**[Maximum Nesting Depth of the Parentheses](https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses/)**

A string is a **valid parentheses string** (denoted **VPS**) if it meets one of the following:

* It is an empty string "", or a single character not equal to "(" or ")",
* It can be written as AB (A concatenated with B), where A and B are **VPS**'s, or
* It can be written as (A), where A is a **VPS**.

We can similarly define the **nesting depth** depth(S) of any VPS S as follows:

* depth("") = 0
* depth(C) = 0, where C is a string with a single character not equal to "(" or ")".
* depth(A + B) = max(depth(A), depth(B)), where A and B are **VPS**'s.
* depth("(" + A + ")") = 1 + depth(A), where A is a **VPS**.

For example, "", "()()", and "()(()())" are **VPS**'s (with nesting depths 0, 1, and 2), and ")(" and "(()" are not **VPS**'s.

Given a **VPS** represented as string s, return *the****nesting depth****of*s.

**Example 1:**

**Input:** s = "(1+(2\*3)+((8)/4))+1"

**Output:** 3

**Explanation:** Digit 8 is inside of 3 nested parentheses in the string.

**Example 2:**

**Input:** s = "(1)+((2))+(((3)))"

**Output:** 3

**Constraints:**

* 1 <= s.length <= 100
* s consists of digits 0-9 and characters '+', '-', '\*', '/', '(', and ')'.
* It is guaranteed that parentheses expression s is a **VPS**.

class Solution {

public:

    int maxDepth(string s) {

        int count = 0;

        int max\_num = 0;

        for (char c : s) {

            if (c == '(') {

                count++;

                if (max\_num < count)

                    max\_num = count;

            } else if (c == ')') {

                count--;

            }

        }

        return max\_num;

    }

};

Link : <https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses/?envType=daily-question&envId=2024-04-04>