**Lab-07**

**Lab Title: Implementation of Linked List**

**Objectives**

The objectives of this lab are as follows:

1. To implement a singly linked list data structure
2. To understand the fundamental operations on a linked list, including traversing, inserting, and deleting elements.
3. To practice working with node structures and references in a linked list.

**Theory:**

**Linked List**

A linked list is a linear data structure in which elements are stored in nodes, and each node points to the next node in the sequence. It is a dynamic data structure that can grow or shrink in size as needed. Unlike arrays, linked lists do not have a fixed size and do not require contiguous memory allocation.

**Node**

A node is the basic building block of a linked list. Each node contains two parts:

* Data: The actual value or data that the node holds.
* Next: A reference (pointer) to the next node in the sequence. In a singly linked list, nodes only have a reference to the next node, while in a doubly linked list, nodes have references to both the next and previous nodes.

**Task 1:**

1. **Traversing**: Traversing a linked list means moving through the list from the head node to the end, one node at a time, to access or display its elements.
2. **Insertion**: Insertion in a linked list involves adding a new node with a specified value. Common insertion points include the beginning (head), end, or after a specific node.
3. **Deletion**: Deletion in a linked list involves removing a node with a specified value from the list. The node to be deleted is located by traversing the list, and its references are adjusted to remove it from the sequence.

**CODE:**

|  |  |  |
| --- | --- | --- |
| #include <bits/stdc++.h>  using namespace std;  class Node  {  public:  int val;  Node \*next;  Node(int val)  {  this->val = val;  this->next = NULL;  }  };  void insert\_at\_tail(Node \*&head, int v)  {  Node \*newNode = new Node(v);  if (head == NULL)  {  head = newNode;  cout << endl  << "Inserted at head" << endl  << endl;  return;  }  Node \*tmp = head;  while (tmp->next != NULL)  {  tmp = tmp->next;  }  // tmp ekhon last node e  tmp->next = newNode;  cout << endl  << "Inserted at tail" << endl  << endl;  }  void print\_linked\_list(Node \*head)  {  cout << endl;  cout << "Your Linked List: ";  Node \*tmp = head;  while (tmp != NULL)  {  cout << tmp->val << " ";  tmp = tmp->next;  }  cout << "Option 1: Insert at Tail" << endl;  cout << "Option 2: Print Linked List" << endl;  cout << "Option 3: Insert at any Position" << endl;  cout << "Option 4: Insert at Head" << endl;  cout << "Option 5: Delete from Position" << endl;  cout << "Option 6: Terminate" << endl;  int op;  cin >> op;  if (op == 1)  {  cout << "Please enter value: ";  int v;  cin >> v;  insert\_at\_tail(head, v);  }  else if (op == 2)  {  print\_linked\_list(head);  }  else if (op == 3)  {  int pos, v;  cout << "Enter position: ";  cin >> pos;  cout << "Enter value: ";  cin >> v; | cout << endl  << endl;  }  void insert\_at\_position(Node \*head, int pos, int v)  {  Node \*newNode = new Node(v);  Node \*tmp = head;  for (int i = 1; i <= pos - 1; i++)  {  tmp = tmp->next;  }  newNode->next = tmp->next;  tmp->next = newNode;  cout << endl  << endl  << "Inserted at position " << pos << endl  << endl;  }  void insert\_at\_head(Node \*&head, int v)  {  Node \*newNode = new Node(v);  newNode->next = head;  head = newNode;  cout << endl  << "Inserted at head" << endl  << endl;  }  void delete\_from\_position(Node \*head, int pos)  {  Node \*tmp = head;  for (int i = 1; i <= pos - 1; i++)  {  tmp = tmp->next;  }  Node \*deleteNode = tmp->next;  tmp->next = tmp->next->next;  delete deleteNode;  }  int main()  {  Node \*head = NULL;  while (true)  {    if (pos == 0)  {  insert\_at\_head(head, v);  }  else  {  insert\_at\_position(head, pos, v);  }  }  else if (op == 4)  {  int v;  cout << "Enter value: ";  cin >> v;  insert\_at\_head(head, v);  }  else if (op == 5)  {  int pos;  cout << "Enter position: ";  cin >> pos;  delete\_from\_position(head, pos);  }  }  return 0;  } |  |

**OUTPUT :**

