Singly Linked LIst Basic operations

Insert Node

1. **At Head**
   Insert a new node at the beginning of the list.

2. **At Tail**
   Insert a new node at the end of the list.

3. **At Nth Position**
   Insert a new node at the N-th position (1-based indexing, before the current N-th node).

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

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

    Node(int data) {
        this->data = data;
        this->Next = NULL;
    }
};

void InsertAtHead(Node* &head, int data) {
    Node* newNode = new Node(data);
    newNode->Next = head;
    head = newNode;
}

void InsertAtTail(Node* &head, int data) {
    Node* newNode = new Node(data);
    Node* temp = head;
    while (temp->Next != NULL) {
        temp = temp->Next;
```

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

void InsertAtNthPosition(Node* &head, int data, int position) {
    if (position == 1) {
        InsertAtHead(head, data);
        return;
    }

    Node* newNode = new Node(data);
    Node* temp = head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->Next;
    }

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

    newNode->Next = temp->Next;
    temp->Next = newNode;
}

void PrintList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->Next;
    }
    cout << "NULL" << endl;
}

int main() {
    Node* head = new Node(10);

   cout<<" insert at head(start)"<<endl;
    InsertAtHead(head, 30);
    PrintList(head);
```

```
    cout<<" insert at tail(last)"<<endl;
    InsertAtTail(head, 50);
    PrintList(head);

    cout<<" insert at nth position (Given position)"<<endl;
    InsertAtNthPosition(head, 40, 2);
    PrintList(head);


    return 0;
}
```

Output

```
 insert at head(start)
30 -> 10 -> NULL
 insert at tail(last)
30 -> 10 -> 50 -> NULL
 insert at nth position (Given position)
30 -> 40 -> 10 -> 50 -> NULL
```

Delete Node

At Head
Delete the first node of the linked list.

At Tail
Delete the last node of the linked list.

At Nth Position
Delete the node at the N-th position (1-based indexing).

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

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

    Node(int data) {
        this->data = data;
        this->Next = NULL;
    }
};

// Insert at head
void InsertAtHead(Node* &head, int data) {
    Node* newNode = new Node(data);
    newNode->Next = head;
    head = newNode;
}
```

```
// Insert at tail
void InsertAtTail(Node* &head, int data) {
    Node* newNode = new Node(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->Next != NULL) {
        temp = temp->Next;
    }
    temp->Next = newNode;
}

// Insert at nth position
void InsertAtNthPosition(Node* &head, int data, int position) {
    if (position == 1) {
        InsertAtHead(head, data);
        return;
    }

    Node* newNode = new Node(data);
    Node* temp = head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->Next;
    }

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

    newNode->Next = temp->Next;
    temp->Next = newNode;
}

// Delete at head
void DeleteAtHead(Node* &head) {
    if (head == NULL) {
        cout << "List is empty!" << endl;
```

```
        return;
    }
    Node* temp = head;
    head = head->Next;
    delete temp;
}

// Delete at tail
void DeleteAtTail(Node* &head) {
    if (head == NULL) {
        cout << "List is empty!" << endl;
        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;
}

// Delete at nth position
void DeleteAtNthPosition(Node* &head, int position) {
    if (head == NULL) {
        cout << "List is empty!" << endl;
        return;
    }

    if (position == 1) {
        DeleteAtHead(head);
        return;
    }
```

```
    Node* temp = head;
    for (int i = 1; i < position - 1 && temp->Next != NULL; i++) {
        temp = temp->Next;
    }

    if (temp->Next == NULL) {
        cout << "Position out of range!" << endl;
        return;
    }

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

// Print the linked list
void PrintList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->Next;
    }
    cout << "NULL" << endl;
}

int main() {
    Node* head = NULL; // Initially list is empty

    InsertAtHead(head, 10);
    InsertAtHead(head, 20);
    InsertAtTail(head, 30);
    InsertAtNthPosition(head, 25, 3);

    cout << "After Insertions:" << endl;
    PrintList(head);  // 20 -> 10 -> 25 -> 30 -> NULL

    DeleteAtHead(head);
    cout << "After Deleting Head:" << endl;
    PrintList(head);  // 10 -> 25 -> 30 -> NULL
```

```
    DeleteAtTail(head);
    cout << "After Deleting Tail:" << endl;
    PrintList(head);  // 10 -> 25 -> NULL

    DeleteAtNthPosition(head, 2);
    cout << "After Deleting at 2nd Position:" << endl;
    PrintList(head);  // 10 -> NULL

    DeleteAtHead(head);
    cout << "After Deleting Last Node:" << endl;
    PrintList(head);  // NULL

    return 0;
}
```

Output

```
After Insertions:
20 -> 10 -> 25 -> 30 -> NULL
After Deleting Head:
10 -> 25 -> 30 -> NULL
After Deleting Tail:
10 -> 25 -> NULL
After Deleting at 2nd Position:
10 -> NULL
After Deleting Last Node:
NULL
```