Submission

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
6283978	11:24:21	Sim	✖ **Run Time Error**	0.02 s	C++
	✔✔✔✔✔✔✔✔✔✔✔✔✖☐☐☐☐				

Test Groups

SAMPLE	ACCEPTED (5/100)

SECRET	RUN TIME ERROR (55/100)

GROUP 1	ACCEPTED (25/100)
GROUP 2	ACCEPTED (5/100)
GROUP 3	ACCEPTED (25/100)
GROUP 4	RUN TIME ERROR (0/100)

Submission contains 1 file: download zip archive

FILENAME	FILESIZE	SHA-1 SUM	
sim.cpp	5878 bytes	162ae624c3b67bd2715aa9b1ddaef41819dcd47f	Help download

Edit and resubmit this submission.

sim.cpp

```
 1 //class definition:
 2
 3 #include <deque>
 4 #include <list>
 5 #include <string>
 6 #include <iostream>
 7 using namespace std;
 8
 9 class Line {
10    public:
11     Line(string& s);  //construct the line
12     ~Line();
13     void print();  //start from head, go to tail.
14     void buildOutputCharPtr();
15     void add(char& c);                 //add char to current element
16     void backspace();                  //does backspace on current que.
17     void front();                      //put cursor to back
18     void back();                       //put cursor to front
19     int strcomp(char* s, int length);  //compare content to string
20     int strcomp(string s);
21
22    private:
23     list<deque<char>> nodeList;
24     list<deque<char>>::iterator currentElement;  //set to first element by default
25     int cursorIsAtFront = 0;                     //either at front or at back
26     char* characters;                            //to be built
27     int size = 0;
28     int printWasCalled = 0;
29 };
30
31 //test
32 void kattis() {
33     string tests;
34     string test;
35     getline(cin, tests);
36     Line* line;
37     int testNum = stoi(tests);
38     for (int testIndex = 0; testIndex < testNum; testIndex++) {
39         getline(cin, test);
40         line = new Line(test);  //    Line* list = new Line(input);
41         line->print();
42         if (testIndex < (testNum - 1)) {
43             cout << endl;
44         }
45         delete line;
46         line = nullptr;
47     }
48 }
49 int main() {
50     kattis();
```

```
51      return 0;
52  }
53
54  Line::Line(string& s) {
55      this->nodeList.push_back(deque<char>());        //push empty deque to serve as
    beginnig and tail.
56      this->currentElement = this->nodeList.begin();  //assign current element to
    first one.
57      for (char& c : s) {
58          if (c == '<') {  //do a backspace
59              this->backspace();
60          } else if (c == '[') {  //move cursor to front
61              this->front();
62          } else if (c == ']') {  //move cursor to back
63              this->back();
64          } else {  //if it is any other character, add it to the line
65              this->add(c);
66          }
67      }
68  }
69  Line::~Line() {
70      if (this->printWasCalled) {
71          delete[] this->characters;
72          this->characters = nullptr;
73      }
74  }
75
76  //add char to current element
77  void Line::add(char& c) {
78      this->currentElement->push_back(c);  //add new character to iterator
79      this->cursorIsAtFront = 0;           //cursor isn't at front anymore.
80      this->size++;
81  }
82  //does backspace on deque in current node
83  void Line::backspace() {
84      if (this->currentElement->size() > 0) {  //current element doesn't have empty
    deque
85          this->currentElement->pop_back();    //destroy last element in deque
86          this->size--;
87      } else if (!this->cursorIsAtFront && this->nodeList.size() > 1) {  //deque empty
    but cursor at back.
88          this->nodeList.pop_back();                                     //destroy
    tail of list.
89          this->currentElement = this->nodeList.end();                   //assign
    current element to new end.
90          this->backspace();                                             //call
    backspace with new, hopefully non empty element at end.
91      }
92      //if currently at front & current element is empty, don't do anything.
93  }
94
95  //print items
96  //TODO: check if was built already.
97  void Line::print() {  //start from head, go to tail.
```

```
    //list<deque<char>>::iterator node;
    //deque<char>::iterator character;
    if (!this->printWasCalled) {     //char* characters was not built yet
        this->buildOutputCharPtr();  //build it.
    }
    cout << this->characters;
}
void Line::buildOutputCharPtr() {                //build char pointer containing output string
    this->characters = new char[this->size + 1];  //one extra space for the null terminator
    int charIndex = 0;
    for (deque<char> node : this->nodeList) {  //node = this->nodeList.begin(); node != this->nodeList.end(); node++) {//iterate through linked list
        for (char character : node) {          // = node->begin(); character != node->end(); character++) {//iterate through deque
            this->characters[charIndex] = character;
            charIndex++;
        }
    }
    this->characters[this->size] = '\0';  //last entry is null terminator
    this->printWasCalled = 1;             //change to characters were built
}
//compare content to string
int Line::strcomp(char* s, int length) {  //compare content to string
    if (length != this->size || !this->printWasCalled) {
        return 0;
    }
    for (int i = 0; i < length; i++) {
        if (this->characters[i] != s[i]) {
            return 0;
        }
    }
    return 1;
}
int Line::strcomp(string s) {
    int length = s.size();
    if (length != this->size || !this->printWasCalled) {
        return 0;
    }
    for (int i = 0; i < length; i++) {
        if (this->characters[i] != s[i]) {
            return 0;
        }
    }
    return 1;
}

//put cursor in front
//change iterator to beginning of list, put flag to not at front
void Line::front() {
    if (this->nodeList.begin()->size()) {          //if element at front not empty
        this->nodeList.push_front(deque<char>());  //create new empty element at front            }
```

```
147     }
148     this->currentElement = this->nodeList.begin();  //update current element to
    point at front
149     this->cursorIsAtFront = 1;
150 }
151 //put cursor in back
152 //change iterator to end of list, put flag to not at front
153 void Line::back() {
154     if (this->nodeList.size()) {
155         this->currentElement = --this->nodeList.end();
156     } else {  //create new element, which is both end and beginning but we will
    treat it as end.
157         this->nodeList.push_back(deque<char>());
158         this->currentElement = this->nodeList.begin();
159     }
160     this->cursorIsAtFront = 0;  //should this be in the if statement?
161 }
```