📘 Linked List in C++

🔰 1. What is a Linked List?

A **Linked List** is a **linear data structure** used to store a collection of elements (called **nodes**), where each node is connected to the next using a pointer.

✅ Each node contains:

- **Data**: The actual value to store
- **Next Pointer**: Address of the next node in the list

🧠 Key Idea:

Linked lists are **not stored in a continuous block of memory** like arrays. Instead, each node can be anywhere in memory.

📦 2. Real-Life Analogy

Imagine a chain of people, each holding a chit with some data and the phone number of the next person.
You just need to know the first person (called the **head**), and then you can follow the chain.

📐 3. Structure of a Node (C++ Code)

```
struct Node {
    int data;         // Data part
    Node* next;       // Pointer to next node
};
```

🔄 4. Types of Linked Lists

Type	Description	Structure Example

Singly Linked List	One pointer: points to next node only	`10 → 20 → 30 → NULL`
Doubly Linked List	Two pointers: next and previous node	`NULL ← 10 ⇄ 20 ⇄ 30 → NULL`
Circular Linked List	Last node points back to head	`10 → 20 → 30 ↺ (back to 10)`

---

5. Basic Terminologies

- **Node**: Element of the linked list
- **Head**: First node
- **Tail**: Last node
- **NULL**: Marks the end of the list

---

6. Why Use Linked List?

Feature	Array	Linked List
Size	Fixed	Dynamic (grow/shrink)
Memory	Contiguous	Non-contiguous
Insertion	Costly (shifting)	Easy (just pointers)
Deletion	Costly (shifting)	Easy
Access	Fast (random)	Slow (sequential)

---

7. Operations on Singly Linked List (C++ Code)

We will now study how to perform:

- Insertions
- Deletions

- Traversal (Display)
- Search

✏️ A. Insertion

➕ i. Insert at Beginning

```
void insertAtBeginning(int value) {
    Node* newNode = new Node();  // Create new node
    newNode->data = value;       // Assign data
    newNode->next = head;        // Point to current head
    head = newNode;              // Update head
}
```

📊 Dry Run:

```
head = NULL;

insertAtBeginning(10);
=> head → [10|NULL]

insertAtBeginning(20);
=> head → [20|] → [10|NULL]
```

➕ ii. Insert at End

```
void insertAtEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        return;
    }

    Node* temp = head;
    while (temp->next != NULL)
```

```
        temp = temp->next;

    temp->next = newNode;
}
```

➕ iii. Insert at Any Position

```
void insertAtPosition(int value, int pos) {
    Node* newNode = new Node();
    newNode->data = value;

    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }

    Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;

    if (temp == NULL) return;

    newNode->next = temp->next;
    temp->next = newNode;
}
```

❌ B. Deletion

➖ i. Delete from Beginning

```
void deleteFromBeginning() {
    if (head == NULL) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
```

➖ ii. Delete from End

```
void deleteFromEnd() {
    if (head == NULL) return;

    if (head->next == NULL) {
        delete head;
        head = NULL;
        return;
    }

    Node* temp = head;
    while (temp->next->next != NULL)
        temp = temp->next;

    delete temp->next;
    temp->next = NULL;
}
```

➖ iii. Delete from Any Position

```
void deleteFromPosition(int pos) {
    if (head == NULL) return;

    if (pos == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* temp = head;
    for (int i = 1; i < pos - 1 && temp->next != NULL; i++)
        temp = temp->next;

    if (temp->next == NULL) return;
    Node* toDelete = temp->next;
    temp->next = toDelete->next;
    delete toDelete;
}
```

👁️ C. Display (Traversal)

```
void display() {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}
```

🔍 D. Search

```
void search(int key) {
    Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            cout << "Element found at position " << pos << endl;
            return;
        }
        temp = temp->next;
        pos++;
    }
    cout << "Element not found\n";
}
```

🧠 Memory Diagram: Step-by-Step Insertion at Beginning

```
insertAtBeginning(10)
  New Node: [10|NULL] → head

insertAtBeginning(20)
  New Node: [20|] → [10|NULL] → head

insertAtBeginning(30)
  New Node: [30|] → [20|] → [10|NULL] → head
```

