Balanced Parentheses

Given an expression string **exp**, write a program to examine whether the pairs and the orders of "{", "}", "(", ")", "[", "]" are correct in the given expression.

Example:

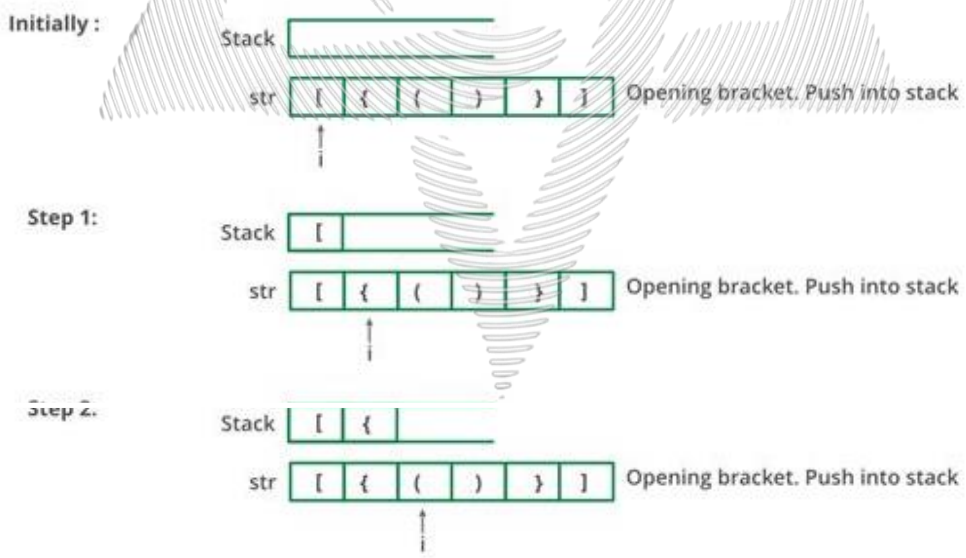
***Input**: exp = "[()]{}{[()()]()}"*
***Output**: Balanced*
***Explanation:** all the brackets are well-formed*

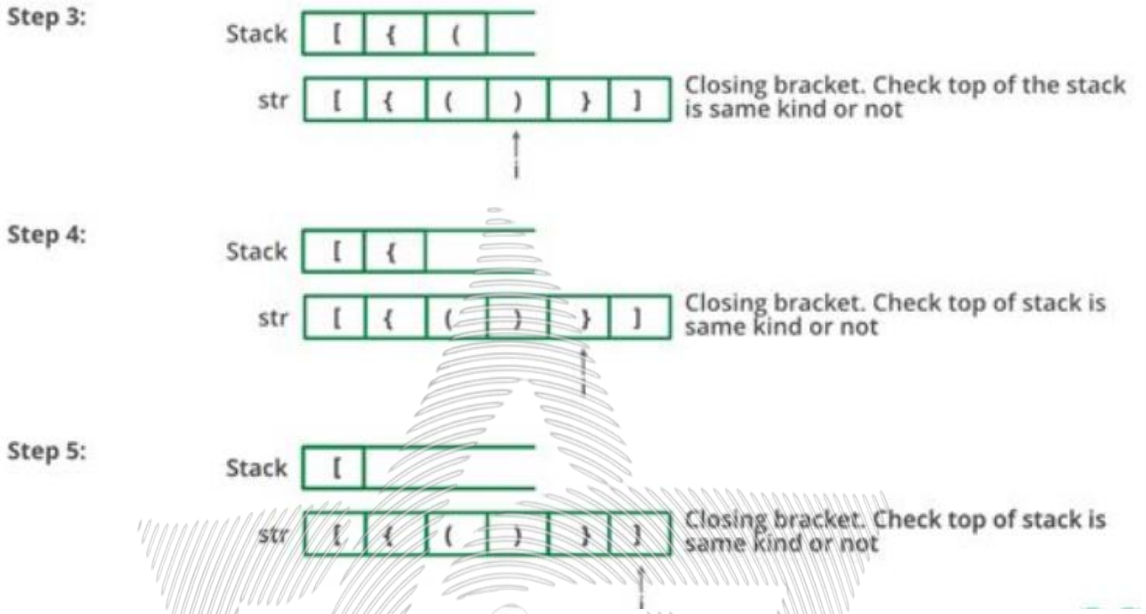
***Input**: exp = "[()"*
***Output**: Not Balanced*
***Explanation: 1** and **4** brackets are not balanced because there is a closing ')' before the closing '('*

Approach:

The idea is to put all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature. If this holds then pop the stack and continue the iteration, in the end if the stack is empty, it means all brackets are well-formed . Otherwise, they are not balanced.

Illustration:
Below is the illustration of the above approach.





