🔗 Linked List in C++

1. Introduction – Why Linked List instead of Arrays?

👉 Arrays have fixed size, insertion/deletion is costly (shifting required).
👉 Linked List is **dynamic** in size, and insertion/deletion is easier.

2. Node Structure

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

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

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

3. Creating & Traversing a Linked List

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

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

int main() {
    // Create nodes
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);

    cout << "Linked List: ";
    printList(head);
    return 0;
}
```

(a) Insert at Beginning

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

void insertAtBeginning(Node*& head, int val) {
    Node* newNode = new Node(val);
    newNode->next = head;
    head = newNode;
}

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

int main() {
    Node* head = NULL;

    insertAtBeginning(head, 30);
    insertAtBeginning(head, 20);
    insertAtBeginning(head, 10);

    cout << "Linked List after Insertion at Beginning: ";
    printList(head);
    return 0;
}
```

(b) Insert at End

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

void insertAtEnd(Node*& head, int val) {
    Node* newNode = new Node(val);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

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

int main() {
    Node* head = NULL;

    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);

    cout << "Linked List after Insertion at End: ";
    printList(head);
    return 0;
}
```

(c) Insert at Given Position

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

void insertAtPosition(Node*& head, int val, int pos) {
    Node* newNode = new Node(val);

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

    if (temp == NULL) {
        cout << "Position out of range!\n";
        return;
    }

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

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

int main() {
    Node* head = NULL;

    insertAtPosition(head, 10, 1);
    insertAtPosition(head, 20, 2);
    insertAtPosition(head, 30, 2);

    cout << "Linked List after Insertion at Position: ";
    printList(head);
    return 0;
}
```

(a) Delete from Beginning

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

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

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

int main() {
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);

    deleteFromBeginning(head);

    cout << "Linked List after Deletion from Beginning: ";
    printList(head);
    return 0;
}
```

(b) Delete from End

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

void deleteFromEnd(Node*& head) {
    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;
}

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

int main() {
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);

    deleteFromEnd(head);

    cout << "Linked List after Deletion from End: ";
    printList(head);
    return 0;
}
```

(c) Delete from Given Position

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

```
class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

void deleteFromPosition(Node*& head, 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; temp != NULL && i < pos - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        cout << "Position out of range!\n";
        return;
    }

    Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    delete nodeToDelete;
}

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

```
int main() {
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);

    deleteFromPosition(head, 2);

    cout << "Linked List after Deletion from Position: ";
    printList(head);
    return 0;
}
```

1️⃣ Singly Linked List (SLL)

👉 Each node has **data + next pointer**.

👉 Traversal goes only **forward**.

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

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

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

int main() {
    Node* head = new Node(10);
    head->next = new Node(20);
    head->next->next = new Node(30);
    cout << "Singly Linked List: ";
    printList(head);

    return 0;
}
```

2️⃣ Doubly Linked List (DLL)

👉 Each node has **data + prev pointer + next pointer**.

👉 Can traverse **both directions**.

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int val) {
        data = val;
        prev = NULL;
        next = NULL;
    }
};

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

void printBackward(Node* tail) {
    Node* temp = tail;
    cout << "Backward: ";
    while (temp != NULL) {
        cout << temp->data << " <-> ";
        temp = temp->prev;
    }
    cout << "NULL\n";
}

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

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

    cout << "Doubly Linked List:\n";
    printForward(head);
    printBackward(third);

    return 0;
}
```