Balanced Parenthesis

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

Example:

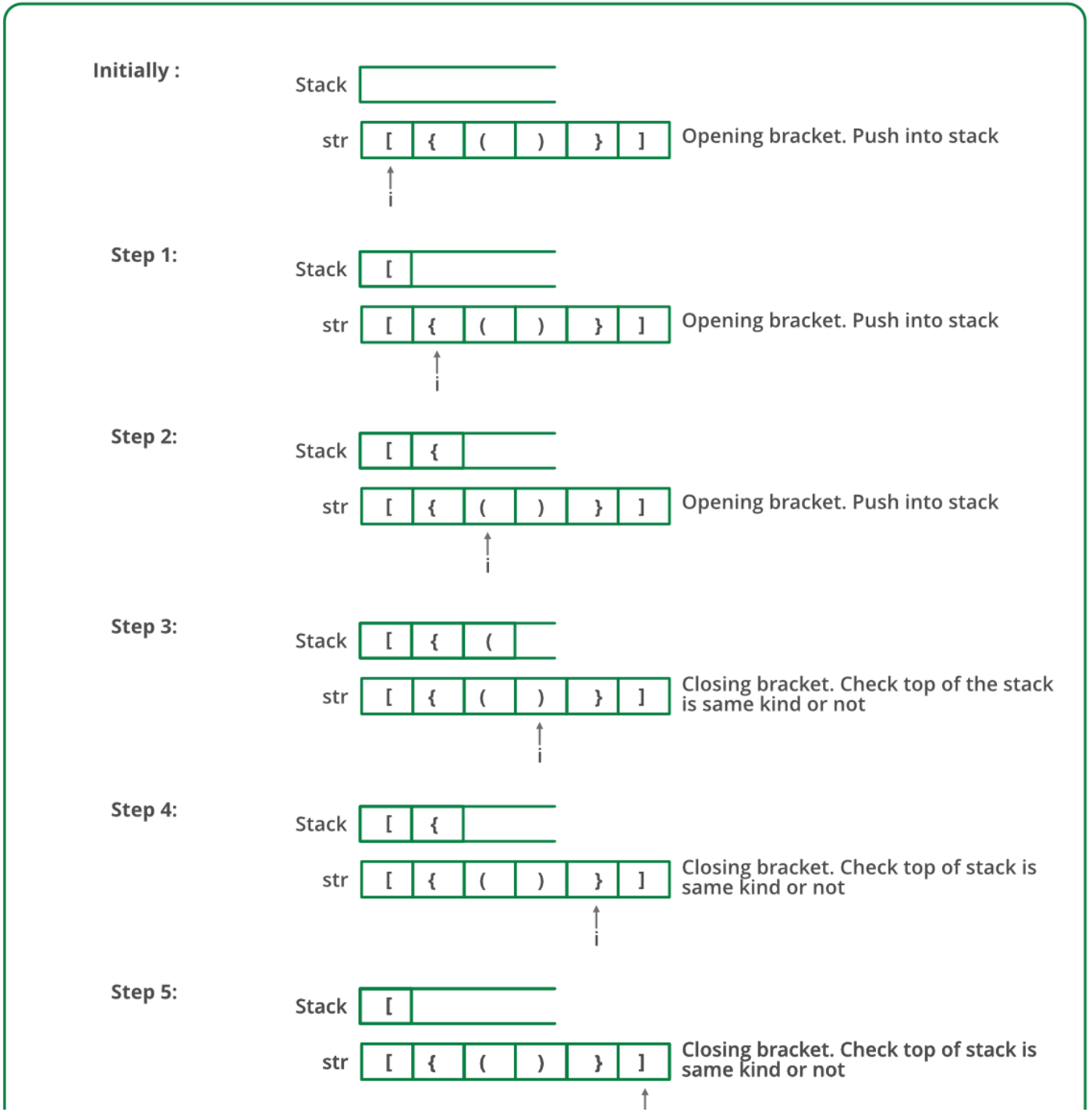
> ***Input***: *exp = “[()]{}{[()()]()}”*
> ***Output***: *Balanced*
> ***Explanation:*** *all the brackets are well-formed*
>
> ***Input***: *exp = “[(])”*
> ***Output***: *Not Balanced*
> ***Explanation: 1*** *and* ***4*** *brackets are not balanced because there is a closing ‘]’ before the closing ‘(‘*

Check for Balanced Bracket expression using Stack:

> *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.


Initially :
Stack
str [{ ( ) }]
i
Opening bracket. Push into stack
Step 1:
Stack [
str [{ ( ) }]
i
Opening bracket. Push into stack
Step 2:
Stack [{
str [{ ( ) }]
i
Opening bracket. Push into stack
Step 3:
Stack [{ (
str [{ ( ) }]
i
Closing bracket. Check top of the stack is same kind or not
Step 4:
Stack [{
str [{ ( ) }]
i
Closing bracket. Check top of stack is same kind or not
Step 5:
Stack [
str [{ ( ) }]
Closing bracket. Check top of stack is same kind or not

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++ Java

```
// Java program for checking
// balanced brackets
import java.util.*;

public class BalancedBrackets {

    // function to check if brackets are balanced
    static boolean areBracketsBalanced(String expr)
    {
        // Using ArrayDeque is faster than using Stack class
        Deque<Character> stack
            = new ArrayDeque<Character>();

        // Traversing the Expression
```

```
for (int i = 0; i < expr.length(); i++) {
    char x = expr.charAt(i);

    if (x == '(' || x == '[' || x == '{') {
        // Push the element in the stack
        stack.push(x);
        continue;
    }

    // If current character is not opening
    // bracket, then it must be closing. So stack
    // cannot be empty at this point.
    if (stack.isEmpty())
        return false;
    char check;
    switch (x) {
    case ')':
        check = stack.pop();
        if (check == '{' || check == '[')
            return false;
        break;

    case '}':
        check = stack.pop();
        if (check == '(' || check == '[')
            return false;
        break;

    case ']':
```

```
                check = stack.pop();
                if (check == '(' || check == '{')
                    return false;
                break;
            }
        }

        // Check Empty Stack
        return (stack.isEmpty());
    }

    // Driver code
    public static void main(String[] args)
    {
        String expr = "([{}])";

        // Function call
        if (areBracketsBalanced(expr))
            System.out.println("Balanced ");
        else
            System.out.println("Not Balanced ");
    }
}
```

Output

```
Balanced
```

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