```
 1 //
 2 // Created by cassa on 2022-01-31.
 3 //
 4
 5 #include <iostream>
 6 #include <sstream>
 7 #include <algorithm>
 8 #include <cctype>
 9 #include <fstream>
10
11 #include "LineBuilder.h"
12 #include "LinkedList.h"
13
14 using namespace std;
15
16
17 LineBuilder::LineBuilder() {m_exit = false,
   line_to_edit = 1;}
18
19 LineBuilder::~LineBuilder() {}
20
21
22 void LineBuilder::create_file(string file) {
23
24     open_file(file);
25     line_to_edit = list.get_size() + 1;
26
27     while (!m_exit){
28         string input;
29         string command;
30
31         cout << line_to_edit << ">";
32         getline(cin, input);
33
34         //Set up stringstream
35         stringstream ss;
36         ss << input;
37         //Get first word as command
38         ss >> command;
39
40         //If first word is longer than 1 character,
```

```
40 can assume it is a not a command -- add it as a new
   line
41          if (command.length() > 1){
42              new_line(input);
43          }
44          else {
45              int start = -1;
46              //ss now contains second "word" -- this
   checks if that word is a number
47              ss >> start;
48              if (ss.fail()) {
49                  ss.clear();
50                  ss >> command;
51                  if (ss.fail()) {
52                      ss.clear();
53                      get_command(command);
54                  } else {
55                      new_line(input);
56                  }
57              } else {
58                  int end = -1;
59                  ss >> end;
60                  if (ss.fail()) {
61                      ss.clear();
62                      get_command(command, start);
63                  }
64                  else {
65                      get_command(command, start, end);
66                  }
67              }
68          }
69      }
70      save_file(file);
71  }
72
73  void LineBuilder::new_line(string data) {
74      if (line_to_edit - 1 == list.get_size()){
75          list.add_node(data);
76      }
77      else {
78          list.insert_node(data, line_to_edit);
```

```
 79     }
 80     line_to_edit++;
 81 }
 82
 83 void LineBuilder::get_command(string command) {
 84     char cmd = command[0];
 85     if (toupper(cmd) == 'L'){
 86         cout << list.output_list(1, list.get_size
    ());
 87         line_to_edit = list.get_size() + 1;
 88     }
 89     else if (toupper(cmd) == 'I'){
 90         line_to_edit --;
 91         if (line_to_edit <= 0){
 92             line_to_edit = 1;
 93         }
 94     }
 95     else if (toupper(cmd) == 'D'){
 96         list.delete_node(line_to_edit - 1);
 97         line_to_edit = list.get_size() + 1;
 98     }
 99     else if (toupper(cmd) == 'E'){
100         m_exit = true;
101     }
102     else {
103         new_line(command);
104     }
105 }
106
107 void LineBuilder::get_command(string command, int
    index) {
108     char cmd = command[0];
109     if (toupper(cmd) == 'L') {
110         if (index > 0 && index <= list.get_size()){
111             cout << list.
112                     output_list(index, index);
113             line_to_edit = list.get_size() + 1;
114         }
115         else {
116             cout << "Index out of range" << endl;
117         }
```

```
118     }
119     else if (toupper(cmd) == 'I') {
120         if (index > 0 && index <= list.get_size()) {
121             line_to_edit = index;
122         }
123         else {
124             cout << "Index out of range" << endl;
125         }
126     }
127     else if (toupper(cmd) == 'D') {
128         if (index > 0 && index <= list.get_size()) {
129             list.delete_node(index);
130         }
131         else {
132             cout << "Index out of range" << endl;
133         }
134     }
135     else if (toupper(cmd) == 'E'){
136         m_exit = true;
137     }
138     else {
139         new_line(command);
140     }
141 }
142 
143 void LineBuilder::get_command(string command, int
    start_index, int end_index){
144     char cmd = command[0];
145     if (toupper(cmd) == 'L') {
146         if (start_index > 0 && end_index <= list.
    get_size()) {
147             cout << list.output_list(start_index,
    end_index);
148             line_to_edit = list.get_size() + 1;
149         }
150         else {
151             cout << "Index out of range" << endl;
152         }
153     }
154     else if (toupper(cmd) == 'D') {
155         if (start_index > 0 && end_index <= list.
