Computational complexity of methods:
The computational complexity of insertEnd is O(1).
The computational complexity of insert is O(n).
The computational complexity of destroy is O(n).
The computational complexity of edit is O(n).
The computational complexity of search is O(n).
The computational complexity of print is O(n).

Thoughts on linked list for line editor; advantages/disadvantages?
I think a linked list is quite alright. Originally, I thought an array would be better but realised this is probably not true because people often delete lines when they are working. This would lead to an array based implementation becoming worse as the document grew. The advantages of a linked list implementation are easy access to the next line, which is the most common movement in creating a document, and low waste deletion. The disadvantages are that one must traverse the whole list to add or delete arbitrary lines. This is in stark contrast to array based implementations, where any access is O(1).

What did you learn? What would you do differently?
I learned about abstraction in terms of implementing a list with a linked list. I found that if I worked hard in the underlying methods of the program that the linking of the elements in the main() was relatively simple. I learned a great deal about general coding techniques in getting the different methods to all produce satisfactory results and spent a great deal of time looking at documentation.

```
#include <string>
#include <iostream>
#include <sstream>

class LineNode{
public:
        LineNode();
   LineNode(std::string someString);
        LineNode * next;
        std::string line;
};

class LineEditor{
private:

public:
        LineNode * head;
        LineNode * tail;
        int size = 0;
                        //so deleting when there's only one node keeps the node
        LineEditor();
        void insertEnd(std::string command);
        void insert(int number, std::string command);
        void destroy(std::string command);
        void print();
   void edit(int number, std::string command);
        void search(std::string command);
};




LineNode::LineNode() {
        this->next = NULL;
        this->line = "";
};


LineNode::LineNode(std::string someString) {
   this->line = someString;
        this->next = NULL;
};
```

```
LineEditor::LineEditor() {
	//this->size = 1;				//set size = 1 in header
	this->head = NULL;
	this->tail = NULL;
};

std::string cut(std::string command){
	while(command[0] != ""){
		command = command.substr(1, command.size() - 2);
	}
	command = command.substr(1, command.size() - 3);
	//rids string of first and last quotation mark
	return command;
}

void LineEditor::insertEnd(std::string command){
	//create new node if needed

	//make command just the string - I think this did it
	while(command[0] != ""){
			//makes command just the text to input
		command = command.substr(1, command.size() - 2);
	}
   if (command[0] != '\"' || command[command.length() - 1] != '\"')
      return;

	command = command.substr(1, command.size() - 2);
	if(command.size() > 80){
			//checking that the input is of legal size
		return;
	}


   if (this->head == NULL)
   {
      this->head = this->tail = new LineNode(command);
   }
   else
   {
      this->tail->next = new LineNode(command);
      this->tail = this->tail->next;
   }
```

```
        // if(this->size == 0){
        //for if there is only one node
        //      this->tail->line = command;                         //add line to node
    // }
        // else{
        //      LineNode* addNode = new LineNode();                 //add node
        //      this->tail->next = addNode;                         //add line to new node
        //      addNode = this->tail;                               //set tail to new node
        //      addNode->line = command;
        // }

        this->size++;
                                //increase size when inserting
        return;
}

void LineEditor::insert(int number, std::string command){
    int index = number, counter = 0;
    LineNode *prev = NULL, *curr = this->head, *next = NULL;
    if (number <= 0 || number > size+1)
        return;
    // head insert:
    if (number == 1)
    {
        if (this->head != NULL)
            next = this->head;
        this->head = new LineNode(command.substr(1, command.length() - 2));
        this->head->next = next;
        if (next == NULL)
            this->tail = this->head;
        return;
    }

    while (curr != NULL && counter != number - 1)
    {
        counter++;
        prev = curr;
        curr = curr->next;
    }

    prev->next = new LineNode(command.substr(1, command.length() - 2));
    prev->next->next = curr;
    if (curr == NULL)
```

```
        tail = prev->next;

    this->size++;


        // int index = number;                                                  //FIXME: what to
do if bad input after "insert"
        // if(index < 1){
                                //checks for bad (negative) input
        //        return;
        // }
        // index--;
                                        //makes index variable zero indexed
    //
        // if(index == this->size){
