Experiment 5.1

Student Name: Sanjay Kumar Thakur **UID: 22BCS10790**
Branch: CSE **Section: 642-IOT/B**
Semester: 6th **DOP: 07/04/2025**
Subject: PBLJ **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 alphabeticCharacters.
The methods should return output in a String type variable "output1" which contains all the words from input2 separated by colon which matches input 1. All words in output1 should be in uppercase.

Code:

```
public class ProblemOne {
    public static String matchFound(String input1, String input2) {
        String[] words = input2.split(":");
        StringBuilder output1 = new StringBuilder();
        for (String word : words) {
            if (word.length() != input1.length()) continue;
            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();
    }
```

Problem 2: Encoding Three Strings: Anand was assigned the task of coming up with an encoding mechanism for any given three strings.He has come up with the following plan.

Step ONE :- Given any three strings,break each string into 3 parts each.

For example- if the three strings are below:

Input 1: “John”

Input 2: “Johny”

Input 3 : “Janardhan”

“John” should be split into”J”,”oh”,”n,” as the FRONT, MIDDLE and END part repectivly.

“Johny” should be spilt into “jo”,” h”, “ny” as the FRONT, MIDDLE and END respectively.

“Janardhan” should be split into “Jan” ,” ard” ,”han” as the FRONT, MIDDLE and END part

respectively.

i.e. If the no. of characters in the string are in multiples of 3 ,then each split – part will contain equal no

of characters, as seen in the example of “Janadhan”.

If the no. of characters in the string are NOT in multiples of 3 ,and if there is one character more than

multiple of 3, then the middle part will get the extra character ,as seen in the example of “john”.

If the no. of characters in the string are Not in multiples of 3 and if there are two characters more than

multiple of 3, then the FRONT and END parts will get one extra character each, as seen in the

example of “Johny”.

Step TWO : Concatenate (join) the FRONT ,MIDDLE and END parts of the string as per the below

specified concatenation – rule to from three Output strings.

Output 1: FRONT part of input 1 + MIDDLE part of input 2 +END part of input 3

Output2:- MIDDLE part of input1+ END part of input2 + FRONT part of input3

Output3: END part of the input1+FRONT part of input2 +MIDDLE part of input3

For example , for the above example input strings:

Output1 = “J” +”h”+han”=”jhhan”

Output2 =”oh”+”ny”+”Jan”=”ohnyjan”

Output3=”n”+Jo”+ard”+=“njoard”

Step THREE :-

Process the resulting output strings based on the output-processing rule .After the above two steps, we

will now have three output strings. Further processing is required only for the third output string as

per below rule-

“Toggle the case of each character in the string “ ,i.e. in the third output string, all lower-case

characters should be made upper-case and vice versa.
For example , for the above example strings ,output3 is “nJoard”, so after applying the toggle rule.
Output3 should become “NjOARD”.
Final Result – The three output strings after applying the above three steps i.e. for the above example .
Output1 =”Jnhan”
Output2=”ohnyJan’
Output3 = “NJOARD”
Help Anand to write a program that would do the above.

Code 2:

```
    public static String[] encodeThreeStrings(String input1, String input2, String input3) {
        String[] parts1 = splitIntoThreeParts(input1);
        String[] parts2 = splitIntoThreeParts(input2);
        String[] parts3 = splitIntoThreeParts(input3);
        String output1 = parts1[0] + parts2[1] + parts3[2];
        String output2 = parts1[1] + parts2[2] + parts3[0];
        String output3 = toggleCase(parts1[2] + parts2[0] + parts3[1]);
        return new String[]{output1, output2, output3};
    }

    private static String[] splitIntoThreeParts(String input) {
        int len = input.length();
        int rem = len % 3;
        int base = len / 3;
        int front = base, middle = base, end = base;
        if (rem == 1) middle += 1;
        else if (rem == 2) {
            front += 1;
            end += 1;
        }
        String frontPart = input.substring(0, front);
        String middlePart = input.substring(front, front + middle);
        String endPart = input.substring(front + middle);
        return new String[]{frontPart, middlePart, endPart};
    }

    private static String toggleCase(String input) {
        StringBuilder toggled = new StringBuilder();
        for (char c : input.toCharArray()) {
            if (Character.isUpperCase(c)) {
                toggled.append(Character.toLowerCase(c));
            } else if (Character.isLowerCase(c)) {
                toggled.append(Character.toUpperCase(c));
            } else {
                toggled.append(c);
```

```
            }
        }
        return toggled.toString();
    }

    // Test both problems
    public static void main(String[] args) {
        // Problem 1 Test
        String result1 = matchFound("c_t", "cat:cut:cot:bat:bot:bit");
        System.out.println("Matched Words: " + result1);

String[] encoded = encodeThreeStrings("John", "Johny", "Janardhan");
        System.out.println("Output 1: " + encoded[0]);  // Expected: "Jhhan"
        System.out.println("Output 2: " + encoded[1]);  // Expected: "ohnyJan"
        System.out.println("Output 3: " + encoded[2]);  // Expected: "NJOARD"
    }
}
```

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.

