

```
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
```

```

1 //
2 // Created by mark- on 2023-01-22.
3 //
4
5 #include "TextEditor.h"
6 #include <iostream>
7 #include <sstream>
8 #include <string>
9 #include "LinkedList.h"
10
11 //int cursorPosition;
12 LinkedList TextEditor::startTextEditor(LinkedList linkedList) {
13     std::string input;
14     linkedList.list();
15     char command;
16     int start, end;
17     int cursorPosition;
18     int count = 0;
19     while (input != "E") {
20         start = '\0';
21
22         if (count == 0) {
23
24             cursorPosition = linkedList.printLastNum();
25             std::cout << cursorPosition << "> ";
26         }
27         getline(std::cin, input);
28
29         std::stringstream ss;
30         std::stringstream ss2;
31         std::stringstream ss3;
32         ss << input;
33         ss2 << input;
34         ss3 << input;
35
36         ss >> command >> start >> end;
37         if (!ss) {
38             // command with start and end not entered
39             ss2 >> command >> start;
40             if (!ss2) {
41                 // command with start not entered
42                 ss3 >> command;
43                 if (!ss3) {
44                     // no commands entered
45                 } else {
46
47                     // just command is entered
48                     if (command == 'L') {
49                         if (count == 0) {
50                             cursorPosition = linkedList.printLastNum();
51                         }
52                         linkedList.list();
53                     }
54                     if (command == 'I') {
55                         if (count == 0 || start == '\0') {
56                             std::string data;
57                             getline(std::cin, data);
58                             linkedList.add(data);
59                             cursorPosition = linkedList.printLastNum();
60                         } else {
61                             std::string data;
62                             getline(std::cin, data);
63                             std::cout << cursorPosition << "> ";
64                             linkedList.insert(data, cursorPosition);
65                         }
66                     }
67                     if (command == 'D') {
68                         if (count == 0) {
69                             linkedList.remove(linkedList.printLastNum());
70                             cursorPosition = linkedList.printLastNum();
71                         } else {
72
73                             linkedList.remove(cursorPosition);

```

```

74         }
75     }
76 }
77 } else {
78     // command and index is entered
79     cursorPosition = start;
80     if (command == 'L') {
81
82         linkedList.list(start);
83     }
84     if (command == 'I') {
85         std::string data;
86         std::cout << cursorPosition << "> ";
87         getline(std::cin, data);
88         linkedList.insert(data, start);
89     }
90     if (command == 'D') {
91         linkedList.remove(start);
92     }
93 }
94 } else {
95     // command with start and end entered
96     cursorPosition = start;
97     if (command == 'L') {
98         cursorPosition = start;
99         linkedList.list(start, end);
100     }
101     if (command == 'D') {
102         cursorPosition = start;
103         linkedList.remove(start, end);
104     }
105 }
106 count++;
107 if (count == 0 || start == '\0') {
108     cursorPosition = linkedList.printLastNum();
109 }
110 std::cout << "\n" << cursorPosition << "> ";
111 } // end while loop
112
113 return linkedList;
114 } // end texteditor function
115
116

```

```
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
```

```
1 //
2 // Created by mark- on 2023-01-22.
3 //
4 #include <iostream>
5
6 #include <string>
7
8 #include <fstream>
9 #include <string>
10 #include <iostream>
11 #include <exception>
12 #include <cstdlib>
13 #include "ReadFile.h"
14 #include "LinkedList.h"
15
16 using namespace std;
17
18 LinkedList ReadFile::readfile(std::string argument, LinkedList linkedList) {
19     try {
20         string line; // declaring string
21         fstream myFileIn; // file in stream reading and writing
22         ofstream myFileOut; // file out stream writing only
23         myFileIn.open(argument, ios::in | ios::out); // original txt file
24         // open for writing
25         if (myFileIn.is_open()) {
26             cout << "File Open" << endl; // confirmation of successful file open
27             while (!myFileIn.eof()) { // continue until end of file
28                 getline(myFileIn, line);
29                 linkedList.add(line);
30             }
31
32             myFileIn.close(); // closing file in stream
33
34             cout << "File closed" << endl;
35             return linkedList;
36         } else {
37             cout << "Input file failed to open. Will make new File on Exit." << endl;
38
39             return linkedList;
40         }
41     } //
42     catch (MyException& e) {
43         //
44         cout << e.error() << endl;
45     } //
46     catch (exception &e) {
47         cout << "Generic error" << endl;
48     }
49     catch (...) {
50         cout << "General error" << endl;
51     }
52 }
53
```

```
1 #include <iostream>
2
3 #include <string>
4
5 #include <fstream>
6 #include <string>
7 #include <iostream>
8 #include <exception>
9 #include <cstdlib>
10 #include "LinkedList.h"
11 #include "ReadFile.h"
12 #include "TextEditor.h"
13
14 using namespace std;
15
16 int main(int argc, char *argv[]) {
17
18     if (argc == 2) {
19         LinkedList linkedList;
20         TextEditor textEditor;
21
22         linkedList = ReadFile::readfile(argv[1], linkedList);
23         linkedList = textEditor.startTextEditor(linkedList);
24         cout << linkedList << endl;
25     }
26     else{
27         cout << "Check Command Line Arguments" << endl;
28     }
29
30
31     return 0;
32 }
```

```
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
```

```
1 //
2 // Created by mark- on 2023-01-22.
3 //
4
5 #include "LinkedListNode.h"
6
```



```
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
```

```
1 //
2 // Created by mark- on 2023-01-22.
3 //
4
5 #include "LinkedList.h"
6 #include <iostream>
7 #include <string>
8 #include <fstream>
9
10 LinkedList::LinkedList() {
11     LinkedListNode *m_start{nullptr};
12     int m_size{0};
13 }
14
15 //LinkedList::~~LinkedList() {
16 //     auto node = m_start;
17 //     while (node != nullptr) {
18 //         auto temp = node;
19 //         node = node->m_next;
20 //         delete temp;
21 //     }
22 //}
23
24 void LinkedList::add(std::string data) {
25     // create a new node
26     auto node = new LinkedListNode();
27     node->m_data = data;
28     if (m_start == nullptr) {
29         // add the first node to the list
30         m_start = node;
31     } else {
32         //add to the end of the list
33         LinkedListNode *current = m_start;
34         LinkedListNode *previous = nullptr;
35
36         //look for the end of the chain
37         while (current != nullptr) {
38             previous = current;
39             current = current->m_next;
40         }
41         //attach the new node
42         previous->m_next = node;
43     }
44     m_size++;
45 }
46
47 void LinkedList::insert(std::string data, int index) {
48
49     if (index > m_size) {
50         return add(data);
51     }
52
53     // create a new node
54     auto node = new LinkedListNode();
55     node->m_data = data;
56
57     //find the index we are inserting before
58     auto current = m_start;
59     LinkedListNode *previous = nullptr;
60
61     auto count{1};
62     while (current != nullptr) {
63         if (count++ == index) {
64             break;
65         }
66         previous = current;
67         current = current->m_next;
68     }
69     // am i inserting at the beginning?
70     if (previous == nullptr) {
71         //insert at the start of the list
72         node->m_next = m_start;
73         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 //          // am i deleting the first node?
166 //          if (prev == nullptr) {
167 //              //first node
168 //              m_start = node->m_next;
169 //          } else {
170 //              //other node
171 //              prev->m_next = node->m_next;
172 //          }
173 //          // finally
174 //          delete node;
175 //      }
176 //      m_size--;
177 //  }
178 //  }
179 //  }
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     auto node = m_start;
216     int counter2 = start;
217     int lineCounter = 1;
218     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
```