Lab Assignment

Student Name: Akshat Srivastava
UID: 22BCS11740
Branch: BE CSE
Section/Group: 22BCS_IOT_618_A
Semester: 6th
Set – Fast Learner Even
Subject Name: PBLJ Lab
Subject Code: 22CSH-359

1)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”

Code:

```
import java.util.*;

public class EncodingThreeStrings {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String s1 = sc.nextLine();
      String s2 = sc.nextLine();
      String s3 = sc.nextLine();

      String[] p1 = split(s1);
      String[] p2 = split(s2);
      String[] p3 = split(s3);

      String out1 = p1[0] + p2[1] + p3[2];
      String out2 = p1[1] + p2[2] + p3[0];
      String out3 = toggleCase(p1[2] + p2[0] + p3[1]);
```

```
        System.out.println("Output1: " + out1);
        System.out.println("Output2: " + out2);
        System.out.println("Output3: " + out3);
    }

    static String[] split(String s) {
        int len = s.length(), rem = len % 3;
        int part = len / 3, f = part, m = part, e = part;
        if (rem == 1) m++;
        else if (rem == 2) { f++; e++; }
        String front = s.substring(0, f);
        String middle = s.substring(f, f + m);
        String end = s.substring(f + m);
        return new String[]{front, middle, end};
    }

    static String toggleCase(String s) {
        StringBuilder sb = new StringBuilder();
        for (char c : s.toCharArray()) {
            sb.append(Character.isUpperCase(c) ? Character.toLowerCase(c) : Character.toUpperCase(c));
        }
        return sb.toString();
    }
}
```

Output:

```
PS D:\java lab\assignment> cd "d:\java lab\assignment\"
 EncodingThreeStrings }
John
Jhonny
Joe
Output1: Jone
Output2: ohnyJ
Output3: NjHO
PS D:\java lab\assignment>
```

1) 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:

```
public class FindAddedCharacter {
  public static void main(String[] args) {
    String s = "abcd";
    String t = "abcde";

    System.out.println("Added character: " + findAddedChar(s, t));
  }

  static char findAddedChar(String s, String t) {
    int sumS = 0, sumT = 0;
    for (char c : s.toCharArray()) sumS += c;
    for (char c : t.toCharArray()) sumT += c;
    return (char)(sumT - sumS);
  }
}
```

Output:

```
PS D:\java lab\assignment> cd
indAddedCharacter }
Added character: e
```

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

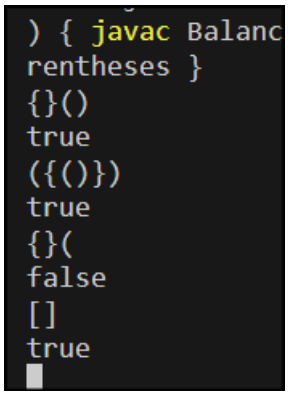
true true false true

Code:

```
import java.util.*;

public class BalancedParentheses {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String line = sc.nextLine();
            System.out.println(isBalanced(line));
        }
        sc.close();
    }

    static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else if (c == ')' && !stack.isEmpty() && stack.peek() == '(') {
                stack.pop();
            } else if (c == '}' && !stack.isEmpty() && stack.peek() == '{') {
                stack.pop();
            } else if (c == ']' && !stack.isEmpty() && stack.peek() == '[') {
                stack.pop();
            } else {
                return false;
            }
        }
        return stack.isEmpty();
```

```
    }
}
```

Output:



4) Java's BigDecimal class can handle arbitrary-precision signed decimal numbers. Let's test your 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

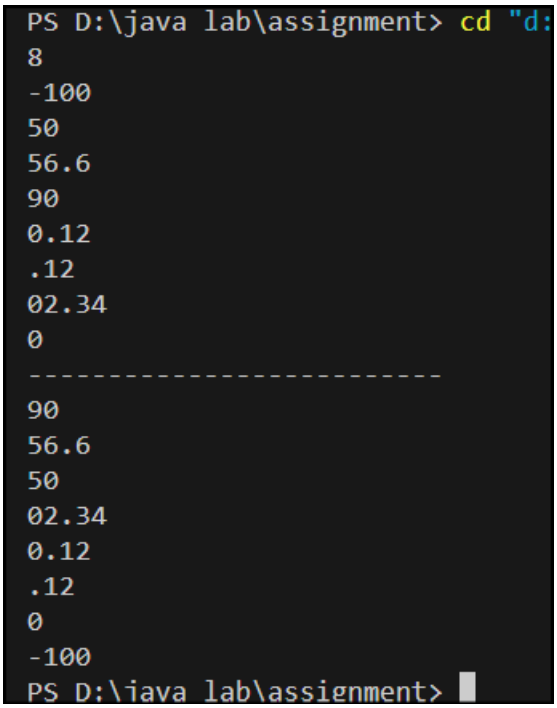
90
56.6
50
02.34
0.12
.12 0
-100

Code:

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

public class BigDecimalSort {
   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) -> {
         BigDecimal bd1 = new BigDecimal(a);
         BigDecimal bd2 = new BigDecimal(b);
         return bd2.compareTo(bd1);
      });
      System.out.print("-------------------------\n");
      for (String val : values) {
         System.out.println(val);
      }

      sc.close();
   }
```

```
}
```

Output:



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

Code:

```
public class FindFirstAndLastPosition {
    public static int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};
        result[0] = findIndex(nums, target, true);
        result[1] = findIndex(nums, target, false);
        return result;
    }

    private static int findIndex(int[] nums, int target, boolean findFirst) {
        int start = 0, end = nums.length - 1, index = -1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (nums[mid] == target) {
                index = mid;
                if (findFirst) end = mid - 1;
                else start = mid + 1;
            } else if (nums[mid] < target) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return index;
    }

    public static void main(String[] args) {
        int[] nums = {5, 7, 7, 8, 8, 10};
        int target = 8;
        int[] result = searchRange(nums, target);
        System.out.println("[" + result[0] + "," + result[1] + "]");
    }
}
```

Output: