

Part 1 — Linked List Basics

1. Create a Singly Linked List

- Create a linked list with 3 nodes [10, 20, 30] and print all elements.

2. Add a Node at the Beginning

- Insert 5 at the start → [5, 10, 20, 30].

3. Add a Node at the End

- Append 40 → [5, 10, 20, 30, 40].

4. Delete the First Node

- Remove head → [10, 20, 30, 40].

5. Search for a Value

- Check if 20 exists → return true/false.
-

Part 2 — Queue Basics Using Linked Lists

1. Implement a Queue

- Use a linked list to create a queue class with enqueue, dequeue, front, rear, isEmpty.

2. Enqueue and Dequeue

- Enqueue [10, 20, 30], dequeue 1 element, print the queue.

3. Check if Queue is Empty

- Test before and after enqueueing items.

4. Get Front and Rear Values

- Print the front and rear of a queue after adding [5, 15, 25].

5. Reverse a Small Queue

- Reverse a queue [1, 2, 3] using a temporary stack/array.
-

Part 3 — Stack Basics Using Arrays or Linked Lists

1. Implement a Stack

- Use an array or linked list to create a stack class with `push(x)`, `pop()`, `top()`, `isEmpty()`, and `size()`.

2. Push and Pop

- Push [10, 20, 30] onto the stack, then pop 1 element.
- Print the stack after each operation.

3. Check if Stack is Empty

- Test the stack before and after adding items using `isEmpty()`.

4. Get Top Element

- Print the top element after pushing [5, 15, 25].
- Pop and check the top again.

5. Reverse a Small Stack

- Reverse a stack [1, 2, 3] using a temporary stack or array.
- Print the reversed stack.

Problems to Solve

1. Valid Parentheses

Problem:

Given a string containing only '(', ')', '{', '}', '[', ']', determine if the string is valid.

A valid string must have brackets closed in the correct order.

Example:

Input: `s = "()[]{}"`

Output: `true`

Input: `s = "()"`

Output: `false`

2. Min Stack

Problem:

Design a stack that supports:

- `push(x)`
 - `pop()`
 - `top()`
 - `getMin() → returns the minimum element in O(1)`
-

3. Implement Queue Using Stacks

Problem:

Implement a queue using **two stacks**.

Operations:

- `push(x)`
 - `pop() → removes front`
 - `peek() → returns front`
 - `empty()`
-

4. Evaluate Reverse Polish Notation (RPN)

Problem:

Evaluate an expression in **Reverse Polish Notation**.

Valid operators: + - * /.

Example:

Input: ["2", "1", "+", "3", "*"]

Output: 9

Explanation: (2 + 1) * 3

5. Backspace String Compare

Problem:

Given two strings s and t , return `true` if they are equal after interpreting `#` as a backspace.

Example:

Input: "ab#c", "ad#c"

Output: true
Both become "ac".

6. Remove Adjacent Duplicates

Problem:

Given a string s , repeatedly remove **adjacent pairs** of equal characters.

Example:

Input: "abbaca"
Output: "ca"