Follow the steps mentioned below to implement the idea:

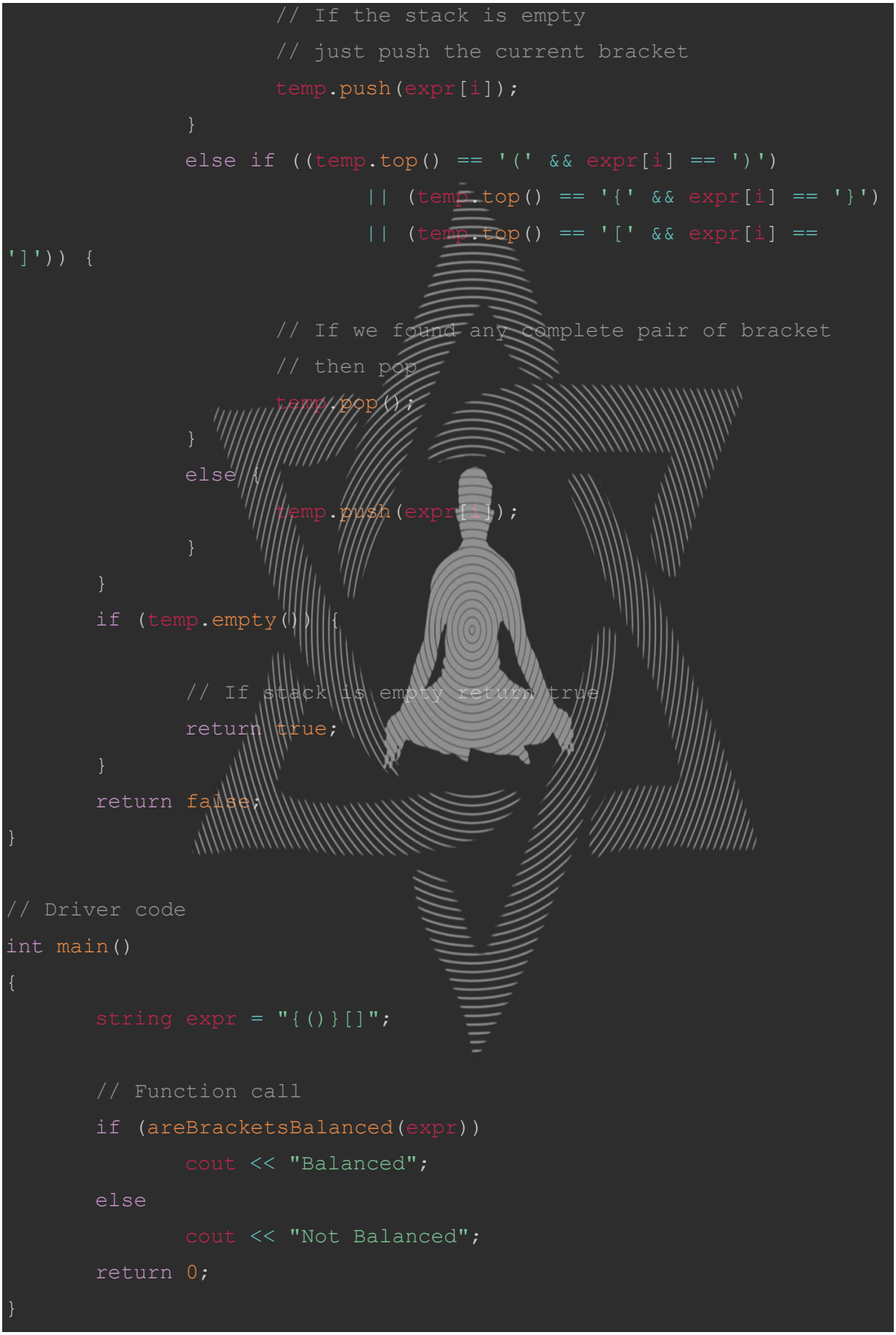
- Declare a character stack (say **temp**).
- Now traverse the string exp.
  - If the current character is a starting bracket (**‘(‘ or ‘{‘ or ‘[‘**) then push it to stack.
  - If the current character is a closing bracket (**‘)’ or ‘}’ or ‘]’**) then pop from stack and if the popped character is the matching starting bracket then fine.
  - Else brackets are **Not Balanced**.
- After complete traversal, if there is some starting bracket left in stack then **Not balanced**, else **Balanced**.

Below is the implementation of the above approach:

```
// C++ program to check for balanced brackets.

#include <bits/stdc++.h>
using namespace std;

// Function to check if brackets are balanced
bool areBracketsBalanced(string expr)
{
	// Declare a stack to hold the previous brackets.
	stack<char> temp;
	for (int i = 0; i < expr.length(); i++) {
		if (temp.empty()) {
```



```
                        // If the stack is empty
                        // just push the current bracket
                        temp.push(expr[i]);
                }
                else if ((temp.top() == '(' && expr[i] == ')')
                                || (temp.top() == '{' && expr[i] == '}')
                                || (temp.top() == '[' && expr[i] ==
']')) {

                        // If we found any complete pair of bracket
                        // then pop
                        temp.pop();
                }
                else {
                        temp.push(expr[i]);
                }
        }
        if (temp.empty()) {

                // If stack is empty return true
                return true;
        }
        return false;
}

// Driver code
int main()
{
        string expr = "{()}[]";

        // Function call
        if (areBracketsBalanced(expr))
                cout << "Balanced";
        else
                cout << "Not Balanced";
        return 0;
}
```

Output
Balanced

Time Complexity: O(N), Iteration over the string of size N one time.
Auxiliary Space: O(N) for stack.