```cpp
//
// Created by mark- on 2023-01-22.
//

#include "LinkedListNode.h"

```

```cpp
//
// Created by mark- on 2023-01-22.
//

#ifndef ASSIGNMENT1_LINKEDLIST_H
#define ASSIGNMENT1_LINKEDLIST_H
#include "LinkedListNode.h"
#include "iostream"


class LinkedList {
private:
    LinkedListNode *m_start{nullptr};
    int m_size{0};
public:
    LinkedList();
    void add(std::string data);
    void insert(std::string data, int index);
    void remove(int index);
    void remove(int start, int end);
    void list();
    void list(int lineNum);
    void list(int start, int end);
    int printLastNum();
    friend std::ostream &operator<<(std::ostream &output, LinkedList &list);
};


#endif //ASSIGNMENT1_LINKEDLIST_H

```

```cpp
//
// Created by mark- on 2023-01-22.
//

#include "LinkedList.h"
#include <iostream>
#include <string>
#include <fstream>

LinkedList::LinkedList() {
    LinkedListNode *m_start{nullptr};
    int m_size{0};
}

void LinkedList::add(std::string data) {
    // create a new node
    auto node = new LinkedListNode();
    node->m_data = data;
    if (m_start == nullptr) {
        // add the first node to the list
        m_start = node;
    } else {
        //add to the end of the list
        LinkedListNode *current = m_start;
        LinkedListNode *previous = nullptr;

        //look for the end of the chain
        while (current != nullptr) {
            previous = current;
            current = current->m_next;
        }
        //attach the new node
        previous->m_next = node;
    }
    m_size++;
}

void LinkedList::insert(std::string data, int index) {

    if (index > m_size) {
        return add(data);
    }

    // create a new node
    auto node = new LinkedListNode();
    node->m_data = data;

    //find the index we are inserting before
    auto current = m_start;
    LinkedListNode *previous = nullptr;

    auto count{1};
    while (current != nullptr) {
        if (count++ == index) {
            break;
        }
        previous = current;
        current = current->m_next;
    }
    // am i inserting at the beginning?
    if (previous == nullptr) {
        //insert at the start of the list
        node->m_next = m_start;
        m_start = node;
    } else {
        // inserting in the middle of the list
        node->m_next = previous->m_next;
        previous->m_next = node;
    }
    m_size++;
}

void LinkedList::remove(int index) {

    //find the node to delete
    auto node = m_start;
    LinkedListNode *prev = nullptr;

    auto count{1};
    while (node != nullptr) {
        // look for the desired index
        if (count++ == index) {
            break;
        }
        prev = node;
        node = node->m_next;
    }
    // did we find the node we are looking for?
    if (node != nullptr) {

        // am i deleting the first node?
        if (prev == nullptr) {
            //first node
            m_start = node->m_next;
        } else {
            //other node
            prev->m_next = node->m_next;
        }

        // finally
        delete node;
    }
    m_size--;
}

void LinkedList::remove(int start, int end) {
    for (int i = start; i <= end; i++) {
        remove(start);
    }
}

//void LinkedList::remove(int start, int end) {
//    auto node = m_start;
//    LinkedListNode *prev = nullptr;
//    int fromStart = start;
//    int lineCounter = 1;
//    while (node != nullptr) {
//        if (lineCounter >= fromStart && lineCounter <= end) {
//            fromStart++;
//            if (node != nullptr) {
//
//                // am i deleting the first node?
//                if (prev == nullptr) {
//                    //first node
//                    m_start = node->m_next;
//                } else {
//                    //other node
//                    prev->m_next = node->m_next;
//                }
//                // finally
//                delete node;
//            }
//            m_size--;
//        }
//
//        prev = node;
//        node = node->m_next;
//        if (lineCounter == end + 1) {
