

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Complex Problem Assignment

Student Name: Lovyansh Dhanraj

UID: 22BCS17264

Branch: CSE

Section: DL_901/B

Semester: 6th

DOP: 03/04/25

Subject: PBLJ

Subject Code:22CSH-359

1. **Problem:** Consider a function `public String matchFound(String input 1, String input 2)`, where

- input1 will contain only a single word with only 1 character replaces by an underscore ‘_’
- input2 will contain a series of words separated by colons and no space character in between
- input2 will not contain any other special character other than underscore and alphabetic characters.

The methods should return output in a String type variable “output1” which contains all the words from input2 separated by colon which matches with input 1. All words in output1 should be in uppercase

Code:

```
public String matchFound(String input1, String input2) {  
    // Split the input2 into individual words based on colon separator.  
    String[] words = input2.split(":");  
  
    // Use a StringBuilder to build the output string.  
    StringBuilder output1 = new StringBuilder();  
  
    // Loop through each word in the array.  
    for (String word : words) {  
        // Check if the length of the word matches the length of input1.  
        if (word.length() == input1.length()) {  
            boolean matches = true;  
            // Compare each character.
```

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

```
for (int i = 0; i < input1.length(); i++) {  
  
    char ch1 = input1.charAt(i);  
    char ch2 = word.charAt(i);  
    // If the character in input1 is not underscore,  
    // then it must match the corresponding character in word.  
    if (ch1 != '_' && Character.toLowerCase(ch1) !=  
Character.toLowerCase(ch2)) {  
        matches = false;  
        break;  
    }  
}  
  
// If the word matches, add it (in uppercase) to the output.  
if (matches) {  
    if (output1.length() > 0) {  
        output1.append(":");  
    }  
    output1.append(word.toUpperCase());  
}  
}  
  
// Return the final output string.  
return output1.toString();  
}
```

Output:

Expected Output:

"APPLE:AMPLE:AXPLE"

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2. **Problem:** String t is generated by random shuffling string s and then add one more letter at a random position.


Return the letter that was added to t.

Code:

```
public class FindAddedLetter {
    public static char findAddedLetter(String s, String t) {
        int result = 0;
        for (char c : s.toCharArray()) {
            result ^= c; // XOR all characters in s
        }
        for (char c : t.toCharArray()) {
            result ^= c; // XOR all characters in t
        }
        return (char) result; // Convert result back to char
    }

    public static void main(String[] args) {
        String s = "abcd";
        String t = "abcde";
        System.out.println("Added letter: " + findAddedLetter(s, t)); // Output:
        "e"
    }
}
```

Output:



Added letter: e

3. **Problem:** The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.
- You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.
- For each $0 \leq i < \text{nums1.length}$, find the index j such that $\text{nums1}[i] == \text{nums2}[j]$ and determine the next greater element of $\text{nums2}[j]$ in nums2. If there is no next greater element, then the answer for this query is -1.
- Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Code:

```
import java.util.*;

public class NextGreaterElement {
    public static int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();

        // Compute next greater elements for nums2
        for (int num : nums2) {
            while (!stack.isEmpty() && stack.peek() < num) {
                nextGreaterMap.put(stack.pop(), num);
            }
            stack.push(num);
        }

        // Fill result for nums1 using the precomputed map
        int[] ans = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            ans[i] = nextGreaterMap.getOrDefault(nums1[i], -1);
        }
        return ans;
    }

    public static void main(String[] args) {
        int[] nums1 = {4, 1, 2};
        int[] nums2 = {1, 3, 4, 2};

        int[] result = nextGreaterElement(nums1, nums2);
        System.out.println(Arrays.toString(result)); // Output: [-1, 3, -1]
    }
}
```

Output:

```
[-1, 3, -1]
```

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

4. **Problem:** A string containing only parentheses is balanced if the following is true: 1. if it is an empty string 2. if A and B are correct, AB is correct, 3. if A is correct, (A) and {A} and [A] are also correct.

