```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};
}

LinkedList::~LinkedList() {
    auto node = m_start;
    while (node != nullptr) {
        auto temp = node;
        node = node->m_next;
        delete temp;
    }
}

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;
```

```
74      } else {
75          // inserting in the middle of the list
76          node->m_next = previous->m_next;
77          previous->m_next = node;
78      }
79      m_size++;
80  }
81
82  void LinkedList::remove(int index) {
83
84      //find the node to delete
85      auto node = m_start;
86      LinkedListNode *prev = nullptr;
87
88      auto count{1};
89      while (node != nullptr) {
90          // look for the desired index
91          if (count++ == index) {
92              break;
93          }
94          prev = node;
95          node = node->m_next;
96      }
97      // did we find the node we are looking for?
98      if (node != nullptr) {
99
100         // am i deleting the first node?
101         if (prev == nullptr) {
102             //first node
103             m_start = node->m_next;
104         } else {
105             //other node
106             prev->m_next = node->m_next;
107         }
108
109         // finally
110         delete node;
111     }
112     m_size--;
113 }
114
115 void LinkedList::remove(int start, int end) {
116     for (int i = start; i <= end; i++) {
117         remove(start);
118     }
119 }
120
121 //void LinkedList::remove(int start, int end) {
122 //    auto node = m_start;
123 //    LinkedListNode *prev = nullptr;
124 //    int fromStart = start;
125 //    int lineCounter = 1;
126 //    while (node != nullptr) {
127 //        if (lineCounter >= fromStart && lineCounter <= end) {
128 //            fromStart++;
129 //            if (node != nullptr) {
130 //
131 //                // am i deleting the first node?
132 //                if (prev == nullptr) {
133 //                    //first node
134 //                    m_start = node->m_next;
135 //                } else {
136 //                    //other node
137 //                    prev->m_next = node->m_next;
138 //                }
139 //                // finally
140 //                delete node;
141 //            }
142 //            m_size--;
143 //        }
144 //
145 //        prev = node;
146 //        node = node->m_next;
```

```
147 //        if (lineCounter == end + 1) {
148 //            break;
149 //        }
150 //        lineCounter++;
151 //    }
152 ////    while (node != nullptr) {
153 ////        while (lineCounter >= start && lineCounter <= end) {
154 ////            // look for the desired index
155 ////            if (lineCounter == start) {
156 ////                fromStart++;
157 ////                break;
158 ////            }
159 ////            prev = node;
160 ////            node = node->m_next;
161 ////            lineCounter++;
162 ////        }
163 ////        // did we find the node we are looking for?
164 ////        if (node != nullptr) {
165 ////
166 ////            // am i deleting the first node?
167 ////            if (prev == nullptr) {
168 ////                //first node
169 ////                m_start = node->m_next;
170 ////            } else {
171 ////                //other node
172 ////                prev->m_next = node->m_next;
173 ////            }
174 ////
175 ////            // finally
176 ////            delete node;
177 ////        }
178 ////    }
179 ////    m_size--;
180 //}
181
182 void LinkedList::list() {
183     auto node = m_start;
184     LinkedListNode *prev = nullptr;
185     auto counter = 1;
186     while (node != nullptr) {
187         std::cout << counter << "> " << node->m_data << " " << "\n";
188         node = node->m_next;
189         counter++;
190         if (node == nullptr) {
191             break;
192         }
193     }
194 }
195
196 void LinkedList::list(int index) {
197
198     auto node = m_start;
199     LinkedListNode *prev = nullptr;
200     int lineCounter = 1;
201     while (node != nullptr) {
202         if (lineCounter == index) {
203             std::cout << index << "> " << node->m_data << " " << "\n";
204         }
205         node = node->m_next;
206         if (lineCounter == index) {
207             break;
208         }
209         lineCounter++;
210     }
211 }
212
213 void LinkedList::list(int start, int end) {
214
215
216     auto node = m_start;
217     int counter2 = start;
218     int lineCounter = 1;
219     while (node != nullptr) {
```

```
220         if (lineCounter >= counter2 && lineCounter <= end) {
221             std::cout << counter2 << "> " << node->m_data << " " << "\n";
222             counter2++;
223         }
224         node = node->m_next;
225         lineCounter++;
226 
227         if (lineCounter == end + 1) {
228             break;
229         }
230     }
231 }
232 
233 
234 std::ostream &operator<<(std::ostream &output, LinkedList &list) {
235     auto node = list.m_start;
236     std::ofstream myFileOut;
237     myFileOut.open("test.txt", std::ios::out);
238     while (node != nullptr) {
239         output << node->m_data << " " << "\n";
240         myFileOut << node->m_data << " " << "\n";
241         node = node->m_next;
242     }
243     //myFileOut << list;
244     myFileOut.close();
245     return output;
246 }
247 
248 
249 int LinkedList::printLastNum() {
250     auto node = m_start;
251     LinkedListNode *prev = nullptr;
252     auto counter = 1;
253     while (node != nullptr) {
254         node = node->m_next;
255         counter++;
256         if (node == nullptr) {
257 //            std::cout << counter << "> ";
258             break;
259         }
260     }
261     return counter;
262 }
263 
264 
265 
266 
267 
268 
```

```
 1 //
 2 // Created by mark- on 2023-01-22.
 3 //
 4
 5 #ifndef ASSIGNMENT1_LINKEDLIST_H
 6 #define ASSIGNMENT1_LINKEDLIST_H
 7 #include "LinkedListNode.h"
 8 #include "iostream"
 9
10
11 class LinkedList {
12 private:
13     LinkedListNode *m_start{nullptr};
14     int m_size{0};
15 public:
16     LinkedList();
17     virtual ~LinkedList();
18
19     void add(std::string data);
20     void insert(std::string data, int index);
21     void remove(int index);
22     void remove(int start, int end);
23     void list();
24     void list(int lineNum);
25     void list(int start, int end);
26     int printLastNum();
27     friend std::ostream &operator<<(std::ostream &output, LinkedList &list);
28 };
29
30
31 #endif //ASSIGNMENT1_LINKEDLIST_H
32
```

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

#include "LinkedListNode.h"

```

```
 1 //
 2 // Created by mark- on 2023-01-22.
 3 //
 4
 5 #ifndef ASSIGNMENT1_LINKEDLISTNODE_H
 6 #define ASSIGNMENT1_LINKEDLISTNODE_H
 7 #include <iostream>
 8
 9 class LinkedListNode {
10 public:
11     std::string m_data = "0";
12     LinkedListNode *m_next{nullptr};
13
14 };
15
16
17 #endif //ASSIGNMENT1_LINKEDLISTNODE_H
18
```

```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;
}
```

```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;
    }
}

```

```
1 //
2 // Created by mark- on 2023-01-22.
3 //
4
5 #ifndef ASSIGNMENT1_READFILE_H
6 #define ASSIGNMENT1_READFILE_H
7 #include "LinkedList.h"
8
9
10 class ReadFile {
11
12 public:
13     static LinkedList readfile(std::string argument, LinkedList linkedList);
14
15 };
16
17
18 #endif //ASSIGNMENT1_READFILE_H
19
```

```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


```

```
 1 //
 2 // Created by mark- on 2023-01-22.
 3 //
 4
 5 #ifndef ASSIGNMENT1_TEXTEDITOR_H
 6 #define ASSIGNMENT1_TEXTEDITOR_H
 7 #include "LinkedList.h"
 8
 9
10 class TextEditor {
11
12 public:
13     LinkedList startTextEditor(LinkedList linkedList);
14 };
15
16
17 #endif //ASSIGNMENT1_TEXTEDITOR_H
18
```