//            break;
//        }
//        lineCounter++;
//    }
////    while (node != nullptr) {
////        while (lineCounter >= start && lineCounter <= end) {
////            // look for the desired index
////            if (lineCounter == start) {
```

```cpp
////                fromStart++;
////                break;
////            }
////            prev = node;
////            node = node->m_next;
////            lineCounter++;
////        }
////        // did we find the node we are looking for?
////        if (node != nullptr) {
////
////            // am i deleting the first node?
////            if (prev == nullptr) {
////                //first node
////                m_start = node->m_next;
////            } else {
////                //other node
////                prev->m_next = node->m_next;
////            }
////
////            // finally
////            delete node;
////        }
////    }
////    m_size--;
//}

void LinkedList::list() {
    auto node = m_start;
    LinkedListNode *prev = nullptr;
    auto counter = 1;
    while (node != nullptr) {
        std::cout << counter << "> " << node->m_data << " " << "\n";
        node = node->m_next;
        counter++;
        if (node == nullptr) {
            break;
        }
    }
}

void LinkedList::list(int index) {

    auto node = m_start;
    LinkedListNode *prev = nullptr;
    int lineCounter = 1;
    while (node != nullptr) {
        if (lineCounter == index) {
            std::cout << index << "> " << node->m_data << " " << "\n";
        }
        node = node->m_next;
        if (lineCounter == index) {
            break;
        }
        lineCounter++;
    }
}

void LinkedList::list(int start, int end) {


    auto node = m_start;
    int counter2 = start;
    int lineCounter = 1;
    while (node != nullptr) {
        if (lineCounter >= counter2 && lineCounter <= end) {
            std::cout << counter2 << "> " << node->m_data << " " << "\n";
            counter2++;
        }
        node = node->m_next;
        lineCounter++;

        if (lineCounter == end + 1) {
            break;
        }
    }
}


std::ostream &operator<<(std::ostream &output, LinkedList &list) {
    auto node = list.m_start;
    std::ofstream myFileOut;
    myFileOut.open("test.txt", std::ios::out);
    while (node != nullptr) {
        output << node->m_data << " " << "\n";
        myFileOut << node->m_data << " " << "\n";
        node = node->m_next;
    }
    //myFileOut << list;
    myFileOut.close();
    return output;
}


int LinkedList::printLastNum() {
    auto node = m_start;
    LinkedListNode *prev = nullptr;
    auto counter = 1;
    while (node != nullptr) {
        node = node->m_next;
        counter++;
        if (node == nullptr) {
//            std::cout << counter << "> ";
            break;
        }
    }
    return counter;
}
```

```cpp
#include <iostream>

#include <string>

#include <fstream>
#include <string>
#include <iostream>
#include <exception>
#include <cstdlib>
#include "LinkedList.h"
#include "ReadFile.h"
#include "TextEditor.h"

using namespace std;

int main(int argc, char *argv[]) {

    if (argc == 2) {
        LinkedList linkedList;
        TextEditor textEditor;

        linkedList =  ReadFile::readfile(argv[1],linkedList);
        linkedList = textEditor.startTextEditor(linkedList);
        cout << linkedList << endl;
    }
    else{
        cout << "Check Command Line Arguments" << endl;
    }


    return 0;
}
```

```
//
// Created by mark- on 2023-01-22.
//

#ifndef ASSIGNMENT1_READFILE_H
#define ASSIGNMENT1_READFILE_H
#include "LinkedList.h"


class ReadFile {

public:
    static LinkedList readfile(std::string argument, LinkedList linkedList);

};


#endif //ASSIGNMENT1_READFILE_H

```

```
//
// Created by mark- on 2023-01-22.
//

#ifndef ASSIGNMENT1_LINKEDLISTNODE_H
#define ASSIGNMENT1_LINKEDLISTNODE_H
#include <iostream>

class LinkedListNode {
public:
    std::string m_data = "0";
    LinkedListNode *m_next{nullptr};

};


#endif //ASSIGNMENT1_LINKEDLISTNODE_H

