```
 1 //
 2 // Created by nick on 9/19/2023.
 3 //
 4
 5 #ifndef ASSIGNMENT1_UTIL_H
 6 #define ASSIGNMENT1_UTIL_H
 7 #include <iostream>
 8 #include <regex>
 9 #include "LinkedList.h"
10
11
12
13 class Util {
14
15
16 public:
17
18     //main function
19     static void line_editor(LinkedList& list);
20
21
22     //util functions
23     static int convert_num(char input);
24     static void display_line(LinkedList& list);
25     static void get_commands(std::vector<char>& commands, std::string
   & input, LinkedList& list);
26     static std::string get_line_input(int line_num);
27     static void insert_vs_add(LinkedList& list, std::string& data);
28     static void scan_input(std::string& input, std::vector<char>&
   commands);
29     static bool validate_name(std::string& file_name);
30 };
31
32 #endif //ASSIGNMENT1_UTIL_H
33
```

```cpp
#include <iostream>
#include "LinkedList.h"
#include "Util.h"

using  std::cout;
using  std::endl;
using  std::string;

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

    //check for the right number of commands
    if (argc != 2)
    {
        cout << "Make sure to include your file name e.g: ./assignment1 test.txt" << endl;
        return -1;
    }

    //grab the filename and make sure it's in a valid format, if its valid begin filling the list
    string file_name = argv[1];
    bool valid_arguments = Util::validate_name(file_name);
    LinkedList main_list;

    if (valid_arguments)
    {
         LinkedList::create_list(file_name, main_list);
    } else
    {
        cout << "Invalid file format. Example file format: file_name.txt" << endl;
        return -1;
    }

    //enter the line editor
    Util::line_editor(main_list);

    //once user has exited the line editor, write the list to a file and exit
    LinkedList::save(file_name, main_list);

    return 0;
}

```

```cpp
//
// Created by nick on 9/19/2023.
//
#include "Util.h"
using  std::cin;
using  std::cout;
using  std::endl;
using  std::regex_match;
using  std::regex;
using  std::string;

//main function
void   Util::line_editor(LinkedList &list)
{
    string            input;
    std::vector<char> commands;

    //display the current list if there is any
    cout << list;
    bool exit = false;
    do
    {

        //display the current line and prompt for input
        Util::get_commands(commands, input, list);




        //check if the input was a command or just text input
        if (commands.size() > 3 || commands.size() < 1)
        {
            insert_vs_add(list, input);
        } else
        {
            //check first command parameter and proceed to
corresponding process
            switch (tolower(commands[0]))
            {
                case 'i':
                    list.insert_process(commands, input);
                    list.set_insert_mode(true);
                    break;
                case 'd':
                    list.remove_process(commands);
                    list.set_insert_mode(false);
                    break;
                case 'l':
                    list.display_process(commands);
                    list.set_insert_mode(false);
                    break;
                case 'e':
                    exit = true;
```

```
 53                      break;
 54                  default:
 55                      insert_vs_add(list, input);
 56                      break;
 57              }
 58          }
 59      } while (!exit);
 60  }
 61
 62
 63  //util functions
 64  int     Util::convert_num(char input)
 65  {
 66      return isdigit(input) ? input - '0' : -1;
 67  }
 68
 69  void    Util::display_line(LinkedList& list)
 70  {
 71      cout << list.get_current_num() << ">";
 72  }
 73
 74  void    Util::get_commands(std::vector<char>& commands, string& input
     , LinkedList& list)
 75  {
 76
 77      //prompt for input, then scan the string to create the commands
     vector
 78      commands.clear();
 79
 80      cout << list.get_current_num() << ">";
 81      std::getline(cin, input);
 82      cin.clear();
 83
 84      scan_input(input, commands);
 85  }
 86
 87  string Util::get_line_input(int line_num)
 88  {
 89      string input;
 90
 91      cout << line_num << ">";
 92      std::getline(cin, input);
 93
 94
 95      return input;
 96  }
 97
 98  void    Util::insert_vs_add(LinkedList& list, string& data)
 99  {
100      //if we're in insert mode, insert, else add to the end of the
     list.
101      if (list.get_insert_mode())
102      {
```

```
103         list.insert(list.get_current_num(), data);
104     } else
105     {
106         list.add(data);
107     }
108 }
109
110 void   Util::scan_input(string& input, std::vector<char>& commands)
111 {
112     //ignores whitespace and adds every character of the input to the
     commands vector
113     for (char character : input)
114     {
115         if (std::isalnum(character))
116         {
117             commands.push_back(character);
118         }
119     }
120
121 }
122
123 bool   Util::validate_name(string& file_name)
124 {
125     regex   regex_mask("[^/]*\\.txt$");
126     return  regex_match(file_name, regex_mask);
127 }
128
```

```
 1 //
 2 // Created by nick on 9/19/2023.
