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


//your linked list implementation here
class Node{
public:

    //Data Members
    std::string Line;
    Node* next = NULL;
    Node* prev = NULL;

    //Member Functions
    void setLine(std::string userLine);

};

//Member Functions
void Node::setLine(std::string userLine) {
    //Checks if the users line to insert is less than 82 characters,
if not,
    //truncates and then sets line to userinput.
    if (userLine.length()>82) {
        userLine=userLine.substr(0,82);
    }
    Line= userLine;

}

//your line editor implementation here

class LineEditor {

public:

    //Data Members
    std::string userInput = "";
    Node* head = new Node();
    Node* curr = head;
    //Member Functions
    void insertEnd(std::string textoInsert);
    void listen(std::string userInput);
    void print();
    void quitLE();
    void search(std::string texttoSearch);
    void edit(std::string replacementText, int lineNumber);
    void deletefromList(int lineNumber);
```

```
    void insert(std::string textoInsert, int lineNumber);

};

void LineEditor::insert(std::string texttoInsert, int lineNumber) {

    //Count number of lines
    int UpperBound = 1;
    while (curr->next!=NULL) {
        UpperBound++;
        curr=curr->next;
    }

    //Reset curr pointer
    curr=head;
    //If insert is not in bounds give error message
    if (lineNumber<0 || lineNumber>UpperBound+1) {
        listen("");
    }

    //If insert is at the end of the list use insertend
    if (lineNumber==UpperBound+1) {
        insertEnd(texttoInsert);
    }

    //If insert is at the begining and head is empty and there are no
further nodes
    if (lineNumber==1 && head->Line.empty() && head->next==NULL) {
        //Use insertEnd
        insertEnd(texttoInsert);
    }

    //If insert is at the head and further nodes exist
    if (lineNumber==1 && head->next!=NULL) {

        //Point next to deleted node's next
        Node* newNode = new Node();
        newNode->setLine(texttoInsert);

        //Point existing nodes to new node
        curr->prev = newNode;

        //Point new node to existing nodes
        newNode->next = curr;

        //Reset pointers
        head = newNode;
        curr = head;

        //Listen for more instructions
```

```
        listen("");

    } else {
        //Iterate to right line
        for (int i = 1; i < lineNumber; i++) {
            curr = curr->next;
        }

        //Create new node and set text
        Node* newNode = new Node();
        newNode->setLine(texttoInsert);

        //Point existing nodes to new node
        if (lineNumber!=1) {
            //No previous if newnode is the head
            curr->prev->next = newNode;
        }
        curr->prev = newNode;

        //Point new node to existing nodes
        //No previous if newnode is the head
        if (lineNumber!=1) {
            newNode->prev = curr->prev;
        }
        newNode->next = curr;

        //Reset pointers
        if (lineNumber==1) {
            head = newNode;
        }
        curr = head;

        //Listen for more instructions
        listen("");
    }
}



void LineEditor::deletefromList(int lineNumber) {

    //Count number of lines
    int UpperBound = 1;
    while (curr->next!=NULL) {
        UpperBound++;
        curr=curr->next;
    }

    //Reset curr pointer
    curr=head;
```

```
    //If delete is not in bounds give error message
    if (lineNumber<0 || lineNumber>UpperBound) {
        //std::cout << "Insert is out of bounds, please try again" <<
std::endl;
        listen("");
    }

    //Head case
    if (lineNumber==1) {
        head = head->next;
        curr = head;
    } else {

        //Iterate to right line
        for (int i = 1; i < lineNumber; i++) {
            curr = curr->next;
        }

        //Point next to deleted node's next
        curr->prev->next = curr->next;

        //Point prev to deleted node's prev
        if (lineNumber!=UpperBound) {
            curr->next->prev = curr->prev;
        }

        //Delete the node
        delete curr;

        //Reset curr pointer
        curr=head;

        //Listen for more commands
        listen("");
    }
}


void LineEditor::edit(std::string replacementText, int lineNumber) {
    //Line counter
    for (int i = 1; i < lineNumber; i++) {
        curr = curr->next;
    }
    //Edit the text
    curr->setLine(replacementText);

    //Reset pointers
    curr = head;
```

```
    //Listen for more instructions
    listen("");
}

void LineEditor::search(std::string texttoSearch) {

    //Line counter
    int i = 1;
    //Determines if the initial search was a success
    int SearchSuccess = 0;

    //While the next node is not empty
    while (curr->next!=NULL) {

        //If the text to search is in the line
        if (curr->Line.find(texttoSearch) < 83) {

            //Output line and linenumber
            std::cout << i++ << " " << curr->Line << std::endl;

            //Set initial success to 1
            SearchSuccess=1;
        }

        //Increment Line counter
        if (curr->Line.find(texttoSearch) > 83) {
            i++;
        }

        //Move to the next node
        curr=curr->next;
    }
    if (curr->Line.find(texttoSearch) < 83) {

        //Output line and linenumber
        std::cout << i << " " << curr->Line << std::endl;

    } else if (SearchSuccess==0) {

        //Else initial fails and last line fails, output error.
        std::cout << "not found" << std::endl;

    }

    //Reset pointer
    curr = head;

    //Relisten for commands.
    listen("");
```

```
}

void LineEditor::quitLE() {
    //Exit with code 0
    exit(0);
}

void LineEditor::print() {
    //Line number counter
    int i = 1;

    //While the next node is not empty
    while (curr->next!=NULL) {

        //Increment the line number and print the current line
        std::cout << i++ << " " << curr->Line << std::endl;

        //Move to the next node
        curr=curr->next;

    }

    //Print the last nodes line
    std::cout << i++ << " " << curr->Line << std::endl;

