```
#include <iostream>
// node objects are the data for a List object
class Node {
public:
   std::string value = "";
   Node* next = NULL;
};

class List{
public:
   Node* head = new Node();

   List(){
   }

   void print();    // prints every node value and its index (starting with head = index 1)
   int calculateSize();    // returns an int of the size of the list (how many nodes are linked)
   bool insertNode(int index, std::string input);    // inserts a string to the index specified
   bool deleteNode(int index);    // deletes a node at the index specified
   void insertEnd(std::string valueInput);    // inserts a node at the end of the list
   bool edit (int index, std::string valueInput); // replaces a current node with a new one
(insertion & deletion)
   int search (std::string input); // returns the index of the line that contains the string provided
   };

void List::print(){
   Node *node = this->head;
   int size = calculateSize();

   if(head->value == ""){
      std::cout << "" << std::endl;
      return;
   }

   for(int i = 1; i <= size; i++){
      std::cout << i << " " << node->value << std::endl;
      node = node->next;
   }
}

int List::calculateSize(){
   Node *node = this->head;
```

```
    int counter = 0;
    while(node != NULL){
        counter++;
        node = node->next;
    }
    return counter;
}


bool List::insertNode(int index, std::string valueInput) {
    Node* prev = new Node();
    Node* temp = new Node();
    Node* current  = this->head;

    // if this item is the first in the list


    int size = calculateSize();

    // Can not add to the list. Proceeds if inserting at 0 for the first node of the list b/c head is
inherent
    if(index > size + 1){
        return false;
    }

    // insert at end
    if(index == size + 1){
        insertEnd(valueInput);
        return true;
    }

    if(index == 1){
        if(head->value == ""){
            head->value = valueInput;
            return true;
        }

        temp->value = valueInput;

        temp->next = head;
        head = temp;
        return true;
    }
```

```
    else{
        for(int i = 1; i < index; i++){
            prev = current;

            if(current == NULL)
                return false;

            current = current->next;
        }
        prev->next  = temp;
        temp->value = valueInput;
        temp->next  = current;
    }

    return true;
}


bool List::deleteNode(int index) {

    Node* head = this->head;
    bool deleted = false;
    //special case if the desired value is at the head
    if(index == 1){
        head = head->next;
        deleted = true;
    }

    //keep track of a previous node so deletion is easier
    Node* prev = head;
    Node* node = head->next;

    //iterate through the list and making changes when theValue is found. Continue until at end of
list (node == NULL)
    int i = 2; // starts at 1 because head(i=0) has already been checked
    while(node != NULL){

        if(i == index){
            prev->next = node->next;
            return true;
        }
        else
            prev = node;
```

```
        i++;
        node = node->next;
    }
    return deleted;

}

void List::insertEnd(std::string valueInput){
    Node* node = head;
    Node* insert = new Node();
    insert->value = valueInput;

    if(head->value == ""){
     //this is the first element
        head = insert;
        return;
    }

    //finding the last node
    while(node->next != NULL){
        node = node->next;
    }

    // checking that the size is within constraint
    if((insert->value).length() < 80)
        node->next = insert;

}

bool List::edit (int index, std::string valueInput){
    if(head->value == ""){
        std::cout << "ERROR: missing node to edit" << std::endl;
        return false;
    }

    insertNode(index, valueInput);
    // delete old node
    deleteNode(index+1);
    // insert new node
    return true;
}
```

```
int List::search (std::string input){
    //std::cout << "inside search" << std::endl;
    int i = 1; // index counter that will be returned at the end
    int found = 0; // 0 if none found, > 0 if

    Node *node = this->head;
    while(node != NULL) { // transverse thru nodes
        // if string is found in the value, return i
        if(node->value.find(input) != std::string::npos){
            std::cout << i << " " << node->value << std::endl;
            found++;
        }
        node = node->next;
        i++;
    }
    if(found == 0)
        std::cout << "not found" << std::endl;
    return found;    // arbitrary
}


//will return the first word found in given string
std::string findWord(std::string sentence){
    std::string word = "";
    int i = 0;
    while(i < sentence.length()){
        if(sentence.at(i) == ' ')
            if(i == 0)
                i++;
            else
                return word;
        if(sentence.at(i) != '"')
            word += sentence.at(i);
        else // quotation is found
            return sentence.substr(i+1,sentence.length() - (i + 2)); // want to take the next chunk of
string, excluding the "
        i++;
    }
    return word;
}

int main()
{
```

```
bool quit = false;
std::string input;
List *list = new List();
while(!quit){
    //your code to invoke line editor here
    std::string input;
    getline(std::cin, input);
    //std::cout << input << std::endl;

    std::string command = findWord(input);
    input = input.substr(command.length());
    //std::cout << input << std::endl;

    //std::cout << "lineNum is : " << lineNum << std::endl;
    //std::cout << "quote is : " << quote << std::endl;


    if(command == "quit"){
        quit = true;
        //std::cout << "quit" << std::endl;
    }
    else if(command == "print"){
        list->print();
    }
     else if(command == "insert"){
         // find line number
         std::string lineNumStr = findWord(input);
         int lineNum = atoi(lineNumStr.c_str());
         // find the quote
         input = input.substr(lineNumStr.length() + 1);
         std::string quote = findWord(input);
         if(quote.length() < 80)
            list->insertNode(lineNum, quote);
    }
     else if(command == "insertEnd"){
        // find the quote
         std::string quote = findWord(input);
         list->insertEnd(quote);
    }
     else if(command == "delete"){
         std::string lineNumStr = findWord(input);
         int lineNum = atoi(lineNumStr.c_str());
        list->deleteNode(lineNum);
```

```
        }
        else if(command == "edit"){
            // find line number
             std::string lineNumStr = findWord(input);
             int lineNum = atoi(lineNumStr.c_str());
             // find the quote
             input = input.substr(lineNumStr.length() + 1);
             std::string quote = findWord(input);

            list->edit(lineNum, quote);
        }
        else if(command == "search"){
            std::string quote = findWord(input);
            list->search(quote);
        }
        else
            std::cout << "ERROR: invalid input" << std::endl;
    }
}
```