Linked list _Intro

📘 LINKED LIST – COMPLETE DOCUMENT (C & C++)

🔷 1️⃣ Introduction to Linked List

A **Linked List** is a **linear data structure** where elements (called **nodes**) are **not stored in contiguous memory locations**.

Each node contains **data** and a **pointer (or reference)** to the next node.

It forms a **chain-like connection** between nodes.

🧠 Real-life analogy:

Imagine a **chain of people**, each holding the hand of the next — if one leaves, the rest remain connected.

Similarly, in a linked list, each node knows where the next one is.

🔷 2️⃣ Drawbacks of Arrays

Although arrays are simple and efficient, they have several **limitations**:

Limitation	Explanation
Fixed Size	Size must be known at compile time. Cannot be changed during runtime.
Contiguous Memory	Requires continuous memory locations, may cause allocation failure.
Costly Insertion/Deletion	To insert or delete elements, other elements must be shifted.
Wasted or Insufficient Memory	Allocating extra memory wastes space; allocating less causes overflow.

✅ **Linked List** solves all these issues.

🔷 3️⃣ Properties of Linked List

1. **Dynamic in nature** – can grow or shrink at runtime.
2. **Non-contiguous storage** – elements stored anywhere in memory.
3. **Sequential access** – elements accessed in order, not randomly like arrays.
4. **Flexible memory usage** – no pre-defined size required.
5. **Insertion and Deletion** – efficient (O(1)) if the position is known.
6. **Extra memory** – each node requires an extra pointer field.

🔷 4️⃣ Need for Linked List

We need linked lists when:

1. Size of data is not known in advance.
2. Frequent insertion/deletion operations are required.
3. Continuous memory allocation (arrays) is not possible.
4. Dynamic data structures like stacks, queues, graphs, and trees must be implemented efficiently.

🔷 5️⃣ Structure of a Node

Each node in a linked list consists of:

1. **Data** — value stored in the node.
2. **Pointer (next)** — address of the next node in the list.

🔶 In C:

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

🔶 In C++:

```
class Node {
public:
    int data;
    Node* next;

    Node(int val) {
        data = val;
        next = nullptr;
    }
};
```

🔷 6️⃣ Memory Representation of Linked List

A linked list stores data in separate memory blocks (nodes).

Each node stores its own data and the address of the next node.

```
  +------+    +------+    +------+
  | 10 | *——→ | 20 | *——→ | 30 | NULL
  +------+    +------+    +------+
  Node1       Node2       Node3
```

- The represents the pointer to the next node.
- The last node's pointer is `NULL`, indicating the end of the list.

🔷 7️⃣ Creation of Linked List

We can create a linked list dynamically using `malloc()` in C and `new` in C++.

🟩 Linked List Creation in C

```
#include <stdio.h>
#include <stdlib.h>
```

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

int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;

    // Allocate memory for nodes in heap
    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    // Assign data and link nodes
    head→data = 10;
    head→next = second;

    second→data = 20;
    second→next = third;

    third→data = 30;
    third→next = NULL;

    // Print linked list
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d → ", temp→data);
        temp = temp→next;
    }
    printf("NULL\n");

    // Free memory
    free(head);
```

```
    free(second);
    free(third);

    return 0;
}
```

🟦 Linked List Creation in C++

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int val) {
        data = val;
        next = nullptr;
    }
};

int main() {
    Node* head = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);

    head->next = second;
    second->next = third;
    third->next = nullptr;

    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
```

```
    }
    cout << "NULL" << endl;

    // Free memory manually (good practice)
    delete head;
    delete second;
    delete third;

    return 0;
}
```

🔷 8️⃣ Types of Linked Lists

🟢 1. **Singly Linked List**

Each node points to the next node only.

```
[10|*] → [20|*] → [30|NULL]
```

➡️ Traversal is **only in one direction**.

🟣 2. **Doubly Linked List**

Each node has two pointers:

- One to the **next node**
- One to the **previous node**

```
NULL ← [10|*|*] ↔ [20|*|*] ↔ [30|*|NULL]
```

➡️ Traversal is **bidirectional** (both directions).

🔵 3. **Circular Linked List**

The **last node points back to the first node**.

```
[10|*] → [20|*] → [30|*]
  ↑_____________________↓
```

➡️ No `NULL` — traversal is circular.

🟡 4. **Circular Doubly Linked List**

Each node has both `prev` and `next` pointers,

and the **last node links to the first**.

```
[10] ↔ [20] ↔ [30]
  ↑_____________↓
```

🔷 9️⃣ Advantages of Linked List over Array

Feature	Array	Linked List
Size	Fixed	Dynamic
Memory usage	Continuous	Scattered
Insertion/Deletion	Costly (O(n))	Easy (O(1))
Access	Random Access	Sequential Access
Memory waste	Possible	None

🔷 🔟 Disadvantages of Linked List

1. Extra memory for pointer fields.
2. No random access (must traverse nodes sequentially).
3. Complex to implement and debug.
4. Reverse traversal not possible in singly linked list.
5. Memory overhead due to dynamic allocation.

🧾 Summary

Concept	Description
Linked List	Dynamic, non-contiguous data structure of connected nodes
Node Structure	Contains data + pointer
Types	Singly, Doubly, Circular, Circular Doubly
Advantages	Dynamic size, fast insertion/deletion
Drawbacks	Extra pointer space, sequential access only

🚀 Conclusion

A **Linked List** is one of the most important **dynamic data structures** in C and C++.

It provides **flexibility** in memory usage and forms the **foundation** for advanced structures like stacks, queues, graphs, and trees.
