Shabari - SDE - Assessment 1

1. Running Sum of 1d Array

Given an array nums. We define a running sum of an array as runningSum[i] = sum(nums[0]…nums[i]). Return the running sum of nums.

Analysis:

- The problem is simple, we need to calculate the running sum of the array.
- We can solve this problem using a simple loop.

Code:

```java
package com.assessment_i;

import java.util.Scanner;

public class RunningSumOf1DArray {
    private static void runningSumOf1DArray(int[] arr){
        for(int i=1;i<arr.length; i++){
            arr[i] += arr[i-1];
        }
        System.out.print("The running sum of 1D array: ");
        for(int num: arr){
            System.out.print(num+" ");
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        runningSumOf1DArray(arr);
```

```
    }
}
```

Output:

```
Enter the size of the array:
5
Enter the elements of the array:
1 2 3 4 5
The running sum of 1D array: 1 3 6 10 15
```

---

2. ATM Withdrawal Transaction Calculator

Pooja would like to withdraw X $US from an ATM. The cash machine will only accept the transaction if X is a multiple of 5, and Pooja's account balance has enough cash to perform the withdrawal transaction (including bank charges). For each successful withdrawal, the bank charges 0.50 $US. Calculate Pooja's account balance after an attempted transaction.

Analysis:

- The problem is simple, we need to calculate the account balance after the withdrawal transaction.
- We can solve this problem using a simple if-else condition.

Code:

```java
package com.assessment_i;

import java.util.Scanner;

public class ATMWithdrawalTransactionCalculator {
    private static boolean verifyWithdraw(double wd, double bb){
        if (wd < bb)
            return (wd % 5) == 0;
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the withdraw amount: $");
```

```
        double withdraw = sc.nextInt();
        System.out.print("Enter the initial bank balance: $");
        double bankBalance = sc.nextInt();
        if (verifyWithdraw(withdraw,bankBalance)){
            System.out.println("Bank charges: 0.50 $US (for the transaction)");
            System.out.println(
                "After the successful withdrawal and bank charge deduction: $"
                + (bankBalance - withdraw - 0.50));
        } else {
            System.out.println("The withdraw amount is not divisible by 5");
            System.out.println("Bank Balance: $" + bankBalance);
        }
    }
}
```

Output:

```
Enter the withdraw amount: $20
Enter the initial bank balance: $100
Bank charges: 0.50 $US (for the transaction)
After the successful withdrawal and bank charge deduction: $79.5
```

---

3. Remove Consecutive Duplicates

For a given string(str), remove all the consecutive duplicate characters.

Analysis:

- The problem is simple, we need to remove the consecutive duplicate characters from the string.
- We can solve this problem using a simple loop.

Code:

```
package com.assessment_i;

import java.util.Scanner;

public class RemoveConsecutiveDuplicates {
```

```
    private static void removeConsecutiveDuplicates(String str){
        StringBuilder sb = new StringBuilder();
        sb.append(str.charAt(0));
        for(int i=1; i<str.length(); i++){
            if (str.charAt(i) != str.charAt(i-1)){
                sb.append(str.charAt(i));
            }
        }
        System.out.println(
            "The string after removing consecutive duplicates: "
            +sb);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = sc.nextLine();
        removeConsecutiveDuplicates(str);
    }
}
```

Output:

```
Enter the string: aaaa
The string after removing consecutive duplicates: a
```

---

4. Length of the longest substring

Given a string S, find the length of the longest substring without repeating characters.

Analysis:

- The problem is simple, we need to find the length of the longest substring without repeating characters.
- We can solve this problem using a simple loop.

Code:

```java
package com.assessment_i;

import java.util.Scanner;

public class LengthOfTheLongestSubstring {
    private static int lengthOfTheLongestSubstring(String str){
        int[] arr = new int[256];
        int i=0, j=0, max = 0;
        while (j < str.length()){
            if (arr[str.charAt(j)] == 0){
                arr[str.charAt(j)] = 1;
                j++;
                max = Math.max(max, j-i);
            } else {
                arr[str.charAt(i)] = 0;
                i++;
            }
        }
        return max;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = sc.nextLine();
        System.out.println(
            "The length of the longest substring without repeating characters: "
            +lengthOfTheLongestSubstring(str));
    }
}
```

Output:

```
Enter the string: shabari
The length of the longest substring without repeating characters: 4
```

---

5. The Celebrity Problem

A celebrity is a person who is known to all but does not know anyone at a party. If you go to a party of N people, find if there is a celebrity in the party or not. A square NxN matrix M[][] is used to represent people

at the party such that if an element of row i and column j is set to 1 it means ith person knows jth person. Here M[i][i] will always be 0. Return the index of the celebrity, if there is no celebrity return -1. Note: Follow 0 based indexing.

Analysis:

- The problem is simple, we need to find the celebrity in the party.
- We can solve this problem using a simple loop.

Code:

```
package com.assessment_i;

import java.util.Scanner;

public class TheCelebrityProblem {
    private static int findCelebrity(int[][] arr){
        int n = arr.length;
        int i=0, j=n-1;
        while (i < j){
            if (arr[i][j] == 1){
                i++;
            } else {
                j--;
            }
        }
        for (int k=0; k<n; k++){
            if (k != i && (arr[i][k] == 1 || arr[k][i] == 0)){
                return -1;
            }
        }
        return i;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of people in the party: ");
        int n = sc.nextInt();
        int[][] arr = new int[n][n];
        System.out.println("Enter the matrix: ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                arr[i][j] = sc.nextInt();
```

```
            }
        }
        int celebrity = findCelebrity(arr);
        if (celebrity == -1){
            System.out.println("No celebrity found");
        } else {
            System.out.println("Celebrity found at index: "+celebrity);
        }
    }
}
```

Output:

```
Enter the number of people in the party: 4
Enter the matrix:
0 0 1 0
0 0 1 0
0 0 0 0
0 0 1 0
Celebrity found at index: 2
```

---

6. Valid String

You have been given a string 'S' containing only three types of characters, i.e. '(', ')' and '*'. A Valid String is defined as follows:

- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string.
- An empty string is also valid. Your task is to find out whether the given string is a Valid String or not.

Analysis:

- The problem is simple, we need to find whether the given string is a valid string or not.
- We can solve this problem using a simple loop.

Code:

```
package com.assessment_i;

import java.util.Scanner;

public class ValidString {
    private static boolean isValidString(String str){
        int low = 0, high = 0;
        for (char c: str.toCharArray()){
            low += c == '(' ? 1 : -1;
            high += c != ')' ? 1 : -1;
            if (high < 0) break;
            low = Math.max(low, 0);
        }
        return low == 0;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = sc.nextLine();
        if (isValidString(str)){
            System.out.println("The string is a valid string");
        } else {
            System.out.println("The string is not a valid string");
        }
    }
}
```

Output:

```
Enter the value of n: 3
Enter the element:
*())
(*)
())*
Yes
Yes
No
```