved me a lot of time and headaches when writing my functions. Additionally, I poorly implemented my head and tails functions, which came to bite me later down the line. So, I would have considered these more when starting if I could do this over again.