Linked List

Introduction

- A **linked list** is a linear data structure consisting of nodes connected in sequence.
- Each node has two parts: **data** and **address** (pointer to the next node).
- The last node points to **null**.
- Linked list is the second most used data structure after arrays.

Representation of Linked List

- Linked list nodes are not stored contiguously; each node points to the next node.

Why Linked List Over Array?

Limitations of Arrays:

- Size must be fixed in advance.
- Increasing size at runtime is time-consuming.
- Elements need to be stored contiguously, causing shifts during insertion/deletion.

Advantages of Linked List:

- **Dynamic memory allocation**; nodes are stored non-contiguously.
- Size is flexible; grows as needed.

Declaring a Linked List

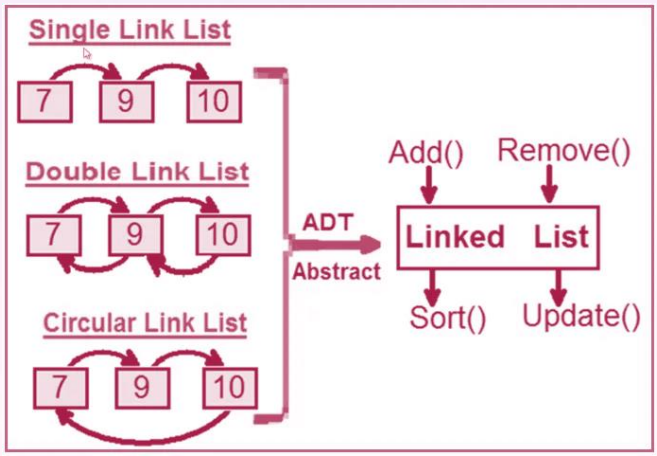
- Linked list requires a **structure** to store data and pointers.

Example declaration:

```
struct node {
  int data;
  struct node *next;
};
```

Types of Linked Lists

1. **Singly Linked List**: Each node points to the next node.
2. **Doubly Linked List**: Each node points to both the next and the previous node.
3. **Circular Singly Linked List**: The last node points to the first node.
4. **Circular Doubly Linked List**: Both first and last nodes point to each other.



Advantages of Linked List

- **Dynamic size**.
- **Efficient insertion/deletion** (no shifting required).
- **Memory efficient**; grows or shrinks as needed.
- Useful for implementing **stacks**, **queues**, etc.

Disadvantages of Linked List

- **Memory usage**: Each node stores extra pointer information.
- **Traversal**: Random access is not possible (must traverse sequentially).
- **Reverse traversal**: Difficult in singly linked lists.

Applications of Linked List

- Representing **polynomials**, **sparse matrices**.
- **Stacks**, **queues**, and **trees** can be implemented using linked lists.
- **Graphs** can be represented as adjacency lists using linked lists.
- **Dynamic memory allocation** at runtime.

Operations on Linked List

- **Insertion**: Add a new element.
- **Deletion**: Remove an element.
- **Display**: Show elements of the list.
- **Search**: Find an element using a key.

Time Complexity

- **Insertion**: O(1)
- **Deletion**: O(1)
- **Search**: O(n)

Space Complexity

- **Insertion**: O(n).
- **Deletion**: O(n).
- **Search**: O(n).