You are given a string consisting of parenthesis style characters ( ) [ ] and { }. A string of this type is said to be correct if:

(a) It is an empty string.

(b) If string A is correct and string B is correct then string AB is correct.

(c) If A is correct then (A), [A] and {A} are all correct.

You are to write a program that takes as input from the keyboard a series of strings of this type that can be of any length. The first line of input will contain a single integer **n** telling you how many strings you will be testing. Following that integer will be **n** lines (each ending in a new line character) that represent the strings you are supposed to test with one string per line.
Your program should output **Yes** if a given string is correct and **No** otherwise.
For example if the user types the following as input

```
3
([])
(([{}]))
([()[]()])()
```

Then your program should give the following output:

```
Yes
No
Yes
```

You will note that if you do your program correctly there is no prompt for data and the yes and no answers will be interspersed with your input data when you enter it manually. The following would be a more accurate representation of what you see on the screen when you cut and paste the input into your program.

```
3
([])
Yes
(([{}]))
No
([()[]()])()
Yes
```

You may find it useful to use a text file for testing your code so that you do not have to keep retyping the tests. You can do this from the command line by typing the name of the executable file followed by a less than sign and the name of your file. For example, if my executable is named day7 then I could type:

day7 < input.txt

This would redirect standard input so that the input instead comes from the file named input.txt. You should note that the input.txt file needs to be in the same directory as your executable file for the above command to work. Alternatively you can give the complete path information for the input file.

You may not assume anything about the size of the test strings coming in. In fact, some of the test strings may be hundreds of thousands of characters long…

How do you solve it? The trick to this problem is realizing that our stack interface can be used to solve the problem. One way to approach the problem is to read in a character, if it is a left marker then push it onto the stack. If it is a right marker then check the top of the stack (if it exists), if it is the correct left marker then pop it out and continue testing the string. If you encounter a right marker and the top of the stack is not the left hand version of the marker you just picked up or the stack is empty then you can determine that the string is not correct, clear the stack and go on to the next example. If you finish a whole string (encounter a new line) without encountering a problem then you can determine that the input string is correct.

You should submit your program to an online checker that will test it for a small set of input values by going to uva.onlinejudge.org, making an account, and submitting your solution as a solution to problem 673. Even though the problem I have given you is a bit harder than the problem given there, your solution to this problem should have no problem solving their version of problem 673 that they have on the site. Submit your main.c and your stack.c, stack.h and any other support files you need along with a text file containing a copy of the email you get from the UVA online judge. Your stack.c implementation file should contain an implementation for the stack that can hold characters instead of integers and should use a linked list implementation of the stack.

All programs must include a comment section at the top of the program as outlined below:

```
/*************************************************************************
        Program:    <name of program>
        Author:     <your name>
        Date:       <date you finish the program>
        Time spent: <total amount of time spent on the project>
        Purpose:    The purpose of this program is to blah blah blah
*************************************************************************/
```