Rajalakshmi Engineering College

Name: Nivedhitha K
Email: 240701371@rajalakshmi.edu.in
Roll no: 240701371
Phone: 9790413580
Branch: REC
Department: CSE - Section 10
Batch: 2028
Degree: B.E - CSE

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 9_CY

Attempt : 2
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

Aarav is developing a music playlist application where users can manage their favorite songs. He wants to implement a feature that allows users to reorder the playlist by moving a song from one position to another.

You need to implement a function that performs the following operations using a LinkedList:

Add songs to the playlist in the given order.Move a song from a specified position to another position in the playlist.Print the final playlist after all operations.

Input Format

The first line of the input consists of an integer n representing the number of songs.

The next n lines, each containing a string representing a song name.

After the songs are given the next line contains an integer m, the number of move operations.

The next m lines, each containing two integers x and y representing the move operation where the song at position x (0-based index) should be moved to position y.

Output Format

The output prints the final playlist, each song on a new line.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
SongA
SongB
SongC
SongD
SongE
2
2 4
0 3
Output: SongB
SongD
SongE
SongA
SongC

Answer

```
// You are using Java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        int n = Integer.parseInt(sc.nextLine().trim());
        LinkedList<String> playlist = new LinkedList<>();

        // Add songs to playlist
        for (int i = 0; i < n; i++) {
            playlist.add(sc.nextLine().trim());
        }

        int m = Integer.parseInt(sc.nextLine().trim());

        // Perform move operations
        for (int i = 0; i < m; i++) {
            int x = sc.nextInt();
            int y = sc.nextInt();

            // Remove the song at position x
            String song = playlist.remove(x);

            // Insert it at position y
            playlist.add(y, song);
        }

        // Print final playlist
        for (String song : playlist) {
            System.out.print(song + "   ");
        }

        sc.close();
    }
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Sanjay is working on a program to merge two sorted linked lists into a single sorted list using Java's LinkedList class from the Collections framework. Given two sorted linked lists, he wants to merge them while maintaining the sorted order.

Write a Java program that:

Reads two sorted linked lists.Merges them into a single sorted linked list.Prints the merged list in ascending order.

Input Format

The first line contains an integer m (the size of the first linked list).

The second line contains m space-separated integers (sorted).

The third line contains an integer n (the size of the second linked list).

The fourth line contains n space-separated integers (sorted).

Output Format

The output prints the merged linked list as space-separated integers.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
5 10
3
1 3 8
Output: 1 3 5 8 10

Answer

```
import java.util.*;
class MergeSortedLinkedLists {

public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);

   int m = sc.nextInt();
   LinkedList<Integer> list1 = new LinkedList<>();
   for (int i = 0; i < m; i++) {
      list1.add(sc.nextInt());
   }
```

```
    int n = sc.nextInt();
    LinkedList<Integer> list2 = new LinkedList<>();
    for (int i = 0; i < n; i++) {
        list2.add(sc.nextInt());
    }

    // Merge both lists
    LinkedList<Integer> merged = new LinkedList<>();
    merged.addAll(list1);
    merged.addAll(list2);

    // Sort merged list
    Collections.sort(merged);

    // Print merged list
    for (int i = 0; i < merged.size(); i++) {
        System.out.print(merged.get(i) + " ");
    }

    sc.close();
}


}
```

Status : Correct ***Marks : 10/10***

3. Problem Statement

Rahul is working on a list manipulation problem where he needs to reverse a specific subarray using a stack. Given an array and two indices l and r, he wants to reverse only the portion of the array from index l to r (both inclusive) while keeping the rest of the array unchanged.

Since Rahul wants to solve this problem efficiently, he decides to use a stack to reverse the subarray in O(r - l) time.

Your task is to help Rahul by implementing this functionality.

Input Format

The first line contains an integer n, the size of the array.

The second line contains n space-separated integers arr[i].

The third line contains two integers l and r, denoting the start and end indices of the subarray to reverse.

Note: The array follows 0-based indexing.

Output Format

The output prints the modified array after reversing the subarray between indices l and r.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 6
1 2 3 4 5 6
1 4
Output: 1 5 4 3 2 6

Answer

```
import java.util.*;

public class Main {

public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);

   int n = sc.nextInt();
   int[] arr = new int[n];
   for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
   }

   int l = sc.nextInt();
   int r = sc.nextInt();

   Stack<Integer> stack = new Stack<>();
```

```
    // Push elements from l to r onto the stack
    for (int i = l; i <= r; i++) {
        stack.push(arr[i]);
    }

    // Pop elements from stack back into the array (reversing that portion)
    for (int i = l; i <= r; i++) {
        arr[i] = stack.pop();
    }

    // Print final array
    for (int i = 0; i < n; i++) {
        System.out.print(arr[i] + " ");
    }

    sc.close();
}

}
```

Status : Correct **Marks : 10/10**

4. Problem Statement

A teacher is filtering a list of words provided by students. Some words contain too many vowels, making them difficult for a spelling competition. The teacher decides to remove all words that contain more than two vowels.

Help the teacher to implement it using ArrayList.

Input Format

The first line contains an integer N, representing the number of words in the list.

The next N lines contain a string representing the words (one per line).

Output Format

The output consists of words that contain two or less than two vowels, printed in the same order they appeared in the input. Each word is printed on a new line.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 1
sri

Output: sri

Answer

```
import java.util.ArrayList;
import java.util.Scanner;

class VowelFilter
{
   public static void filterWords(int n,Scanner sc)
   {
      ArrayList<String> words=new ArrayList<>();
      for(int i=0;i<n;i++)
      {
         words.add(sc.nextLine());
      }
      for(String word:words)
      {
         if(countVowels(word)<=2)
         {
            System.out.println(word);
         }
      }
   }
   private static int countVowels(String word)
   {
      int count=0;
      String vowels="aeiouAEIOU";
      for(char ch:word.toCharArray())
      {
         if(vowels.indexOf(ch)!=-1)
         {
            count++;
         }
```

```
        }
        return count;

    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        VowelFilter.filterWords(n, sc);
        sc.close();
    }
}
```

Status : Correct **Marks : 10/10**