🧪 Complete Menu-Driven Program

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

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

Node* head = NULL;

void insertAtBeginning(int value);
void insertAtEnd(int value);
void deleteFromBeginning();
void deleteFromEnd();
void display();
void search(int value);

int main() {
    int choice, value;

    while (true) {
        cout << "\n--- MENU ---\n";
        cout << "1. Insert at Beginning\n2. Insert at End\n3. Delete from Beginning\n4. Delete from End\n";
        cout << "5. Display\n6. Search\n7. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value: ";
                cin >> value;
                insertAtBeginning(value);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> value;
                insertAtEnd(value);
                break;
            case 3:
                deleteFromBeginning();
                break;
            case 4:
                deleteFromEnd();
                break;
            case 5:
                display();
                break;
            case 6:
                cout << "Enter value to search: ";
                cin >> value;
                search(value);
                break;
            case 7:
                return 0;
            default:
                cout << "Invalid choice!";
        }
    }
    return 0;
}
```

🎓 Quiz for Students

1. What is a linked list?

2. Define the structure of a node.
   #

3. What is the purpose of the `next` pointer?
4. Write code to insert a node at the beginning.
5. What will the linked list look like after inserting: 30, 20, 10 (at beginning)?
6. Why is linked list better than array for insertion and deletion?
7. What is the output of `display()` if the list is empty?
8. What does `head = head->next` do in deletion?

✅ Summary

- Linked List is a collection of nodes, each pointing to the next.
- It is dynamic and memory efficient for insert/delete.
- Access is sequential, not random.
- Mastering insertion, deletion, and traversal is key.

📋 Linked List Quiz

✅ Section A: Conceptual Questions

1. What is a linked list? How is it different from an array?
2. What is a node in a linked list? What are its components?
3. Define:
    - Head
    - NULL
    - Pointer
4. What is the difference between:
    - Singly Linked List
    - Doubly Linked List
    - Circular Linked List
5. What is the time complexity of:
    - Insertion at beginning?
    - Insertion at end?
    - Deletion from beginning?
6. Explain how memory is allocated for a linked list.
7. Why is insertion and deletion easier in linked list compared to arrays?
8. Can we access the last node directly in a singly linked list? Why or why not?
9. What does the `next` pointer in the last node of a singly linked list point to?
10. What happens if you delete the head node but forget to update the `head` pointer?

✅ Section B: Code Understanding

11. Identify the error in this code:

```
Node* newNode = new Node;
newNode->data = 10;
newNode->next = NULL;
head->next = newNode;
```

12. Write the structure definition of a singly linked list node in C++.

13. What is the output of the following code?

```
insertAtBeginning(10);
insertAtBeginning(20);
display();
```

Output: ________

14. What will happen if we try to delete a node from an empty linked list?

15. In the function `insertAtEnd()`, why do we use a `while` loop?

✅ Section C: Dry Run (Trace the Output)

16. Given this series of operations, draw the linked list:

```
insertAtBeginning(30);

insertAtBeginning(20);

insertAtBeginning(10);
```

Final List: ________

17. Suppose `head` points to:

```
head → [10|next] → [20|next] → [30|NULL]
```

After deleting from the beginning, what will the list look like?

18. What is the result of the following search operation?

```
search(50); // if list has 10 → 20 → 30
```

✅ Section D: Fill in the Blanks

19. In a linked list, each node contains two parts: ________ and ________.

20. In a singly linked list, we can only move in ______ direction.

21. In `deleteFromEnd()`, we stop at the ________ node.

22. The last node of a circular linked list points to the ________.

23. The purpose of the `next` pointer is to store ________.

✅ Section E: Practical Coding Tasks (Short)

24. Write a function to count the number of nodes in a linked list.

25. Write a function to insert a node at a given position.

26. Write code to delete a node with a given key value.

27. Write a function to search a node and return its position (1-based index).

28. Modify the display function to also print total number of nodes.

✅ Section F: Output Prediction

29. Predict the output:

```
insertAtEnd(5);
insertAtBeginning(10);
insertAtEnd(15);
display();
```

Output: ________

30. If the list is: `10 → 20 → 30 → NULL`
    What will be the list after calling `deleteFromEnd()`?

Answer: _______