Examples of some correctly balanced strings are: "{}()", "[{}]", "({})"

Examples of some unbalanced strings are: "{}(", "({})", "[[", "{}" etc.

Given a string, determine if it is balanced or not.

Code:

```
import java.util.*;

public class BalancedParentheses {
    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c); // Push opening brackets
            } else if (c == ')' || c == '}' || c == ']') {
                if (stack.isEmpty()) return false; // No matching opening bracket
                char top = stack.pop();
                if ((c == ')' && top != '(') ||
                    (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false; // Mismatched pair
                }
            }
        }
        return stack.isEmpty(); // If stack is empty, it's balanced
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<String> results = new ArrayList<>();

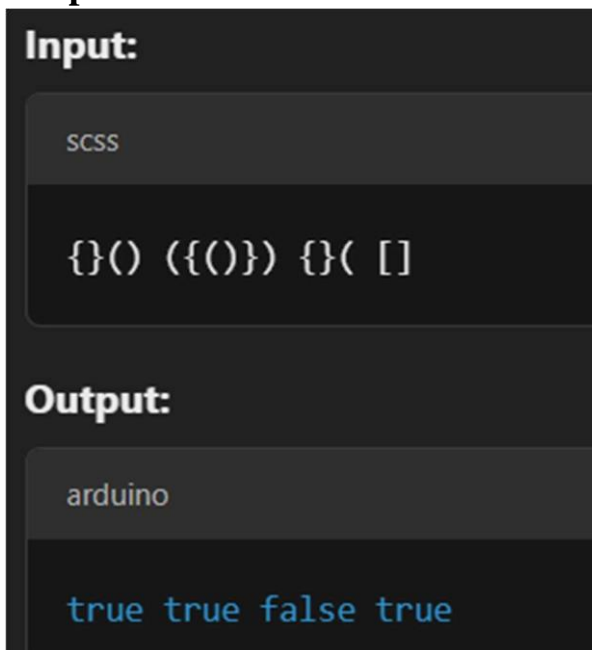
        while (scanner.hasNext()) { // Read input till EOF
            String input = scanner.next();
            results.add(isBalanced(input) ? "true" : "false");
        }
    }
}
```

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

```
    }

    // Print results in a single line
    System.out.println(String.join(" ", results));
    scanner.close();
}
}
```

Output:



The screenshot shows a terminal window with a dark background. Under the heading "Input:", there are two input lines. The first line contains the text "SCSS". The second line contains the text "{}() ({}()) {}([]". Under the heading "Output:", there is one output line containing the text "arduino". Below that, the program's output is displayed as "true true false true" in a light blue color.

5. **Problem:** Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return `[-1, -1]`.

You must write an algorithm with $O(\log n)$ runtime complexity.

Code:

```
import java.util.Arrays;
```

```
public class FirstLastPosition {
    public static int[] searchRange(int[] nums, int target) {
        int first = findBound(nums, target, true);
        int last = findBound(nums, target, false);
        return new int[]{first, last};
    }
}
```

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

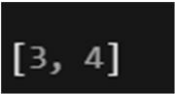
```
private static int findBound(int[] nums, int target, boolean isFirst) {
    int left = 0, right = nums.length - 1;
    int bound = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (nums[mid] == target) {
            bound = mid; // Store the found position
            if (isFirst) {
                right = mid - 1; // Search towards the left
            } else {
                left = mid + 1; // Search towards the right
            }
        } else if (nums[mid] < target) {
            left = mid + 1; // Move right
        } else {
            right = mid - 1; // Move left
        }
    }
    return bound;
}

public static void main(String[] args) {
    int[] nums = {5,7,7,8,8,10};
    int target = 8;
    System.out.println(Arrays.toString(searchRange(nums, target))); //
    Output: [3, 4]
}
```

Output:



[3, 4]