    //Reset the curr pointer to head
    curr = head;

    //Listen for more commands
    listen("");
}

void LineEditor::insertEnd(std::string textoInsert) {
    //If the head of the list is empty insert at the existing head
    if (head->Line.empty()){
        head->setLine(textoInsert);
    } else {
        //Iterate to the correct node
        while (curr->next!=NULL) {
            curr=curr->next;
        }
        //Make sure curr next is NULL
        if (curr->next==NULL) {

            //Create a new node
            curr->next=new Node();

            //Set the text
            curr->next->setLine(textoInsert);
```

```
            //Set existing nodes to point to new node
            curr->next->prev=curr;

            //Reset pointer
            curr=head;
        }
    }
    //Listen for more instructions
    listen("");

}

void LineEditor::listen(std::string userInput) {

    while (userInput == "") {
        //Grab user input for the command
        getline(std::cin, userInput);
    }

    //Truncate user input in order to find command
    std::string truncateduserInput = userInput.substr(0,9);

    if (truncateduserInput.find("insertEnd") == 0) {

        //Uses space as delimiter to get text to insert from user
command
        size_t firstspace = userInput.find(" ");

        //Generates texttoInsert using delimiter
        std::string texttoInsert = userInput.substr(firstspace+2);

        //Removes final "
        texttoInsert.pop_back();

        //Calls the function
        insertEnd(texttoInsert);

    }

    else if (truncateduserInput.find("insert ") == 0) {
        //Finding the positions of the first and second space to
seperate inpute
        size_t firstSpace = userInput.find(" ");
        size_t nextSpace = userInput.find_first_of(" " ,
firstSpace+1);

        //Get the line number as a string and then use stringstream to
convert into an integer
        std::string linenumberString =
userInput.substr(firstSpace+1,nextSpace-firstSpace);
```

```
        std::stringstream stringtoNum(linenumberString);
        int lineNumber = 0;
        stringtoNum >> lineNumber;

        //Create a substring beginning after the " of the text
        std::string texttoInsert = userInput.substr(nextSpace+2);

        //Pop the final " off the back of the substring
        texttoInsert.pop_back();

        //Call insert
        insert(texttoInsert, lineNumber);

    }

    else if (truncateduserInput.find("delete ") == 0) {

        //Find pos of first space to use as delimiter
        size_t firstSpace = userInput.find(" ");

        //Get the line number as a string and then use stringstream to
convert into an integer
        std::string linenumberString = userInput.substr(firstSpace+1);
        std::stringstream stringtoNum(linenumberString);
        int lineNumber = 0;
        stringtoNum >> lineNumber;

        //Calls the function
        deletefromList(lineNumber);
    }

    else if (truncateduserInput.find("edit") == 0) {

        //Finding the positions of the first and second space to
seperate inpute
        size_t firstSpace = userInput.find(" ");
        size_t nextSpace = userInput.find_first_of(" " ,
firstSpace+1);

        //Get the line number as a string and then use stringstream to
convert into an integer
        std::string linenumberString =
userInput.substr(firstSpace+1,nextSpace-firstSpace);
        std::stringstream stringtoNum(linenumberString);
        int lineNumber = 0;
        stringtoNum >> lineNumber;

        //Create a substring beginning after the " of the text
        std::string texttoReplace = userInput.substr(nextSpace+2);
```

```
        //Pop the final " off the back of the substring
        texttoReplace.pop_back();

        //Call edit function with info.
        edit(texttoReplace, lineNumber);
    }

    else if (truncateduserInput.find("print") == 0) {
        //Call print function
        print();
    }

    else if (truncateduserInput.find("search ") == 0) {

        //find beginning of text to search with space delimiter
        size_t firstspace = userInput.find(" ");

        //seperate text to search into substring starting at first
char
        std::string texttoSearch = userInput.substr(firstspace+2);

        //Removes final "
        texttoSearch.pop_back();

        //Call search function
        search(texttoSearch);
    }

    else if (truncateduserInput.find("quit") == 0) {
        //Call quit function
        quitLE();
    }




}


int main()
{
    //your code to invoke line editor here
    //Create a line editor
    LineEditor* lineEditor = new LineEditor;
    //Call the listen function
    lineEditor->listen("");
    return 0;
}
```

Commentary

Computational Complexity

- Node::setline
  - O(1)
- LineEditor::insert
  - O(n)
- LineEditor::deletefromList
  - O(n)
- LineEdtor::edit
  - O(n)
- LineEditor::search
  - O(n)
- LineEditor::quitLE
  - O(1)
- LineEditor::Print
  - O(n)
- LineEditor::insertEnd
  - O(n)
- LineEditor::listen
  - O(1)

Thoughts on using Linked List

I think that using a linked list for the line editor was a good choice. It is flexible in the way that you can easily add and remove lines from the list without reallocated memory like an array would have to do. The disadvantages of using the linked list was that any time you had to access a particular line you would have to iterate to the line (O(n)) instead of directly calling it like you could in an array or vector implementation (constant time).

What I learned from this assignment

Firstly this assignment required me to make my own implementation of a linked list, so I became knowledgeable of all the functions that would require. This assignment also was one of the first times that I worked with header files in C++.

What I would change or do differently if I had to start over.

If I had to start over I would attempt to make more of my functions with less computational complexity, because they were all either O(n) or O(1). I would attempt to bring some of the O(n) functions to O(log n). Another thing I would have done differently is try and have more of the functions under the Node class. I feel like I used Node simply to hold the data for the linked list and used the Line Editor as the linked list implementation of those nodes. There may have been a way to have the Node class perform more functions that the Linked List implementation could have called.