```

```
155 get_size()) {
156             for (int i = end_index; i >= start_index
    ; i--) {
157                 list.delete_node(i);
158             }
159         }
160         else {
161             cout << "Index out of range" << endl;
162         }
163     }
164     else if (toupper(cmd) == 'E'){
165         m_exit = true;
166     }
167     else {
168         new_line(command);
169     }
170 }
171
172 void LineBuilder::open_file(string file_name) {
173     ifstream inFile;
174     inFile.open(file_name + ".txt");
175
176     if (!inFile.fail()){
177         string line;
178         while (getline(inFile, line)){
179             list.add_node(line);
180         }
181         inFile.close();
182         cout << list.output_list(1, list.get_size
    ());
183     }
184     else {
185         cout << "No file found, a new one will be
    created upon exiting." << endl;
186     }
187 }
188
189 void LineBuilder::save_file(string file_name) {
190     ofstream outFile;
191     outFile.open(file_name + ".txt", ios::app);
192     if (!outFile.fail()){
```

```
193         outFile << list.output_all_node_data();
194     }
195     else {
196         cout << "File save error" << endl;
197     }
198     outFile.close();
199 }
200 
201 
202 
203 //https://www.geeksforgeeks.org/write-a-function-to-
    get-nth-node-in-a-linked-list/
204 //https://www.cplusplus.com/reference/string/string/
    find/
```

```
 1 //
 2 // Created by cassa on 2022-01-31.
 3 //
 4
 5 #ifndef ASSIGNMENT1_LINEBUILDER_H
 6 #define ASSIGNMENT1_LINEBUILDER_H
 7
 8
 9 #include "LinkedList.h"
10
11 class LineBuilder {
12 private:
13     bool m_exit;
14     int line_to_edit;
15     LinkedList list;
16
17 public:
18     LineBuilder();
19     virtual ~LineBuilder();
20
21     void create_file(string file);
22     void new_line(string data);
23     void get_command(string);
24     void get_command(string, int);
25     void get_command(string, int, int);
26     void open_file(string);
27     void save_file(string);
28 };
29
30
31 #endif //ASSIGNMENT1_LINEBUILDER_H
32
```

```
 1 //
 2 // Created by cassa on 2022-01-31.
 3 //
 4 #include <iostream>
 5 #include <sstream>
 6 #include <algorithm>
 7
 8 #include "LinkedList.h"
 9 #include "LinkedListNode.h"
10
11 LinkedList::LinkedList() : m_start(nullptr), m_size(0
   ) {}
12
13 LinkedList::~LinkedList() {
14     LinkedListNode *node = m_start;
15     while (node != nullptr) {
16         LinkedListNode *temp = node;
17         node = node->m_next;
18         delete temp;
19     }
20 }
21
22 void LinkedList::add_node(string data) {
23     //Create a new node (line of text)
24     LinkedListNode* new_node = new LinkedListNode();
25     new_node->m_data = data;
26
27     //Add first node
28     if (m_start == nullptr){
29         m_start = new_node;
30     }
31         //Add node to end of existing chain
32     else {
33         LinkedListNode* node = m_start;
34         LinkedListNode* prev = nullptr;
35
36         //Look for the node with a null "next"
   pointer
37         while (node != nullptr){
38             prev = node;
39             node = node->m_next;
```

```
40          }
41
42          //Attach new node to end of the chain
43          if (prev != nullptr){
44              prev->m_next = new_node;
45          }
46      }
47      m_size++;
48  }
49
50  void LinkedList::insert_node(string data, int
    position) {
51
52      // check to see at least one node to insert
    before
53      if (position > m_size) {
54          cout << "Node to insert before doesn't exist
    ." << endl;
55          return;
56      }
57
58      LinkedListNode* new_node = new LinkedListNode();
59      if (new_node == nullptr) {
60          cout << "Couldn't allocate memory for new
    nodes." << endl;
61          return;
62      }
63      new_node->m_data = data;
64
65      // find position
66      LinkedListNode* node = m_start;
67      LinkedListNode* prev = nullptr;
68      int curr_pos = 1;
69
70      while (node != nullptr) {
71
72          // application specific - use position to
    find node
73          if (curr_pos == position) {
74              break;
75          }
```

```
 76          curr_pos++;
 77          prev = node;
 78          node = node->m_next;
 79      }
 80
 81      if (prev == nullptr) {
 82          // insert node at the start
 83          new_node->m_next = m_start;
 84          m_start = new_node;
 85      } else {
 86          // insert node in the middle
 87          new_node->m_next = prev->m_next;
