Sheryians Coding School

Live Cohort

Day 96

Topic Name: Singly Linked List in JavaScript

🔹 Definition

A Singly Linked List is a linear data structure where each element (node) points to the next one. It allows dynamic memory allocation and is useful when frequent insertions and deletions are required.

Each node contains:

- val: the data.
- next: a pointer to the next node.

🔹 Code (Example)

```
class Node {
    constructor(val) {
        this.val = val;
        this.next = null;
    }
}

class LinkedList {
    constructor() {
        this.head = null;
        this.size = 0;
    }
```

```
insertAtFirst(val){
    this.size++;
    const newNode = new Node(val);
    if(this.head === null){
        this.head = newNode;
        return;
    }
    newNode.next = this.head;
    this.head = newNode;
}

inserAtLast(val){
    this.size++;
    const newNode = new Node(val);
    if(this.head == null){
        this.head = newNode;
        return;
    }
    let temp = this.head;
    while(temp.next != null){
        temp = temp.next;
```

```
        }
        temp.next = newNode;
    }
```

```
deleteAtFirst(){
    if(this.head === null){
        console.log("Empty list");
        return;
    }
    this.size--;
    this.head = this.head.next;
}

deleteAtLast(){
    if(this.head === null){
        console.log("empty list");
        return;
    }
    this.size--;
    if(this.head.next == null){
        this.head = null;
        return;
```

```
        }
        this.size--;
        if(this.head.next == null){
            this.head = null;
            return;
        }
        let temp = this.head;
        while(temp.next.next != null){
            temp = temp.next;
        }
        temp.next = null;
    }
```

```
    printLL(){
        if(this.head === null){
            console.log("empty list");
            return;
        }
        let temp = this.head;
        while(temp != null){
            process.stdout.write(temp.val + "->");
            temp = temp.next;
        }
        console.log("null");
    }
}
```

Use Case

- Efficient insertions/deletions from the start or middle.
- Used in stacks, queues, graph adjacency lists.
- Dynamic memory usage (unlike arrays).

```
let obj = new LinkedList();
obj.insertAtFirst(10);
obj.insertAtFirst(20);
obj.insertAtFirst(30);
obj.inserAtLast(100);
obj.printLL(); // Output: 30->20->10->100->null
obj.deleteAtLast();
obj.printLL(); // Output: 30->20->10->null
```

Interview Q&A

Q1: What is a Linked List?

A: A linked list is a linear data structure where elements are stored in nodes, and each node points to the next one. Unlike arrays, memory is not contiguous.

Q2: Difference between insertAtFirst and insertAtLast?

A:

- insertAtFirst: Adds node at the beginning (O(1)).
- insertAtLast: Adds node at the end (O(n)).

Q3: How to delete the last node from a singly linked list?

A: Traverse to the second last node and set its next to null.

Q4: What is the time complexity of operations in a singly linked list?

Operation	Time Complexity
insertAtFirst	O(1)
inserAtLast	O(n)
deleteAtFirst	O(1)
deleteAtLast	O(n)
printLL	O(n)

Q5: When would you prefer a linked list over an array?

A: When frequent insertions/deletions are required, especially from the beginning or middle of the list.