DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ASSIGNMENT

Name: Aashish Pant **UID:22BCS12862**
Branch: BE-CSE **Section/Group:642 IOT B**
Semester: 6th **Date of Performance:13/3/2025**
Subject Name:PLJB **Subject Code: 22CSH-359**

Problem 1.

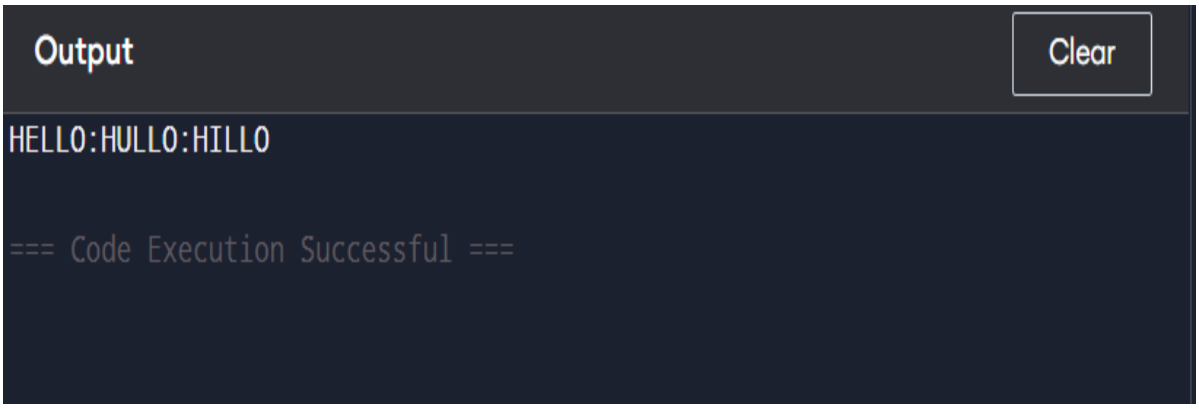
Consider a function **public String matchFound(String input 1, String input 2),** where
- **input1** will contain only a single word with only 1 character replaces by an underscore '_'
- **input2** will contain a series of words separated by colons and no space character in between
- **input2** will not contain any other special character other than underscore and alphabetic characters.

CODE:

```
public class WordMatcher {

public String matchFound(String input1, String input2) {
   String[] words = input2.split(":");
   StringBuilder output1 = new StringBuilder();

   for (String word : words) {
      if (word.length() == input1.length()) {
         boolean match = true;
         for (int i = 0; i < input1.length(); i++) {
            if (input1.charAt(i) != '_' && input1.charAt(i) != word.charAt(i)) {
               match = false;
               break;
            }
         }
         if (match) {
            if (output1.length() > 0) {
               output1.append(":");
            }
            output1.append(word.toUpperCase());
         }
      }
   }

   return output1.toString();
}

public static void main(String[] args) {
   WordMatcher wm = new WordMatcher();
```

```
        // Sample Test
        String input1 = "h_llo";
        String input2 = "hello:hullo:halloo:hella:hillo";
        String result = wm.matchFound(input1, input2);
        System.out.println(result);  // Output: HELLO:HULLO:HILLA:HILLO
    }
}
```

OUTPUT



Problem 2:

Given a String (In Uppercase alphabets or Lowercase alphabets), new alphabets is to be appended with following rule:
(i) If the alphabet is present in the input string, use the numeric value of that alphabet.

E.g. a or A numeric value is 1 and so on. New alphabet to be appended between 2 alphabets:
(a) If (sum of numeric value of 2 alphabets) %26 is 0, then append 0.

E.g. string is ay. Numeric value of a is 1, y is 25. Sum is 26. Remainder is 0, the new string will be a0y.
(b) Otherwise (sum of numeric value of 2 alphabets) %26 numeric value alphabet is to be appended. E.g. ac is string. Numeric value of a is 1, c is 3, sum is 4. Remainder with 26 is 4. Alphabet to be appended is d. output will be adc.
(ii) If a digit is present, it will be the same in the output string. E.g. string is 12, output string is 12.
(iii) If only a single alphabet is present, it will be the same in the output string. E.g. input string is 1a, output will be 1a.
(iv) If space is present, it will be the same in the output string. E.g. string is ac 12a, output will be adc 12a.

