Introduction to Linked List

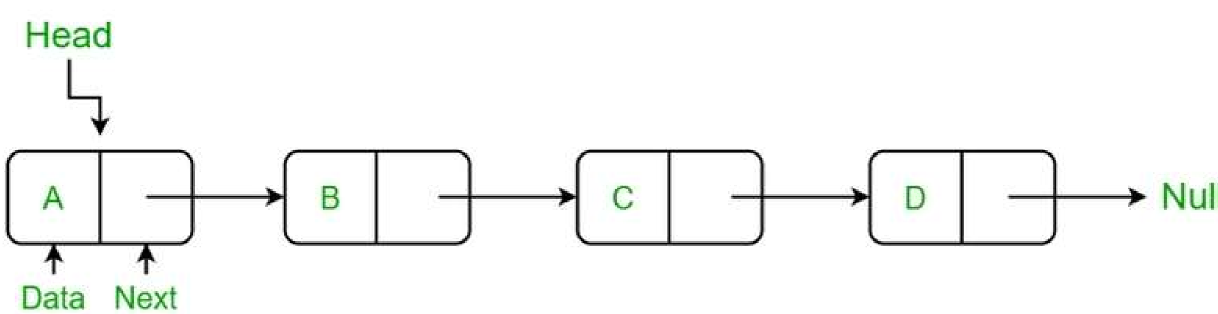
1. Definition

- A **Linked List** is a **linear data structure** where elements (called **nodes**) are stored at **non-contiguous memory locations**, and each node is connected using a **pointer/reference**.
- Unlike arrays, linked lists do not require **contiguous memory allocation**.

Each **node** has two parts:

1. **Data** → stores the value.
2. **Pointer (next)** → stores the address of the next node.

2. Linked List Representation

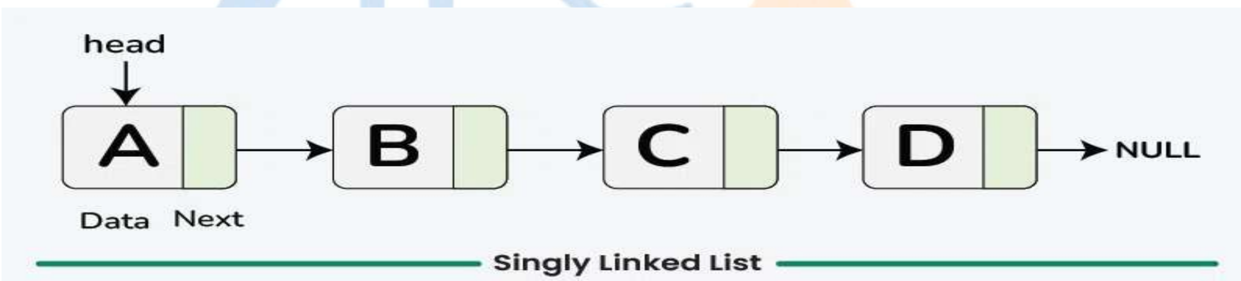


3. Why Linked List?

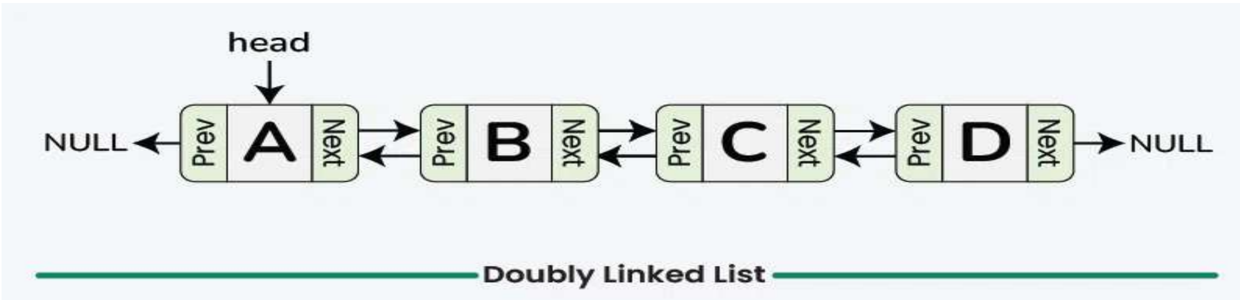
- Arrays have fixed size → difficult to resize.
- Insertion/Deletion in arrays is costly (O(n)).
- Linked list overcomes these issues by using **dynamic memory allocation**.

4. Types of Linked Lists

1. **Singly Linked List**
   - Each node points to the next node.
   - Last node points to `NULL`.



