Chapter Three

Linked List

Lab 1: Linked List Codes

Objective:

- To understand or learn to **declare, initialize**, and **access Linked List** elements in C++ programming with the help of examples.
- To familiarize and understand Application of Linked List Oprations.

Program 1: this program show how to **insert elements in to Linked List in three .**

```
// A complete working C++ program to
// demonstrate all insertion methods on Linked List
#include <bits/stdc++.h>
using namespace std;

// A linked list node
class Node
{
    public:
    int data;
    Node *next;
};

// Given a reference (pointer to pointer)
// to the head of a list and an int, inserts
// a new node on the front of the list.
void InsertAtFront(Node** head_ref, int new_data)
{

    // 1. allocate node
    Node* new_node = new Node();

    // 2. put in the data
    new_node->data = new_data;

    // 3. Make next of new node as head
    new_node->next = (*head_ref);

    // 4. move the head to point to the new node
    (*head_ref) = new_node;
```

```
}

// Given a node prev_node, insert a new
// node after the given prev_node
void InsertAfterNode(Node* prev_node, int new_data)
{
    // 1. check if the given prev_node is NULL
    if (prev_node !== NULL)
    {
        cout<<"The given previous node cannot be NULL";
        return;
    }

    // 2. allocate new node
    Node* new_node = new Node();

    // 3. put in the data
    new_node.data = new_data;

    // 4. Make next of new node as next of prev_node
    new_node.next = prev_node->next;

    // 5. move the next of prev_node as new_node
    prev_node->next = new_node;
}

// Given a reference (pointer to pointer)
// to the head of a list and an int, appends a new node at the end
void InsertAtLast(Node** head_ref, int new_data)
{
    // 1. allocate node
    Node* new_node = new Node();

    //used in step 5
    Node *last = *head_ref;

    // 2. put in the data
    new_node->data = new_data;

    /* 3. This new node is going to be
    the last node, so make next of
    it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty,
    then make the new node as head */
```

```
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
    {
        last = last->next;
    }

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}

// This function prints contents of linked list starting from head
void printList(Node *node)
{
    while (node != NULL)
    {
        cout<<" "<<node->data;
        node = node->next;
    }
}

// Driver code
int main()
{

    // Start with the empty list
    Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    InsertatLast(&head, 6);

    // Insert 7 at the beginning. So linked list becomes 7->6->NULL
    InsertatFront(&head, 7);

    // Insert 1 at the beginning.
    // So linked list becomes 1->7->6->NULL
    InsertatFront(&head, 1);

    // Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
    InsertAtLast(&head, 4);
```

```
    // Insert 8, after 7. So linked
    // list becomes 1->7->8->6->4->NULL
    InsertafterNode(head->next, 8);

    cout<<"Created Linked list is: ";
    printlist(head);

    return 0;
}
```

Program 2: this program show how to delete **elements from Linked List in three methods.**

```
#include <bits/stdc++.h>
using namespace std;

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

void Push(Node** head, int A)
{
    Node* n = (Node*)malloc(sizeof(Node));
    n->number = A;
    n->next = *head;
    *head = n;
}

void deleteN(Node** head, int position)
{
    Node* temp;
    Node* prev;
    temp = *head;
    prev = *head;
    for (int i = 0; i < position; i++) {
        if (i == 0 && position == 1) {
            *head = (*head)->next;
            free(temp);
        }
        else {
            if (i == position - 1 && temp) {
                prev->next = temp->next;
                free(temp);
```

```
            }
            else {
                prev = temp;

                // Position was greater than
                // number of nodes in the list
                if (prev == NULL)
                    break;
                temp = temp->next;
            }
        }
    }
}

void printList(Node* head)
{
    while (head) {
        if (head->next == NULL)
            cout << "[" << head->number << "] "
                 << "[" << head << "]->"
                 << "(nil)" << endl;
        else
            cout << "[" << head->number << "] "
                 << "[" << head << "]->" << head->next
                 << endl;
        head = head->next;
    }
    cout << endl << endl;
}

// Driver code
int main()
{
    Node* list = (Node*)malloc(sizeof(Node));
    list->next = NULL;
    Push(&list, 1);
    Push(&list, 2);
    Push(&list, 3);

    printList(list);

    // Delete any position from list
    deleteN(&list, 1);
    printList(list);
```

```
    return 0;
}
```

Program 3: this program show how to search **elements from Linked List.**

```
// C++ program to find n'th
// node in linked list
#include <assert.h>
#include <bits/stdc++.h>
using namespace std;

// Link list node
class Node {
public:
    int data;
    Node* next;
};

/* Given a reference (pointer to
pointer) to the head of a list
and an int, push a new node on
the front of the list. */
void push(Node** head_ref, int new_data)
{

    // allocate node
    Node* new_node = new Node();

    // put in the data
    new_node->data = new_data;

    // link the old list
    // of the new node
    new_node->next = (*head_ref);

    // move the head to point
    // to the new node
    (*head_ref) = new_node;
}

// Takes head pointer of
// the linked list and index
// as arguments and return
// data at index
int GetNth(Node* head, int index)
```

```
{

    Node* current = head;

    // the index of the
    // node we're currently
    // looking at
    int count = 0;
    while (current != NULL) {
        if (count == index)
            return (current->data);
        count++;
        current = current->next;
    }

    /* if we get to this line,
    the caller was asking
    for a non-existent element
    so we assert fail */
    assert(0);
}

// Driver Code
int main()
{
    // Start with the
    // empty list
    Node* head = NULL;

    // Use push() to construct
    // below list
    // 1->12->1->4->1
    push(&head, 1);
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
    push(&head, 1);

    // Check the count
    // function
    cout << "Element at index 3 is " << GetNth(head, 3);
    return 0;
}
```