Constraint: Whether string alphabets are In Uppercase or Lowercase, appended alphabets must be in lower case. Output string must also be in lowercase.

```
public class AlphabetAppender {

    public static String appendNewAlphabets(String input) {
        StringBuilder output = new StringBuilder();
        // Convert to lowercase to enforce the output constraint
        String lowerInput = input.toLowerCase();
        int len = lowerInput.length();

        // Process each character
        for (int i = 0; i < len; i++) {
            char current = lowerInput.charAt(i);
            output.append(current);

            // Lookahead: if there's a next character, and both are alphabets, process them.
            if (i < len - 1) {
                char next = lowerInput.charAt(i + 1);
                if (Character.isLetter(current) && Character.isLetter(next)) {
                    // Compute numeric values: 'a' = 1, 'b' = 2, ..., 'z' = 26.
                    int currentVal = current - 'a' + 1;
                    int nextVal = next - 'a' + 1;
                    int sum = currentVal + nextVal;

                    // Compute the remainder when divided by 26
                    int remainder = sum % 26;

                    if (remainder == 0) {
                        // If remainder is 0, append the digit '0'
                        output.append('0');
                    } else {
                        // Otherwise, map remainder to corresponding lowercase letter
                        // For example: remainder 4 -> 'd' because 'a' + 4 - 1 = 'd'
                        char newChar = (char) ('a' + remainder - 1);
```

```
                output.append(newChar);
            }
        }
    }
}

    return output.toString();
}

public static void main(String[] args) {
    // Test examples
    System.out.println(appendNewAlphabets("ay"));       // Expected: a0y  (1+25 = 26 -> 0 appended)
    System.out.println(appendNewAlphabets("ac"));       // Expected: adc  (1+3 = 4 -> 'd' appended)
    System.out.println(appendNewAlphabets("12"));       // Expected: 12   (digits remain unchanged)
    System.out.println(appendNewAlphabets("1a"));       // Expected: 1a   (only one letter 'a', so remains same)
    System.out.println(appendNewAlphabets("ac 12a"));   // Expected: adc 12a
    System.out.println(appendNewAlphabets("HELLO"));    // Expected: heblmlpo
}
}
```

Output Clear

```
a0y
adc
12
1a
adc 12a
hmeqlxlao

=== Code Execution Successful ===
```

PROBLEM 3:

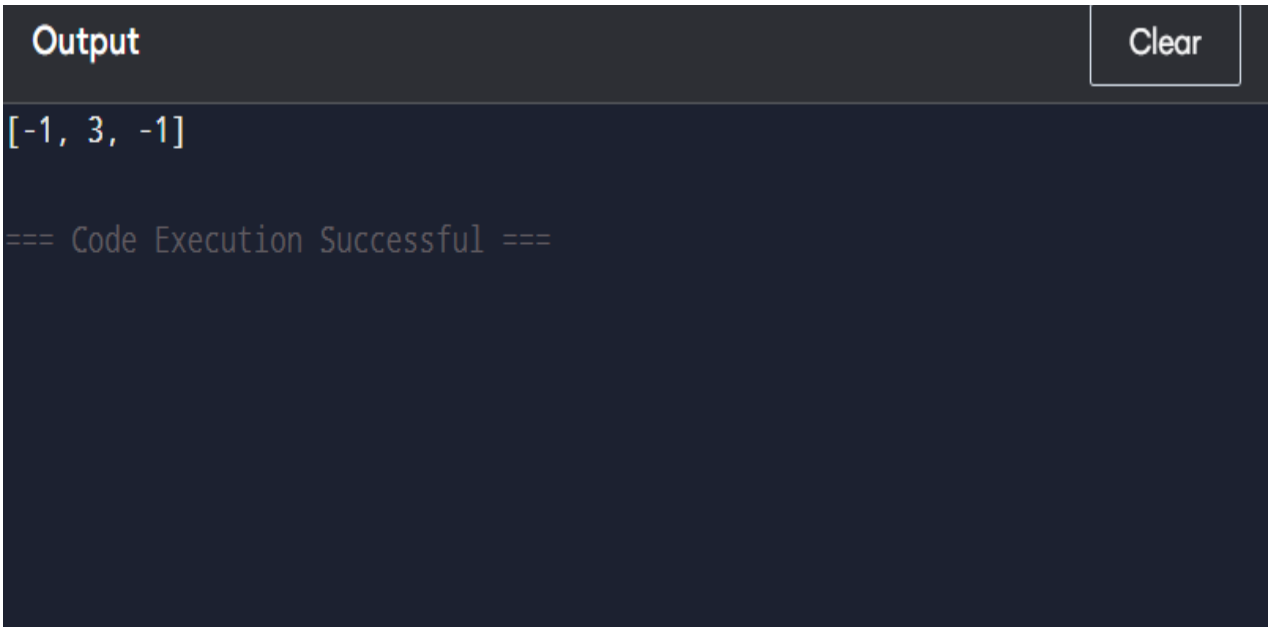
The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.
You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.
For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of 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.
Hint:
Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
Output: [-1,3,-1]
Explanation: The next greater element for each value of nums1 is as follows:
- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