 3 //
 4
 5 #ifndef ASSIGNMENT1_LINKEDLIST_H
 6 #define ASSIGNMENT1_LINKEDLIST_H
 7 #include <iostream>
 8 #include <fstream>
 9 #include <vector>
10
11
12 class LinkedList {
13 private:
14     struct Node {
15         //give the node data and a link to the next node
16         std::string data{"null"};
17         int line_number {-1};
18         Node *next_node{nullptr};
19     };
20
21     bool insert_mode = false;
22     //will be given to the first node in the list
23     Node *starting_address    = nullptr;
24     Node *null__node_ptr      = nullptr;
25     int  starting_num         = 1;
26     int  current_num          = 1;
27
28 public:
29
30
31     //getters / setters
32     int get_current_num() const{return current_num;}
33     void set_insert_mode(bool input){insert_mode = input;}
34     bool get_insert_mode() const {return insert_mode;}
35     Node* get_starting_address(){return starting_address;}
36
37
38     //Operational methods
39     void add(std::string& data);
40     static void create_list(std::string& file_name, LinkedList &list);
41     void display_process(std::vector<char> commands);
42     void display(int display_line);
43     void display(int display_start, int display_end);
44     void insert_process(std::vector<char> commands, std::string& input
   );
45     void insert();
46     void insert(int insert_line, std::string& input);
47     void remove_process(std::vector<char> commands);
48     void remove(int line_to_delete);
49     void remove(int delete_start, int delete_end);
50     static void save(std::string& file_name, LinkedList& list);
51
52
```

```
53      //functional methods
54      int  assign_num(std::string& data);
55      void calibrate_list();
56      bool line_exists(int line);
57      int  size();
58
59
60      //operator and destructor
61      virtual ~LinkedList();
62      friend std::ostream &operator<<(std::ostream &output, LinkedList
    &list);
63  };
64
65
66  #endif //ASSIGNMENT1_LINKEDLIST_H
67
```

```cpp
//
// Created by nick on 9/19/2023.
//

#include "LinkedList.h"
#include "Util.h"
using  std::cout;
using  std::endl;
using  std::string;


//operational methods
void LinkedList::add(std::string& data)
{
    //create node, add the data and give it an address
    auto node  = new Node();
    node->data = data;


    //if the list is empty, place the new node at the start
    if (starting_address == nullptr)
    {
        starting_address  = node;
        node->line_number = starting_num;
        current_num++;
    } else
    {
        //find the end of the list and add the node
        auto current_node  = starting_address;
        auto previous_node = (Node *) nullptr;

        while (current_node != nullptr)
        {
            previous_node = current_node;
            current_node  = current_node->next_node;
        }
        previous_node->next_node = node;
        node->line_number        = current_num;
        current_num++;
    }
}

void LinkedList::create_list(string& file_name, LinkedList& list)
{

    //attempt to open file
    try
    {
        std::ifstream input_stream(file_name);

        if (!input_stream.fail())
        {
            //if file is found read in each line and add them to the
```

```
 53 list
 54             string in_line;
 55             while (!input_stream.eof())
 56             {
 57                 getline(input_stream, in_line);
 58                 list.add(in_line);
 59             }
 60         } else
 61         {
 62             //if no file is found, return an empty list
 63             cout << "New File!" << endl;
 64         }
 65 
 66         input_stream.close();
 67     }
 68     catch (std::ios_base::failure& failure)
 69     {
 70         cout << failure.what() << endl;
 71     }
 72 
 73 }
 74 
 75 void LinkedList::display_process(std::vector<char> commands)
 76 {
 77     switch (commands.size())
 78     {
 79         case 1:
 80             //just print the full list
 81             cout << *this;
 82             break;
 83         case 2:
 84             //display specific line
 85             display(Util::convert_num(commands[1]));
 86             break;
 87         case 3:
 88             display(Util::convert_num(commands[1]), Util::convert_num
    (commands[2]));
 89             break;
 90         default:
 91             cout << endl << "Invalid number of commands" << endl;
 92             break;
 93     }
 94     //set current num to be one after the end of the list
 95     calibrate_list();
 96 }
 97 
 98 void LinkedList::display(int display_line)
 99 {
100     auto current_node = starting_address;
101 
102     if (line_exists(display_line))
103     {
104         //search for the specific line and output it
```

```
105         while (current_node->line_number != display_line)
106         {
107             current_node = current_node->next_node;
108         }
109 
110         cout << current_node->line_number << '>' << current_node->
    data << endl;
111     }
112 }
113 
114 void LinkedList::display(int start_num, int end_num)
115 {
116     //display the nodes starting from start_num and ending on end_num
