Check for Balanced Parentheses

Problem Statement: Check Balanced Parentheses. Given string str containing just the characters ‘(‘, ‘)’, ‘{‘, ‘}’, ‘[‘ and ‘]’, check if the input string is valid and return true if the string is balanced otherwise return false.

Note: string str is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

Input: str = “( )[{ } ( )]”

Output: True

Explanation: As every open bracket has its corresponding
close bracket. Match parentheses are in correct order
hence they are balanced.

Example 2:

Input: str = “[( )”

Output: False

Explanation: As ‘[‘ does not have ‘]’ hence it is
not valid and will return false.

Solution

***Disclaimer**: Don’t jump directly to the solution, try it out yourself first.*

Intuition: We have to keep track of previous as well as most recent opening brackets and also keep in mind the sequence, as after opening of the bracket there should be opposite pairs of brackets. Also handle the corner cases like [ ) ( ] where closing bracket occurs first and opening bracket after it, which is an invalid sequence, as well as [( ]) where the most recent opening didn’t get its opposite pair hence it will also not be valid.
So we have to use a data structure that will keep track of first in and last out, hence we will use the **stack**.

Approach:

- Whenever we get the opening bracket we will push it into the stack. I.e ‘{‘, ’[’, ’(‘.
- Whenever we get the closing bracket we will check if the stack is non-empty or not.
- If the stack is empty we will return false, else if it is nonempty then we will check if the topmost element of the stack is the opposite pair of the closing bracket or not.
- If it is not the opposite pair of the closing bracket then return false, else move ahead.
- After we move out of the string the stack has to be empty if it is non-empty then return it as invalid else it is a valid string.

Code:

- C++ Code
- Java Code
- Python Code

```
#include<bits/stdc++.h>
using namespace std;
bool isValid(string s) {
        stack<char>st;
        for(auto it: s) {
            if(it=='(' || it=='{' || it == '[') st.push(it);
            else {
                if(st.size() == 0) return false;
                char ch = st.top();
                st.pop();
                if((it == ')' and ch == '(') or  (it == ']' and ch == '[') or (it == '}' and ch == '{')) continue;
                else return false;
            }
        }
        return st.empty();
    }
int main()
{
    string s="()[{}()]";
    if(isValid(s))
    cout<<"True"<<endl;
    else
    cout<<"False"<<endl;
}
```

Output: True

Time Complexity: O(N)

Space Complexity: O(N)