Code 3:

```
public class Problem3 {
```

```
    public static String transform(String input) {
        StringBuilder result = new StringBuilder();
        input = input.toLowerCase();
        for (int i = 0; i < input.length(); i++) {
            char ch1 = input.charAt(i);
            result.append(ch1);

            if (i < input.length() - 1) {
                char ch2 = input.charAt(i + 1);

                if (Character.isLetter(ch1) && Character.isLetter(ch2)) {
                    int val1 = ch1 - 'a' + 1;
                    int val2 = ch2 - 'a' + 1;
                    int sum = val1 + val2;
                    if (sum % 26 == 0) result.append('0');
                    else result.append((char) ('a' + (sum % 26) - 1));
                }
            }
        }
        return result.toString();
    }
    public static void main(String[] args) {
        String input = "ac";
        System.out.println(transform(input));  // Output: adc
    }
}
```

Problem 4: 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"

Code 4:

```
public class Problem4 {
    public static char findExtraChar(String s, String t) {
        int result = 0;
        for (char c : s.toCharArray()) result ^= c;
        for (char c : t.toCharArray()) result ^= c;
        return (char) result;
    }

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

Problem 5:
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 5:

```
import java.util.*;

public class Problem5 {
    public static int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> map = new HashMap<>();
        Stack<Integer> stack = new Stack<>();
        for (int num : nums2) {
            while (!stack.isEmpty() && num > stack.peek()) {
                map.put(stack.pop(), num);
            }
            stack.push(num);
        }
        int[] result = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            result[i] = map.getOrDefault(nums1[i], -1);
        }
        return result;
    }

    public static void main(String[] args) {
        int[] nums1 = {4, 1, 2};
        int[] nums2 = {1, 3, 4, 2};
        System.out.println(Arrays.toString(nextGreaterElement(nums1, nums2)));
// Output: [-1, 3, -1]
    }
}
```

Problem 6:

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

Code 6:

```
import java.util.*;
public class Problem6 {
    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();
        Map<Character, Character> map = Map.of(')', '(', '}', '{', ']', '[');
        for (char c : s.toCharArray()) {
            if (map.containsValue(c)) stack.push(c);
            else if (map.containsKey(c)) {
                if (stack.isEmpty() || stack.pop() != map.get(c)) return false;
            }
        }
        return stack.isEmpty();
    }
    public static void main(String[] args) {
        String[] inputs = {"{}()", "({()})", "{}(", "[]"};
        for (String input : inputs) {
            System.out.println(isBalanced(input)); }
         }
}
```

Problem 7:

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.
Sample Input
5
amy 100
david 100
heraldo 50
aakansha 75
aleksa 150
Sample Output
aleksa 150
amy 100
david 100
aakansha 75
heraldo 50

Code 7:

```
import java.util.*;

class Player {
    String name;
    int score;
    Player(String n, int s) { name = n; score = s; }
}

class Checker implements Comparator<Player> {
    public int compare(Player a, Player b) {
```

```
        if (a.score != b.score) return b.score - a.score;
        return a.name.compareTo(b.name);
    }
}

public class Problem7 {
    public static void main(String[] args) {
        Player[] players = {
            new Player("amy", 100),
            new Player("david", 100),
            new Player("heraldo", 50),
            new Player("aakansha", 75),
            new Player("aleksa", 150)
        };
        Arrays.sort(players, new Checker());
        for (Player p : players) {
            System.out.println(p.name + " " + p.score);
        }
    }
}
```

Problem 8:

knowledge of them!

Given an array, , of real number strings, sort them in descending order — but wait, there's more!

Each number must be printed in the exact same format as it was read from stdin, meaning that is printed as , and is printed as . If two numbers represent numerically equivalent values (e.g.,), then they must be listed in the same order as they were received as input).

You must rearrange array 's elements according to the instructions above.

Input Format

The first line consists of a single integer, , denoting the number of integer strings.

Each line of the subsequent lines contains a real number denoting the value of .

Constraints

• Each has at most digits.

Sample Input

-100

50

56.6
90
0.12
.12
02.34
Sample Output
90
56.6
50
02.34
0.12
.12 0
-100

Code 8:

```
import java.math.BigDecimal;
import java.util.*;

public class Problem8 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());
        String[] values = new String[n];
        for (int i = 0; i < n; i++) values[i] = sc.nextLine();
        Arrays.sort(values, (a, b) -> new BigDecimal(b).compareTo(new
BigDecimal(a)));
        for (String val : values) {
            System.out.println(val);
        }
    }
}
```

Problem 9:

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 '*'.

Code 9:

```
public class Problem9 {
    public static boolean isMatch(String s, String p) {
        int m = s.length(), n = p.length();
        boolean[][] dp = new boolean[m+1][n+1];
        dp[0][0] = true;

        for (int j = 1; j <= n; j++) {
            if (p.charAt(j-1) == '*') dp[0][j] = dp[0][j-1];
        }

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (p.charAt(j-1) == '*')
                    dp[i][j] = dp[i][j-1] || dp[i-1][j];
                else if (p.charAt(j-1) == '?' || s.charAt(i-1) == p.charAt(j-1))
                    dp[i][j] = dp[i-1][j-1];
            }
        }

        return dp[m][n];
    }

    public static void main(String[] args) {
        System.out.println(isMatch("aa", "a"));      // false
```

```
        System.out.println(isMatch("aa", "*"));      // true
        System.out.println(isMatch("cb", "?a"));     // false
    }
}
```

Problem 10:

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 <= nums.length <= 105
• -109 <= nums[i] <= 109
• nums is a non-decreasing array.
• -109 <= target <= 109

Code 10:

```
import java.util.*;

public class Problem10 {
    public static int[] searchRange(int[] nums, int target) {
        int left = binarySearch(nums, target, true);
        int right = binarySearch(nums, target, false);
        return new int[]{left, right};
    }

    private static int binarySearch(int[] nums, int target, boolean findFirst) {
        int index = -1, low = 0, high = nums.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] < target) low = mid + 1;
            else if (nums[mid] > target) high = mid - 1;
            else {
                index = mid;
                if (findFirst) high = mid - 1;
                else low = mid + 1;
            }
```

```
        }
        return index;
    }

    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)));  // [3, 4]
    }
}
```