117     auto current_node = starting_address;
118 
119     if (line_exists(start_num) && line_exists(end_num))
120     {
121         while (current_node->line_number != start_num)
122         {
123             current_node = current_node->next_node;
124         }
125 
126         if (current_node != nullptr)
127         {
128             while (current_node != nullptr)
129             {
130                 if (current_node->line_number > end_num)
131                 {
132                     break;
133                 }
134                 cout << current_node->line_number << ">" <<
    current_node->data << endl;
135                 current_node = current_node->next_node;
136             }
137         }
138     }
139 
140 }
141 
142 void LinkedList::insert_process(std::vector<char> commands, string&
    input)
143 {
144     //only enter insert mode if the commands are correct
145     switch (commands.size())
146     {
147         case 1:
148             //insert on current line
149             insert();
150             insert_mode = true;
151             break;
152         case 2:
153             //insert of specific line
154             insert(Util::convert_num(commands[1]), input);
```

```
155             insert_mode = true;
156             break;
157         default:
158             cout << "insert cannot be called on a range, must be
    called on individual line number" << endl;
159             break;
160     }
161 }
162
163 void LinkedList::insert(int insert_line, string& input)
164 {
165     if (line_exists(insert_line))
166     {
167         //find the node we want to insert before
168         auto node          = new Node;
169         auto current_node  = starting_address;
170         auto previous_node = null__node_ptr;
171
172         while (current_node != nullptr)
173         {
174             if (current_node->line_number == insert_line)
175             {
176                 break;
177             }
178             previous_node = current_node;
179             current_node  = current_node->next_node;
180         }
181
182
183         //if we're in insert mode we will already have input
184         if (!insert_mode)
185         {
186             input = Util::get_line_input(current_node->line_number);
187         }
188
189
190         //if we are inserting in the last node, just add it instead.
191         if (current_node == nullptr)
192         {
193             add(input);
194         } else
195         {
196             //insert at beginning of list
197             if (previous_node == nullptr)
198             {
199                 node->next_node   = current_node;
200                 node->data        = input;
201                 node->line_number = starting_num;
202                 current_node->line_number++;
203                 starting_address  = node;
204             } else
205             {
206                 //insert in the middle of the list
```

```
207                 previous_node->next_node  = node;
208                 node->next_node           = current_node;
209                 node->data                = input;
210                 node->line_number         = assign_num(node->data);
211                 current_node->line_number = node->line_number + 1;
212             }
213 
214             //adjust the current line number to be one more than the
    last entered lines number
215             current_num = node->line_number+1;
216         }
217     }
218 }
219 
220 void LinkedList::insert()
221 {
222     //base insert just acts like add with one extra step
223     string input;
224     Util::display_line(*this);
225     std::getline(std::cin, input);
226     this->add(input);
227 }
228 
229 void LinkedList::remove_process(std::vector<char> commands)
230 {
231 
232     switch (commands.size())
233     {
234         case 1:
235             //remove current line
236             remove(this->size());
237             break;
238         case 2:
239             //remove on specific line
240             remove(Util::convert_num(commands[1]));
241             break;
242         case 3:
243             //remove on range
244             remove(Util::convert_num(commands[1]), Util::convert_num(
    commands[2]));
245             break;
246         default:
247             cout << "only two commands max" << endl;
248             break;
249     }
250 
251     //adjust the line numbers of each node to reflect the removal
252     calibrate_list();
253 }
254 
255 void LinkedList::remove(int line_to_delete)
256 {
257     auto current_node = starting_address;
```

```
258     auto previous_node = null__node_ptr;
259 
260     if (line_exists(line_to_delete))
261     {
262         //find the line we want to delete
263         while (current_node != nullptr)
264         {
265             if (current_node->line_number == line_to_delete)
266             {
267                 break;
268             }
269 
270             previous_node = current_node;
271             current_node = current_node->next_node;
272         }
273 
274 
275         //if it exists delink its references from the list and delete the node.
