Faculty of Engineering, University of Jaffna,

Department of Computer Engineering

EC4070: Data Structures and algorithms

Lab 03

Chapter 3: Linear abstract data types

Duration: 3 Hours Lecturer: Ms. Sujanthika M.

Instructions

i. Submit the code files and screenshot of the outputs in a zipped folder by naming as 2022EAAA_Lab03(AAA – Your Registration Number)
ii. Submit your zip file before the given deadline.
iii. Any plagiarized work will be given 0 marks.

Question 1: Balanced Bracket Validator

You are tasked with designing a program to validate expressions containing brackets. The brackets can be of the following types:

- Round brackets: ( )
- Square brackets: [ ]
- Curly brackets: { }

An expression is considered balanced if:

1. Every opening bracket has a corresponding closing bracket of the same type.
2. Brackets close in the correct order (e.g., {[()]} is valid, but {[(])} is not).

You must use a stack data structure to implement the solution.

Input Format

1. The first line contains an integer t (1 ≤ t ≤ 10) — the number of test cases.
2. Each of the next t lines contains a string s (1 ≤ |s| ≤ 10⁵), representing the bracket expression.

Output Format

For each test case, output YES if the expression is balanced, otherwise output NO.

Sample Input:

3

{[()]}

{[(])}

{{[[(())]]}}

Sample Output:

YES

NO

YES

Question 2: Train Route Manager

You are building a Train Route Manager to manage the stops on a train's route using a doubly linked list. Each stop has the following details:

- Stop ID: A unique integer representing the stop.
- Stop Name: A string representing the name of the stop.

Your program should support the following operations:

1. Add Stop: Add a new stop to the route at a specific position.
    a. If the position is 1, the stop becomes the first stop.
    b. If the position is greater than the number of stops, the stop is added to the end of the route.
2. Remove Stop: Remove a stop by its Stop ID.
3. Find Stop by ID: Find a stop by its Stop ID and display its details.

4. Reverse Route: Reverse the order of stops in the route.
5. Display Route: Display the complete route from the first to the last stop.

Input Format

1. The first line contains an integer `t` (1 ≤ t ≤ 5) — the number of test cases.
2. For each test case:
    a. The first line contains an integer n (1 ≤ n ≤ 20) — the number of initial stops on the route.
    b. The next n lines contain details of each stop in the format:
    `StopID StopName`
    c. The next line contains an integer k (1 ≤ k ≤ 50) — the number of operations to perform.
    d. Each of the next k lines contains an operation in one of the following formats:
        i. `1 StopID StopName Position` (Add Stop)
        ii. `2 StopID` (Remove Stop)
        iii. `3 StopID` (Find Stop by ID)
        iv. 4 (Reverse Route)
        v. 5 (Display Route)

Output Format

For each test case:

- For operation 3, output the Stop Name and Stop ID in the format:
  `StopID StopName`
  If the stop is not found, output `Stop not found`.
- For operation 5, output the route in the format:
  `StopID:StopName -> StopID:StopName -> ....`

Sample Input:

```
1
3
1 Start
2 Midway
3 End
6
1 4 StationA 2
```

2 3
3 4
4
5
5

Sample Output:

4 StationA
2:Midway -> 4:StationA -> 1:Start
1:Start -> 4:StationA -> 2:Midway

Explanation:

1. Initially, the route is `1:Start -> 2:Midway -> 3:End`.
2. Adding Stop `4:StationA` at position 2 changes the route to:
   `1:Start -> 4:StationA -> 2:Midway -> 3:End`.
3. Removing Stop 3 updates the route to:
   `1:Start -> 4:StationA -> 2:Midway`.
4. Finding Stop 4 outputs its details: `4 StationA`.
5. Reversing the route changes it to:
   `2:Midway -> 4:StationA -> 1:Start`.
6. Displaying the route again shows the reversed order.