                        //if node to add is at the end, reduces case to insertEnd
        //        // insertEnd(command.substr(2,command.length() - 2));
        //takes care of end node and if only one node
    //    if (this->head == NULL)
    //        this->head = this->tail = new LineNode(command.substr(1, command.length() - 2));
    //    else
    //    {
    //        this->tail->next = new LineNode(command.substr(1, command.length() - 2));
    //        this->tail = this->tail->next;
    //
    //    }
    //    return;
        // }
    //
        // if(index < this->size - 1){
        //        LineNode * addNode = new LineNode();
        //        while(command[0] != ""){
                        //makes command just the text to input
        //                command = command.substr(1, command.size() - 2);
        //        }
        //        command = command.substr(1, command.size() - 3);
        //        if(command.size() > 80){
                        //checking that the input is of legal size
        //                return;
        //        }
    //
        //        if(index == 0){
                                //takes care of head node
```

```
        //              addNode->next = this->head;
        //              this->head = addNode;
        //      }
        //      else{
        //              int i = 0;
                                        //iterates to index and inserts addNode;FIXME: could make
a for loop
        //              LineNode * currNode = this->head;
        //              while(i < index - 1){
        //                      currNode = currNode->next;
        //                      i++;
        //              }
        //              addNode->next = currNode->next;
        //              currNode->next = addNode;
        //              addNode->line = command;
                        //adds line to addNode
        //      }
        //      this->size++;
//increase size when inserting; done here so large index doesn't increment size
        // }
        // return;
}

void LineEditor::destroy(std::string command){
        int index = std::stoi(command.substr(0, 1), nullptr, 10);
   LineNode *next = head;
        if(index < 1 || index > size){
                                                //checks for bad (negative) input
                return;
        }
   index--;
                                //makes index variable zero indexed
        if(this->size <= 0){
                        //nothing to delete
                return;
        }

        if(this->size == 1){
                        //if only one line present
      next = head->next;
      delete this->head;
      if (next == NULL)
        this->head = this->tail = NULL;
```

```
        head = next;
                this->size--;
                return;
        }

    if (index == 0)
    {
        LineNode* tempNext = head->next;
        delete head;
        this->size--;
        head = tempNext;
        if (tempNext == NULL)
            tail = head = NULL;
        return;
    }

        if(index <= this->size - 1){
                int i = 0;
                LineNode* currNode = this->head;
                while(currNode != NULL){
                        if(i + 1 == index){
                                //FIXME: account for if node to delete is tail
                                LineNode* temp = currNode->next;
                                currNode->next = currNode->next->next;
                                delete temp;
                                if(i + 2 == this->size){
                                        this->tail = currNode;
                                }
                        }
                        currNode = currNode->next;
                        i++;
                }
                this->size--;
                                        //decrement when deleting
        }

        return;
}

void LineEditor::edit(int number, std::string command){

        int index = number, counter = 0;
        LineNode *prev = NULL, *curr = this->head, *next = NULL;
```

```
// head insert:
if (number == 1)
{
	this->head->line = command.substr(1, command.length() - 2);
	return;
}

while (curr != NULL && counter != number - 1)
{
	counter++;
	prev = curr;
	curr = curr->next;
}

prev->next->line = command.substr(1, command.length() - 2);


// int index = std::stoi(command.substr(0, 1), nullptr, 10);
// index--;
                                    //makes index variable zero indexed
// if(index < 0 || index > this->size - 1){
//illegal index
//         return;
// }
//
// while(command[0] != ""){
                    //makes command just the text to input
//         command = command.substr(1, command.size() - 2);
// }
// command = command.substr(1, command.size() - 3);
// if(command.size() > 80){
                    //checking that the input is of legal size
//         return;
// }
//
// int i = 0;
// LineNode* currNode = this->head;
// while(i < index){
//         i++;
//         currNode = currNode->next;
// }
// currNode->line = command;
```

```
        //
        // return;
}

void LineEditor::search(std::string command){
        // while(command[0] != ""){
