Linked List – Theoretical Explanation

A linked list is a linear data structure in which elements, called nodes, are not stored in contiguous memory locations.
Each node contains two parts: data and a reference (or link) to the next node in the sequence.
Unlike arrays, linked lists allow dynamic memory allocation, making insertion and deletion operations efficient.

Basic Structure of a Linked List
Each node consists of:
- Data: stores the actual value
- Link: stores the address of the next node

Types of Linked Lists
1. Singly Linked List: Each node points to the next node, and the last node points to NULL.
2. Doubly Linked List: Each node has two links, one to the previous node and one to the next node.
3. Circular Linked List: The last node points back to the first node, forming a circle.

Traversing a Linked List
Traversing means visiting each node of the linked list exactly once to access or display its data.

Types of Traversing
1. Forward Traversing (Singly & Doubly Linked List):
Starting from the head node, move to the next node using the link until NULL is reached.

2. Backward Traversing (Doubly Linked List):
Starting from the last node, move backward using the previous link until the first node is reached.

3. Circular Traversing (Circular Linked List):
Start from any node and continue moving through the list until the starting node is reached again.

Advantages of Linked List
- Dynamic size
- Efficient insertion and deletion
- Better memory utilization compared to arrays

Disadvantages of Linked List
- Extra memory for storing pointers
- No direct access to elements (no indexing)
- Traversal can be time-consuming

Conclusion
Linked lists are powerful data structures widely used in dynamic memory management, stacks, queues,
and real-time applications where frequent insertions and deletions are required.