

```
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     void add(std::string data);
18     void insert(std::string data, int index);
19     void remove(int index);
20     void remove(int start, int end);
21     void list();
22     void list(int lineNum);
23     void list(int start, int end);
24     int printLastNum();
25     friend std::ostream &operator<<(std::ostream &output, LinkedList &list);
26 };
27
28
29 #endif //ASSIGNMENT1_LINKEDLIST_H
30
```

```
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 void LinkedList::add(std::string data) {
16     // create a new node
17     auto node = new LinkedListNode();
18     node->m_data = data;
19     if (m_start == nullptr) {
20         // add the first node to the list
21         m_start = node;
22     } else {
23         //add to the end of the list
24         LinkedListNode *current = m_start;
25         LinkedListNode *previous = nullptr;
26
27         //look for the end of the chain
28         while (current != nullptr) {
29             previous = current;
30             current = current->m_next;
31         }
32         //attach the new node
33         previous->m_next = node;
34     }
35     m_size++;
36 }
37
38 void LinkedList::insert(std::string data, int index) {
39
40     if (index > m_size) {
41         return add(data);
42     }
43
44     // create a new node
45     auto node = new LinkedListNode();
46     node->m_data = data;
47
48     //find the index we are inserting before
49     auto current = m_start;
50     LinkedListNode *previous = nullptr;
51
52     auto count{1};
53     while (current != nullptr) {
54         if (count++ == index) {
55             break;
56         }
57         previous = current;
58         current = current->m_next;
59     }
60     // am i inserting at the beginning?
61     if (previous == nullptr) {
62         //insert at the start of the list
63         node->m_next = m_start;
64         m_start = node;
65     } else {
66         // inserting in the middle of the list
67         node->m_next = previous->m_next;
68         previous->m_next = node;
69     }
70     m_size++;
71 }
72
73 void LinkedList::remove(int index) {
```

```

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

```

```

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

```

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

```
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_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
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 #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 //
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) {
56                             std::string data;
57                             getline(std::cin, data);
58                             linkedList.insert(data, linkedList.printLastNum());
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) {
108     cursorPosition = linkedList.printLastNum();
109 } else {
110     std::cout << "\n" << cursorPosition << "> ";
111 }
112 } // end while loop
113
114 return linkedList;
115 } // end texteditor function
116
117

```