```

```cpp
//
// Created by mark- on 2023-01-22.
//
#include <iostream>

#include <string>

#include <fstream>
#include <string>
#include <iostream>
#include <exception>
#include <cstdlib>
#include "ReadFile.h"
#include "LinkedList.h"

using namespace std;

LinkedList ReadFile::readfile(std::string argument, LinkedList linkedList) {
    try {
        string line; // declaring string
        fstream myFileIn; // file in stream reading and writing
        ofstream myFileOut; //  file out stream writing only
        myFileIn.open(argument, ios::in | ios::out); // original txt file
        // open for writing
        if (myFileIn.is_open()) {
            cout << "File Open" << endl; // confirmation of successful file open
            while (!myFileIn.eof()) { // continue until end of file
                getline(myFileIn, line);
                linkedList.add(line);
            }

            myFileIn.close(); // closing file in stream

            cout << "File closed" << endl;
            return linkedList;
        } else {
            cout << "Input file failed to open. Will make new File on Exit." << endl;

            return linkedList;
        }
//
    }
//        catch (MyException& e) {
//            cout << e.error() << endl;
//        }
    catch (exception &e) {
        cout << "Generic error" << endl;
    }
    catch (...) {
        cout << "General error" << endl;
    }
}
```

```
//
// Created by mark- on 2023-01-22.
//

#ifndef ASSIGNMENT1_TEXTEDITOR_H
#define ASSIGNMENT1_TEXTEDITOR_H
#include "LinkedList.h"


class TextEditor {

public:
    LinkedList startTextEditor(LinkedList linkedList);
};


#endif //ASSIGNMENT1_TEXTEDITOR_H

```

```cpp
//
// Created by mark- on 2023-01-22.
//

#include "TextEditor.h"
#include <iostream>
#include <sstream>
#include <string>
#include "LinkedList.h"

//int cursorPosition;
LinkedList TextEditor::startTextEditor(LinkedList linkedList) {
    std::string input;
    linkedList.list();
    char command;
    int start, end;
    int cursorPosition;
    int count = 0;
    while (input != "E") {
        start = '\0';

        if (count == 0) {

            cursorPosition = linkedList.printLastNum();
            std::cout << cursorPosition << "> ";
        }
        getline(std::cin, input);

        std::stringstream ss;
        std::stringstream ss2;
        std::stringstream ss3;
        ss << input;
        ss2 << input;
        ss3 << input;

        ss >> command >> start >> end;
        if (!ss) {
            // command with start and end not entered
            ss2 >> command >> start;
            if (!ss2) {
                // command with start not entered
                ss3 >> command;
                if (!ss3) {
                    // no commands entered
                } else {

                    // just command is entered
                    if (command == 'L') {
                        if (count == 0) {
                            cursorPosition = linkedList.printLastNum();
                        }
                        linkedList.list();
                    }
                    if (command == 'I') {
                        if (count == 0) {
                            std::string data;
                            getline(std::cin, data);
                            linkedList.insert(data, linkedList.printLastNum());
                            cursorPosition = linkedList.printLastNum();
                        } else {
                            std::string data;
                            getline(std::cin, data);
                            std::cout << cursorPosition << "> ";
                            linkedList.insert(data, cursorPosition);
                        }
                    }
                    if (command == 'D') {
                        if (count == 0) {
                            linkedList.remove(linkedList.printLastNum());
                            cursorPosition = linkedList.printLastNum();
                        } else {

                            linkedList.remove(cursorPosition);
                        }
                    }
                }
            } else {
                // command and index is entered
                cursorPosition = start;
                if (command == 'L') {

                    linkedList.list(start);
                }
                if (command == 'I') {
                    std::string data;
                    std::cout << cursorPosition << "> ";
                    getline(std::cin, data);
                    linkedList.insert(data, start);
                }
                if (command == 'D') {
                    linkedList.remove(start);
                }
            }
        } else {
            // command with start and end entered
            cursorPosition = start;
            if (command == 'L') {
                cursorPosition = start;
                linkedList.list(start, end);
            }
            if (command == 'D') {
                cursorPosition = start;
                linkedList.remove(start, end);
            }
        }
        count++;
        if (count == 0) {
            cursorPosition = linkedList.printLastNum();
        } else {
            std::cout << "\n" << cursorPosition << "> ";
        }
    } //  end while loop

    return linkedList;
} // end texteditor function


```