 88          prev->m_next = new_node;
 89      }
 90
 91      m_size++;
 92  }
 93
 94  bool LinkedList::delete_node(int index) {
 95      LinkedListNode* node = m_start;
 96      LinkedListNode* prev = nullptr;
 97
 98      string data = get_data_by_index(index);
 99
100      while (node != nullptr){
101          //Logic ot find code
102          if (node->m_data == data){
103              break;
104          }
105          prev = node;
106          node = node->m_next;
107      }
108
109      //If node was found, would have value, otherwise
     would reach end of chain
110      if (node != nullptr){
111          //Check if deleting first node
112          if (prev == nullptr){
113              m_start = node->m_next;
114          }
115          //Otherwise deleting any other node
```

```
116         else {
117             prev->m_next = node->m_next;
118         }
119         delete node;
120         m_size--;
121         return true;
122     }
123     return false;
124 }
125 
126 ostream& operator<<(ostream& output, LinkedList&
    list){
127     LinkedListNode* node = list.m_start;
128     int lineCount = 0;
129     //Output data from all the nodes in the chain
130     while (node != nullptr){
131         lineCount++;
132         cout << lineCount << ">" << node->m_data <<
    endl;
133         node = node->m_next;
134     }
135     return output;
136 }
137 
138 string LinkedList::get_data_by_index(int index) {
139     LinkedListNode* node = m_start;
140     int count = 1;
141     while (node != nullptr){
142         if (count == index){
143             return node->m_data;
144         }
145         count++;
146         node = node->m_next;
147     }
148     //return "Error -- Line non-existent";
149 }
150 
151 int LinkedList::get_size() {
152     return m_size;
153 }
154 
```

```
155 string LinkedList::output_list(int start_index, int
    end_index) {
156     stringstream output;
157     for (int i = start_index; i <= end_index; i++){
158         string data = get_data_by_index(i);
159         output << i << ">" << data << endl;
160     }
161     return output.str();
162 }
163
164 string LinkedList::output_all_node_data() {
165     stringstream output;
166     for (int i = 1; i <= m_size; i++){
167         string data = get_data_by_index(i);
168         output << data << endl;
169     }
170     return output.str();
171 }
172
```

```
 1 //
 2 // Created by cassa on 2022-01-31.
 3 //
 4 #include <string>
 5
 6 #ifndef ASSIGNMENT1_LINKEDLIST_H
 7 #define ASSIGNMENT1_LINKEDLIST_H
 8
 9 using namespace std;
10
11 class LinkedListNode;
12
13 class LinkedList {
14 private:
15     int m_size;     //size of the current list
16     LinkedListNode* m_start;    //pointer to the
   start of the first node in the chain
17
18 public:
19     LinkedList();
20     virtual ~LinkedList();
21     void add_node(string);
22     void insert_node(string, int);
23     bool delete_node(int);
24     int get_size();
25     string get_data_by_index(int);
26     string output_list(int, int);
27     string output_all_node_data();
28     friend ostream& operator<<(ostream& output,
   LinkedList& list);
29 };
30
31
32
33 #endif //ASSIGNMENT1_LINKEDLIST_H
34
```

```
1 //
2 // Created by cassa on 2022-01-31.
3 //
4
5 #include "LinkedListNode.h"
6
7 LinkedListNode::LinkedListNode() : m_data(""), m_next
  (nullptr) {}
```

```
 1 //
 2 // Created by cassa on 2022-01-31.
 3 //
 4 #include <string>
 5 #ifndef ASSIGNMENT1_LINKEDLISTNODE_H
 6 #define ASSIGNMENT1_LINKEDLISTNODE_H
 7
 8 using namespace std;
 9
10 class LinkedListNode {
11 public:
12     string m_data;     //the data
13     LinkedListNode* m_next;     //pointer to the
   address in memory of the next line of text
14     LinkedListNode();
15 };
16
17
18 #endif //ASSIGNMENT1_LINKEDLISTNODE_H
19
```

```cpp
#include <iostream>
#include <sstream>
#include <algorithm>

#include "LineBuilder.h"
#include "LinkedList.h"
using namespace std;

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

    LineBuilder lineBuilder;

    if (argc == 3 && string(argv[1]) == "EDIT")
    {
        lineBuilder.create_file(string(argv[2]));
    }
    else
    {
        cout << "Invalid command, must begin with EDIT followed by file name." << endl;
    }

    return 0;
}

//First word in string: https://stackoverflow.com/questions/41992747/how-to-only-print-the-first-word-of-a-string-in-c/41992882
//Remove whitesapces: https://www.techiedelight.com/remove-whitespaces-string-cpp/
```