CODE

```
import java.util.*;

public class NextGreaterElement {

    public static int[] nextGreaterElement(int[] nums1, int[] nums2) {
        // Map from element to its next greater element
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();

        // Traverse nums2 and compute next greater for each element
        for (int num : nums2) {
            // Pop elements that are smaller than current number
            while (!stack.isEmpty() && num > stack.peek()) {
                nextGreaterMap.put(stack.pop(), num);
            }
            stack.push(num);
        }

        // For remaining elements in the stack, no next greater element
        while (!stack.isEmpty()) {
            nextGreaterMap.put(stack.pop(), -1);
        }

        // Build result for nums1 based on the map
        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) {
```

```
        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



Problem 4:
Comparators are used to compare two objects. In this challenge, you'll create a comparator and use it to sort an array.
The Player class has fields: a String and a integer.
Given an array of Player objects, write a comparator that sorts them in order of decreasing score; if or more players have the same score, sort those players alphabetically by name.
To do this, you must create a Checker class that implements the Comparator interface, then write an int compare(Player a, Player b) method implementing the Comparator.compare(T o1, T o2) method.
Input Format
The first line contains an integer, denoting the number of players. Each of the subsequent lines contains a player's and , respectively.
Constraints
• players can have the same name.
• Player names consist of lowercase English letters.

```
import java.util.*;

// Player class
class Player {
    String name;
    int score;

    Player(String name, int score) {
        this.name = name;
        this.score = score;
    }
}

// Checker class that implements Comparator
class Checker implements Comparator<Player> {
    public int compare(Player a, Player b) {
        // First compare scores (descending)
        if (a.score != b.score) {
            return b.score - a.score;
        }
        // If scores are equal, compare names (ascending)
        return a.name.compareTo(b.name);
    }
}

public class PlayerSorter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // number of players
        scanner.nextLine(); // consume newline

        Player[] players = new Player[n];

        for (int i = 0; i < n; i++) {
            String name = scanner.next();
            int score = scanner.nextInt();
```

```
            players[i] = new Player(name, score);
        }

        // Sort using custom comparator
        Arrays.sort(players, new Checker());

        // Output the sorted list
        for (Player p : players) {
            System.out.println(p.name + " " + p.score);
        }

        scanner.close();
    }
}
```

Problem 5:
Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:
• '?' Matches any single character.
• '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).
Example 1:
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
Constraints:
• 0 <= s.length, p.length <= 2000
• s contains only lowercase English letters.
• p contains only lowercase English letters, '?' or '*'.

```
public class WildcardMatching {

    public static boolean isMatch(String s, String p) {
        int m = s.length();
        int n = p.length();

        // dp[i][j] = whether s[0..i-1] matches p[0..j-1]
        boolean[][] dp = new boolean[m + 1][n + 1];

        dp[0][0] = true; // Empty pattern matches empty string

        // Initialize for pattern starting with *
        for (int j = 1; j <= n; j++) {
            if (p.charAt(j - 1) == '*')
```

```
            dp[0][j] = dp[0][j - 1];
        }

        // Fill DP table
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                char sc = s.charAt(i - 1);
                char pc = p.charAt(j - 1);

                if (pc == '?' || pc == sc) {
                    dp[i][j] = dp[i - 1][j - 1];
                } else if (pc == '*') {
                    dp[i][j] = dp[i][j - 1] || dp[i - 1][j];
                }
            }
        }

        return dp[m][n];
    }

    public static void main(String[] args) {
        String s = "aa";
        String p = "a";

        System.out.println(isMatch(s, p));  // Output: false
    }
}
```

Output Clear

```
false

=== Code Execution Successful ===
```