276         if (current_node != nullptr)
277         {
278             if (current_node == starting_address)
279             {
280                 starting_address = current_node->next_node;
281             }else
282             {
283                 previous_node->next_node = current_node->next_node;
284             }
285 
286             delete current_node;
287             current_num--;
288         }
289 
290     }
291 
292 
293 
294 }
295 
296 void LinkedList::remove(int delete_start, int delete_end)
297 {
298 
299     if (line_exists(delete_start) && line_exists(delete_end))
300     {
301         auto current_node = starting_address;
302         auto previous_node = null__node_ptr;
303 
304         //if the user inputs 4 2 flip the start and end variables
305         if (delete_start > delete_end)
306         {
307             int temp     = delete_start;
308             delete_start = delete_end;
309             delete_end   = temp;
```

```
310         }
311
312         //find the line to start the delete on
313         while (current_node != nullptr)
314         {
315             if (current_node->line_number == delete_start)
316             {
317                 break;
318             }
319
320             if (previous_node == null__node_ptr)
321             {
322                 previous_node = current_node;
323             }
324             else
325             {
326                 previous_node->next_node = current_node;
327             }
328
329             current_node = current_node->next_node;
330         }
331
332         if (current_node != nullptr)
333         {
334             //once the start is found, delete from that node to the
    ending node
335             while (current_node != nullptr)
336             {
337
338                 if (current_node->line_number > delete_end)
339                 {
340                     break;
341                 }
342
343                 //grab the next node before we delete the current
    node
344                 auto next_node_var = current_node->next_node;
345
346                 if (current_node == starting_address)
347                 {
348                     starting_address = next_node_var;
349                     delete current_node;
350                 } else
351                 {
352                     previous_node->next_node = current_node->
    next_node;
353                     delete current_node;
354                 }
355
356                 current_num--;
357                 current_node = next_node_var;
358             }
359         }
```

```
360     }
361 
362 }
363 
364 void LinkedList::save(string& file_name, LinkedList& list)
365 {
366     try
367     {
368         //output_file will be written to, input is used to avoid
    duplicating the text file
369         std::ofstream output_file(file_name);
370         std::ifstream input_file(file_name);
371 
372         if (!output_file.fail())
373         {
374             string current_line;
375             auto current_node = list.get_starting_address();
376             char white_space  = '\n';
377 
378             while (current_node != nullptr)
379             {
380                 //write the contents of the list to the file
381                 std::getline(input_file,  current_line);
382                 if (current_node->data != current_line)
383                 {
384                     output_file << current_node->data;
385                     if (current_node->next_node != nullptr)
386                     {
387                         output_file << white_space;
388                     }
389                 }
390                 current_node = current_node->next_node;
391             }
392         } else
393         {
394             //if the file fails to open alert the user.
395             cout << "Error writing to file: " << file_name << endl;
396         }
397 
398         //close the files
399         output_file.close();
400         input_file.close();
401     }
402     catch(std::ios_base::failure& fail)
403     {
404 
405         cout << "Error closing: " << file_name;
406         cout << fail.what();
407     }
408 }
409 
410 
411 //functional methods
```

```
412 int LinkedList::assign_num(string& data)
413 {
414     auto current_node = starting_address;
415     int counter = 0;
416 
417     while (current_node != nullptr)
418     {
419         counter++;
420         if (current_node->data == data)
421         {
422             return counter;
423         }
424 
425         current_node = current_node->next_node;
426     }
427 
428     return -1;
429 }
430 
431 void LinkedList::calibrate_list()
432 {
433     int counter = 0;
434     auto current_node = starting_address;
435 
436     //make sure each node has the correct line number and also adjust
     the next line number to display to the user
437     while (current_node != nullptr)
438     {
439         counter++;
440         current_node->line_number = counter;
441         current_node = current_node->next_node;
442     }
443 
444     current_num = (counter + 1);
445 }
446 
447 bool LinkedList::line_exists(int line)
448 {
449     //if the line is within the range of our list or if its insert
    mode and we're inserting on the current line, return true
450     if ((line > 0 && line <= this->size()) || (insert_mode && line
     == current_num))
451     {
452         return true;
453     }
454 
455 
456     cout  << "Line: " << line << " doesn't exist" << endl << endl;
457     return false;
458 }
459 
460 int LinkedList::size()
461 {
```

```
462     //increment counter for every node in the list.
463     int counter = 0;
464     auto current_node  = starting_address;
465 
466     while (current_node != nullptr)
467     {
468         counter++;
469         current_node  = current_node->next_node;
470     }
471 
472     return counter;
473 }
474 
475 
476 //operator and destructor
477 std::ostream& operator<<(std::ostream& output, LinkedList& list)
478 {
479 
480     auto current_node = list.starting_address;
481     char line_marker = '>';
482 
483     //display every line of the list
484     while (current_node != nullptr)
485     {
486         output << current_node->line_number << line_marker <<
    current_node->data << endl;
487         current_node = current_node->next_node;
488     }
489 
490     list.current_num = list.size() + 1;
491 
492     cout << endl;
493     return output;
494 }
495 
496 LinkedList::~LinkedList()
497 {
498     //loop from start-end of list, giving the address of the current
    node to temp, moving to the next node and deleting temp
499     auto node = starting_address;
500 
501     while (node != nullptr)
502     {
503         auto temp = node;
504         node = node->next_node;
505         delete temp;
506     }
507 }
508 
```