Quiz 1 (in-class)

- Due Sep 8 at 12:15pm
- Points 15
- Questions 1
- Available Sep 8 at 11:45am - Sep 8 at 12:25pm 40 minutes
- Time Limit 25 Minutes

Instructions

In-class quiz on Sep 8 (Monday). Type your codes into solution window.

This quiz was locked Sep 8 at 12:25pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	24 minutes	15 out of 15

ⓘ Correct answers are hidden.

Score for this quiz: 15 out of 15
Submitted Sep 8 at 12:09pm
This attempt took 24 minutes.

Question 1
15 / 15 pts

Problem Description

You are given a string s that contains only three kinds of characters: `"("`, `")"`, and `"*"`. Write a C++ code to determine whether the string s is a ***good format*** string.

A string s is a ***good format*** string if it satisfies the following conditions:

- Every left parenthesis `"("` must have a matching right parenthesis `")"`.
- Every right parenthesis `")"` must have a matching left parenthesis `"("`.
- Left parentheses `"("` must always appear before their corresponding right parentheses `")"` in the string.
- Each `"*"` character in s can be interpreted as one of the following characters in making a string *good format*:
  - A single right parenthesis `")"`,
  - A single left parenthesis `"("`,
  - An empty string.

Your C++ code shall return `TRUE` if the string s is a good format string according to these rules, and `FALSE` otherwise.

Input

A string s containing only the characters `"("`, `")"`, and `"*"`.

Output

Return `TRUE` if the string is a good format string, otherwise return `FALSE`.

Example 1

Input: `"(*)*"`
Output: `TRUE`

Explanation

The `"*"` can be treated as an empty string, making the parentheses balanced.

Example 2

Input: `"*()("`
Output: `FALSE`

Explanation

There are unmatched left parentheses, even though the `"*"` can act as a right parenthesis or an empty string.

Example 3

Input: `"((*)"`
Output: `TRUE`

Explanation

The `'*'` can be treated as a right parenthesis, balancing the two left parentheses.

Example 3

Input: `"((*)"`
Output: `TRUE`

Explanation

The `'*'` can be treated as a right parenthesis, balancing the two left parentheses.

Requirements

1. The solution to this problem must be implemented using C++ only. No other programming languages (including Python, Java, Matlab, R, etc.) are allowed.
2. Please respect the honor code: you are not allowed to use the internet, including ChatGPT or any other AI tools, to assist in the implementation of this problem. Otherwise, points will be deducted.
3. Do not change the framework we provided to you as follows, i.e., you can write whatever code you want between the two comments below, but do not change anything else. (Except for the value of string s in the main function, you can alter it when you do testing).

```
#include <iostream>
#include <string>
using namespace std;

class ChkString {
public:
    bool checkValidString(string s) {
    /* Your code starts here */




    /* Your code ends here */
    }
};


int main() {
    ChkString sol;
    string s = "(*))";
    cout << sol.checkValidString(s) << endl;  // Output: TRUE or FALSE
    return 0;
}
```

Hints

You may disregard the hints here if you have your own ideas.

To solve this problem, you can treat the parentheses and the ' $*$ ' character as part of a cumulative sum approach. Each left parenthesis ' (' increases the sum by 1, and each right parenthesis ')' decreases the sum by 1. The ' $*$ ' character can be interpreted as either a left parenthesis, a right parenthesis, or an empty string. To handle this flexibility, you maintain two variables: left_min and left_max. left_min represents the smallest possible balance (assuming every ' $*$ ' is a right parenthesis or empty), while left_max represents the largest possible balance (assuming every ' $*$ ' is a left parenthesis). As you process the string, update both variables for each character. If at any point left_max becomes negative, it means there are too many right parentheses, and the string is invalid. At the end of the string, if left_min is zero, the string is valid because all parentheses are balanced. If left_min is greater than zero, there are unmatched left parentheses, so the string is invalid. This approach efficiently checks the validity of the string without requiring backtracking or complex combinations.

Your Answer:

```
#include <iostream>
#include <string>
using namespace std;

class ChkString {
public:
bool checkValidString(string s) {
int minOpen = 0, maxOpen = 0;
for (char c : s) {
if (c == '(') {
minOpen++;
maxOpen++;
} else if (c == ')') {
minOpen--;
maxOpen--;
} else {
minOpen--;
maxOpen++;
}
if (maxOpen < 0) return false;
if (minOpen < 0) minOpen = 0;
}
return minOpen == 0;
}
};

int main() {
ChkString sol;
```

```
string s = "(*))";
cout << sol.checkValidString(s) << endl; // Output: TRUE or FALSE
return 0;
}
```

Quiz Score: 15 out of 15