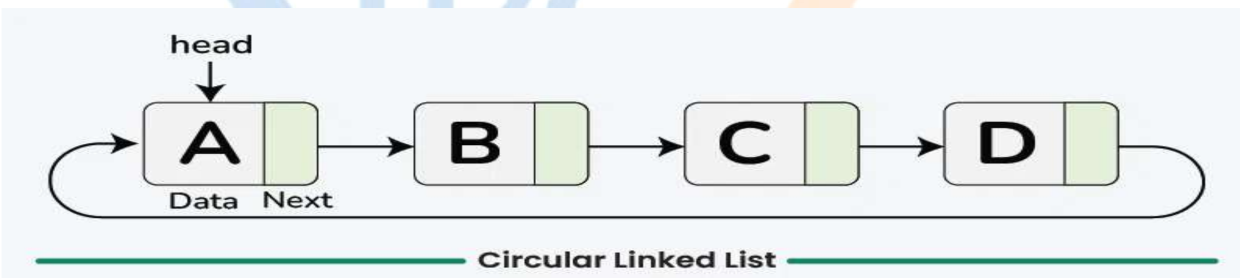
2. **Doubly Linked List**
   - Each node has two pointers: `prev` and `next`.
   - Can traverse in both directions.



3. **Circular Linked List**

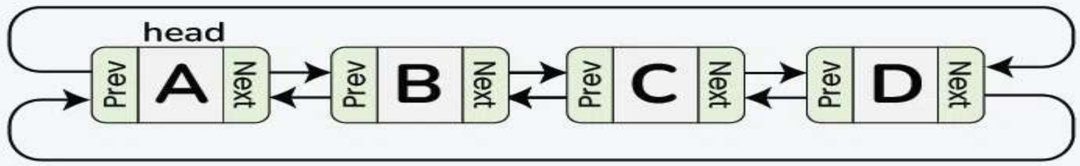
    ○ Last node points back to the first node.

    ○ Can be **singly** or **doubly circular**.



4. Doubly Circular Linked List

- Each node has **two pointers**: `prev` and `next`.
- The **last node's next points to the first node**, and the **first node's prev points to the last node**.
- Allows traversal in **both directions**, circularly.



Doubly Circular Linked List

5. Basic Operations

1. **Traversal** → Visiting each node from head to end.
2. **Insertion** → Add node at beginning, end, or specific position.
3. **Deletion** → Remove node from beginning, end, or specific position.
4. **Searching** → Find an element by value.
5. **Updation** → Update data of a node.

6. Advantages of Linked List

Dynamic size (can grow/shrink at runtime).
Efficient insertion and deletion (O(1) at head).
Memory is utilized efficiently (no wastage like arrays).

7. Disadvantages of Linked List

Extra memory for storing pointers.
Sequential access only (no direct/random access like arrays).
More complex implementation.
Cache unfriendly (not stored in contiguous memory).

8. Time Complexity

Operation	Singly Linked List	Doubly Linked List
Traversal	O(n)	O(n)
Insertion (start)	O(1)	O(1)
Insertion (end)	O(n)	O(1) if tail given
Deletion (start)	O(1)	O(1)
Deletion (end)	O(n)	O(1) if tail given
Searching	O(n)	O(n)

9. Applications

- Implementing **stacks and queues**.
- Dynamic memory management.
- Undo/Redo functionality in editors.
- Browser history (forward & backward).
- Polynomial arithmetic.

Real-Life Examples of Linked List

1. Music / Video Playlist (Circular Linked List)

- In a music player, songs are stored like nodes.
- After the last song, the player continues from the **first song** (circular).

- You can also move **forward (next song)** and **backward (previous song)** → like **Doubly Circular Linked List**.

2. Web Browser History (Doubly Linked List)

- When you browse, each page is a node.
- You can click **Back** (move to prev) and **Forward** (move to next).
- Doubly Linked List is best suited.

3. Undo / Redo in Text Editors (Doubly Linked List)

- Every action (typing, deleting) is stored as a node.
- **Undo** → move to the previous state.
- **Redo** → move to the next state.

4. Train Coaches (Singly / Doubly Linked List)

- Each coach in a train is linked to the **next coach**.
- If connections exist in **both directions** (between two coaches), it resembles a **Doubly Linked List**.

5. Round-Robin Scheduling in Operating System (Circular Linked List)

- CPU allocates time to processes in a round-robin manner.
- After the last process, CPU returns to the **first process**.
- This is exactly how a **Circular Linked List** works.

6. Image Slideshow Viewer (Circular Linked List)

- In a slideshow, after the last image, it returns to the **first image**.
- Can move **next** or **previous** → implemented using **Circular Doubly Linked List**.

7. Blockchain (Singly Linked List)

- Each block stores **data + pointer to the next block (hash of previous)**.
- Forms a chain → exactly like a **singly linked list**.

8. Polynomials / Sparse Matrices (Linked List Representation)

- Polynomials like 5x^3 + 4x^2 + 2 can be stored in linked lists → each node has **coefficient & power**.
- Sparse matrices are efficiently stored using linked lists instead of 2D arrays.

So, whenever you explain linked lists in class, you can say:

- **Playlist → Circular Linked List**
- **Browser History → Doubly Linked List**
- **Undo/Redo → Doubly Linked List**
- **Round Robin Scheduling → Circular Linked List**
- **Blockchain → Singly Linked List**