**QPREP9-Valid parenthesis**

**Module Introduction**

Write a program to take a string containing '(', ')', '{', '}', '[' and ']' as input and output whether it's a valid string.

Problem Solving

#### Objective

Given a string containing the characters '(', ')', '{', '}', '[' and ']' as input, confirm if the input string is valid per the following rules:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Assume that an empty string is also valid.

#### Examples

**Example 1**

Input: []

Output: true

**Example 2**

Input: []{}()

Output: true

**Example 3**

Input: {]

Output: false

**Example 4**

Input: ([)]

Output: false

**Example 5**

Input: {[()]}

Output: true

***SOLUTION STEPS FROM NEXT PAGE:***

**Write down at least 3 examples in the following format. Kindly stick to the format.**

**Suggestion:**

EXAMPLE#1

INPUT:

([)]

OUTPUT:

false

EXAMPLE#2

INPUT:

{[]}

OUTPUT:

true

EXAMPLE#3

INPUT:

(([]){})

OUTPUT:

true

**Detail your problem understanding here**

**Suggestion:**

A string consisting of the these characters - ‘[‘, ‘]’, ‘{‘, ‘}’, ‘(‘ and ‘)’ - will be provided as input.

We have to verify if the string represents logical and complete bracket structure. If an outer bracket has inner brackets within it, the inner brackets have to be closed before the outer bracket is closed. All brackets that have been opened have to be closed with their corresponding closure bracket, for the string to be valid.

**Does this problem follow a known algorithmic pattern or standard application of a data structure? If there are multiple approaches, which one would you choose and why? Write down your chosen approach in 2-3 sentences like you would explain to a 10 year old.**

This problem is related to String manipulation. The concept of Stack can be used to effectively solve this problem. We can push each opening bracket to the stack and pop it when a matching closing bracket is found. The stack should be empty at the end.

**Write the pseudocode here in plain English**

Read the input string

initializeMatchingBrackets()

Initialize an empty stack

For each bracket in the input string {

If isOpenBracket(), push it to stack

If isClosingBracket() {

If stack is empty, return false

Pop from the stack and fetch the opening bracket

If the closing and opening bracket are not of the same type, return false

}

}

If the stack is not empty, return false

**Can you specify a few boundary or edge cases here?**

**Edge cases**

EXAMPLE#1

INPUT:

{

OUTPUT:

false

EXAMPLE#2

INPUT:

(empty)

OUTPUT:

true

EXAMPLE#3

INPUT:

((((

OUTPUT:

false

**Write the functions you would create here**

void initializeMatchingBrackets()

Boolean validateBracketString(String input)

Boolean isOpenBracket(String input)

Boolean isClosingBracket(String input)

Boolean isMatchingBracket(String openBracket, String closingBracket)

#### Summary

Starting with a brief explanation of the problem statement followed by pseudocode and then implementing the solution helps you approach the problem in a systematic way. This methodology helps with easy as well as hard problems.

**Time Complexity: O(n)**

Where n is the length of the input string. We traverse through the string once, handling one bracket at a time.

**Space Complexity: O(n)**

We use the stack and push/pop each of the characters in the string as needed.

#### Concepts

Concepts covered in this Module

* Stack
* String

Similar problems

* <https://leetcode.com/problems/generate-parentheses/>
* <https://leetcode.com/problems/longest-valid-parentheses/>
* <https://leetcode.com/problems/remove-invalid-parentheses/>

References

* <https://www.geeksforgeeks.org/stack-data-structure-introduction-program/>
* <https://www.geeksforgeeks.org/stack-data-structure/>
* <https://medium.com/@codingfreak/stack-data-structure-practice-problems-and-interview-questions-9f08a35a7f19>

SOLUTION:

APPROACH 1:

import java.io.\*;

import java.util.\*;

public class ValidParenthesis {

// Implement your solution by completing the below function

public boolean isValid(String s) {

String input=s.replaceAll("[^\\(\\)\\[\\]\\{\\}]", "");

while(input.length() != (input = input.replaceAll("\\(\\)|\\[\\]|\\{\\}", "")).length());

return input.isEmpty();

}

public static void main(String[] args) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

String parenthesis = in.readLine();

boolean result = new ValidParenthesis().isValid(parenthesis);

System.out.print(String.valueOf(result));

}

}

**Complexity Analysis:**

* **Time Complexity:**
* **Space Complexity:**

private HashMap<Character, Character> mappings;

public ValidParenthesis() {

this.mappings = new HashMap<Character, Character>();

this.mappings.put(')', '(');

this.mappings.put('}', '{');

this.mappings.put(']', '[');

}

public boolean isValid(String s) {

// Initialize a stack to be used in the algorithm.

Stack<Character> stack = new Stack<Character>();

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

// If the current character is a closing bracket.

if (this.mappings.containsKey(c)) {

// Get the top element of the stack. If the stack is empty, set a dummy value of '#'

char topElement = stack.empty() ? '#' : stack.pop();

// If the mapping for this bracket doesn't match the stack's top element, return false.

if (topElement != this.mappings.get(c)) {

return false;

}

} else {

// If it was an opening bracket, push to the stack.

stack.push(c);

}

}

// If the stack still contains elements, then it is an invalid expression.

return stack.isEmpty();

}