                        //makes command just the text to input
        //      command = command.substr(1, command.size() - 2);
        // }
        command = command.substr(1, command.size() - 2);
   bool wordFound = false;
        // if(command.size() > 80){
                        //checking that the input is of legal size
        //      return;
        // }

        int i = 0;
        LineNode* currNode = this->head;
        while(currNode != NULL){
                if(currNode->line.find(command) != std::string::npos){
                        std::cout<<i + 1<<" "<<currNode->line<<std::endl;
        wordFound = true;
                }
                i++;
                currNode = currNode->next;
        }
        // currNode->line = command;
   if (wordFound == false)
      std::cout << "not found" << std::endl;
        return;



}

void LineEditor::print(){
        int i = 0;
        LineNode* currNode = this->head;
        while(currNode != NULL){
                std::cout<<i + 1<<" "<<currNode->line<<std::endl;
                i++;
                currNode = currNode->next;
        }
```

```
        return;
}

std::string HelpWithCommand(std::string typedIn)
{
   std::string returnWord;
   std::istringstream iss(typedIn);
   iss >> returnWord;
   return returnWord;
}

int main(){
        // std::cout << "test" << std::endl;
        LineEditor * myEditor = new LineEditor();


        std::string command;
   std::string justTheCommand;
        std::string first;

        while(true){
                command = "";
                first = "";
                // std::getline(std::cin, first, ' ');

      std::getline(std::cin, first);
      justTheCommand = HelpWithCommand(first);

                if(justTheCommand.compare("insertEnd") == 0){
                //      std::cout<<"Hit insertEnd function"<<std::endl;
                                                                //FIXME: remove
                        myEditor->insertEnd(first.substr(10, first.length() - 10));
                }
                else if(justTheCommand.compare("insert") == 0){
                //      std::cout<<"Hit insert function"<<std::endl;
                                                        //FIXME: remove

        std::string temp1, temp2, temp3;
        std::istringstream iss(first);
        iss >> temp1;

        iss >> temp1;
```

```
if (first[first.length() - 1] == '\"')
{
   iss >> temp3;
   temp2 += temp3;
   while (temp2[temp2.length() - 1] != '\"')
   {
      iss >> temp3;
      temp2 += " ";
      temp2 += temp3;
   }
}
int theNumber = stoi(temp1);

myEditor->insert(theNumber, temp2);
// myEditor->insert(int number, std::string string)
         }
         else if(justTheCommand.compare("delete") == 0){
         //       std::cout<<"Hit delete function"<<std::endl;

std::istringstream iss(first);
iss >> command;
iss >> command;

                  myEditor->destroy(command);
         }
         else if(justTheCommand.compare("edit") == 0){
         //       std::cout<<"Hit edit function"<<std::endl;
                                                //FIXME: remove

                  std::string temp1, temp2, temp3;
                  std::istringstream iss(first);
                  iss >> temp1;

                  iss >> temp1;

                  if (first[first.length() - 1] == '\"')
                  {
                           iss >> temp3;
                           temp2 += temp3;
                           while (temp2[temp2.length() - 1] != '\"')
                           {
                                    iss >> temp3;
                                    temp2 += " ";
```

```
                        temp2 += temp3;
                    }
                }
                int theNumber = stoi(temp1);
                if (theNumber > 0)
                    myEditor->edit(theNumber, temp2);
            }
            else if(justTheCommand.compare("search") == 0){
            //      std::cout<<"Hit search function"<<std::endl;
                                                    //FIXME: remove
                std::istringstream iss(first);
                std::string temp1, temp2, temp3;

                iss >> temp3;
                if (first[7] == '\"' && first[first.length() - 1] == '\"')
                {
                    iss >> temp3;
                    temp2 += temp3;
                    while (temp2[temp2.length() - 1] != '\"')
                    {
                        iss >> temp3;
                        temp2 += " ";
                        temp2 += temp3;
                    }
                }
                if (temp2[0] == '\"' && temp2[temp2.length() - 1] == '\"')
                    myEditor->search(temp2);
            }
            else if(justTheCommand.compare("print") == 0){
            //      std::cout<<"Hit print function"<<std::endl;
                                                    //FIXME: remove
                myEditor->print();
            }
            else if(justTheCommand.compare("quit") == 0){
            //      std::cout << "Quit correctly" << std::endl;
                                                    //FIXME: remove
                return 0;
            }
        }
}
```