



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## ASSIGNMENT

**Student Name:** Utsav Mallik

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** PBLJ

**UID:** 22BCS11462

**Section/Group:** IOT\_644B

**Date of Performance:** 03/04/25

**Subject Code:** 22CSH-359

### **Problem 1.**

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.

### **Implementation/Code:**

```
public class MatchFinder {  
  
    // Method to find matching words  
    public String matchFound(String input1, String input2) {  
        String[] words = input2.split(":");  
        StringBuilder output1 = new StringBuilder();  
  
        int missingIndex = input1.indexOf('_');  
  
        for (String word : words) {  
            if (word.length() != input1.length()) {  
                continue;  
            }  
  
            boolean isMatch = true;  
            for (int i = 0; i < input1.length(); i++) {  
                if (i != missingIndex && input1.charAt(i) != word.charAt(i)) {  
                    isMatch = false;  
                    break;  
                }  
            }  
  
            if (isMatch) {  
                output1.append(word + ":");  
            }  
        }  
  
        return output1.toString().trim();  
    }  
}
```

```
        }  
    }  
  
    if (isMatch) {  
        if (output1.length() > 0) {  
            output1.append(":");  
        }  
        output1.append(word.toUpperCase());  
    }  
}  
  
return output1.toString();  
}  
  
// Main method to test the code  
public static void main(String[] args) {  
    MatchFinder mf = new MatchFinder();  
  
    // Sample test  
    String input1 = "he_lo";  
    String input2 = "hello:helpo:hezlo:healo";  
  
    String result = mf.matchFound(input1, input2);  
    System.out.println("Matching words: " + result);  
}  
}
```

## Output

### Output

Matching words: HELLO:HEZLO:HEALO

=== Code Execution Successful ===

## Problem 2:

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.

Hint:

Input: s = "abcd", t = "abcde"

Output: "e"

## Implementation/Code:

```
public class ExtraCharacterFinder {  
  
    public char findTheDifference(String s, String t) {  
  
        int sumS = 0, sumT = 0;  
  
        for (char ch : s.toCharArray()) {  
  
            sumS += ch;  
  
        }  
  
        for (char ch : t.toCharArray()) {  
  
            sumT += ch;  
  
        }  
  
        return (char)(sumT - sumS);  
  
    }  
  
    public static void main(String[] args) {  
  
        ExtraCharacterFinder finder = new ExtraCharacterFinder();  
  
        String s = "abcd";  
  
        String t = "abcde";  
  
        char result = finder.findTheDifference(s, t);  
  
        System.out.println("The added character is: " + result);  
  
    }  
}
```

}

**Output**

The added character is: e

=== Code Execution Successful ===

**Problem 3:**

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  $\text{nums1}$  and  $\text{nums2}$ , where  $\text{nums1}$  is a subset of  $\text{nums2}$ . 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  $\text{nums2}$ . If there is no next greater element, then the answer for this query is  $-1$ . Return an array  $\text{ans}$  of length  $\text{nums1.length}$  such that  $\text{ans}[i]$  is the next greater element as described above.

Hint: Input:  $\text{nums1} = [4,1,2]$ ,  $\text{nums2} = [1,3,4,2]$

Output:  $[-1,3,-1]$

Explanation: The next greater element for each value of  $\text{nums1}$  is as follows:  $-4$  is underlined in  $\text{nums2} = [1,3,4,2]$ . There is no next greater element, so the answer is  $-1$ .  $-1$  is underlined in  $\text{nums2} = [1,3,4,2]$ . The next greater element is  $3$ .  $-2$  is underlined in  $\text{nums2} = [1,3,4,2]$ . There is no next greater element, so the answer is  $-1$ .

**Implementation/Code:**

```
import java.util.*;
public class NextGreaterElementFinder {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();
        for (int num : nums2) {
            while (!stack.isEmpty() && num > stack.peek()) {
                nextGreaterMap.put(stack.pop(), num);
            }
        }
    }
}
```

```
        }
        stack.push(num);
    }
    while (!stack.isEmpty()) {
        nextGreaterMap.put(stack.pop(), -1);
    }

    // Step 3: Build the result for nums1
    int[] result = new int[nums1.length];
    for (int i = 0; i < nums1.length; i++) {
        result[i] = nextGreaterMap.get(nums1[i]);
    }
    return result;
}

public static void main(String[] args) {
    NextGreaterElementFinder finder = new NextGreaterElementFinder();

    int[] nums1 = {4, 1, 2};
    int[] nums2 = {1, 3, 4, 2};

    int[] result = finder.nextGreaterElement(nums1, nums2);
    System.out.println("Next greater elements: " + Arrays.toString(result));
}
}
```

## Output

Next greater elements: [-1, 3, -1]

=== Code Execution Successful ===

## Problem 4:

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.

**Input Format** There will be multiple lines in the input file, each having a single non-empty string. You should read input till end-of-file.

**Output Format** For each case, print 'true' if the string is balanced, 'false' otherwise.

**Sample Input** {}() ({()}) {}( [] **Sample Output** true true false true

## Implementation/Code:

```
import java.util.*;

import java.io.*;

public class BalancedParenthesesChecker {

    public static boolean isBalanced(String str) {

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

        for (char ch : str.toCharArray()) {

            switch (ch) {

                case '(': case '{': case '[':

                    stack.push(ch);

                    break;

                case ')':

                    if (stack.isEmpty() || stack.pop() != '(') return false;

                    break;

                case '}':

                    if (stack.isEmpty() || stack.pop() != '{') return false;
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        break;

    case ']':

        if (stack.isEmpty() || stack.pop() != '[') return false;

        break;

    }

}

return stack.isEmpty();

}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    while (scanner.hasNextLine()) {

        String line = scanner.nextLine().trim();

        if (!line.isEmpty()) {

            System.out.println(isBalanced(line));

        }

    }

    scanner.close();

}

}
```

## OUTPUT:

Output

```
{ } ( [ ]  
false  
( ) { }  
true
```

### Problem 5:

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.

Example 1: Input: `nums = [5,7,7,8,8,10]`, `target = 8`

Output: `[3,4]`

Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- `nums` is a non-decreasing array.
- $-10^9 \leq \text{target} \leq 10^9$

### Implementation/Code:

```
public class TargetRangeFinder {  
    // Method to find the first and last position  
    public int[] searchRange(int[] nums, int target) {  
        int first = findPosition(nums, target, true);  
        int last = findPosition(nums, target, false);  
        return new int[] { first, last };  
    }  
    // Helper method for binary search  
    private int findPosition(int[] nums, int target, boolean findFirst) {  
        int left = 0, right = nums.length - 1;  
        int result = -1;  
  
        while (left <= right) {  
            int mid = left + (right - left) / 2;
```



```
        if (nums[mid] == target) {
            result = mid;
            // Keep looking left for first, right for last
            if (findFirst) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return result;
}

// Main method for testing
public static void main(String[] args) {
    TargetRangeFinder finder = new TargetRangeFinder();

    int[] nums = {5, 7, 7, 8, 8, 10};
    int target = 8;

    int[] result = finder.searchRange(nums, target);
    System.out.println("Target range: [" + result[0] + ", " + result[1] + "]");
}
}
```

## OUTPUT:

### Output

Target range: [3, 4]

=== Code Execution Successful ===