**Experiment No. 4**

**Title :** Implementation of Linked list Linear Data Structure

**Problem Statement :** Implementing linear data structure linked list using class and structure with functions

addNode()

delNode()

Display()

InsertNode()

searchNode()

**Algorithm :**

**S1 :** Start

**S2 :** Declare a structure Node with data and a pointer pointing to itself. Create a class with the member functions and head and tail pointers. In constructor of class initialize head and tail values as NULL.

**S3 :** Call the functions in a switch statement according to the choice value

**S4 :** In addNode function declare a new node and ask for the value to be added as node. If there is no value in linked list point head and tail to the created node else point the tail to temp node and in tail->next pointed to NULL.

**S5 :** In delNode function declare two new node pointers prev and curr and keep incrementing using curr pointer and if value at curr node is equal to one to be deleted make the prev->next equal to address of node one next to curr. By this we are skipping the node to be deleted.

**S6 :** In search function declare a pointer and keep incrementing until value at node matches and return the position

**S7 :** In insert function take the value to be inserted in a temp node and the in temp->next give head address. By this we have inserted the node at beginning.

**S8 :** In display function declare a temp node with head as initial value and keep incrementing until temp becomes NULL and print value at node.

**S9 :** Stop

**Code :**

#include<iostream>

using namespace std;

struct Node{ //Declare the structure Node

int data;

Node \*next; //pointer pointing to Node structure

};

class linkedlist{ //Declare class linked list

Node \*head,\*tail;

public :

linkedlist() //Constructor to make head and tail to NULL initially

{

head = NULL;

tail = NULL;

}

void addnode(int n) // Declare add node function

{

Node \*temp = new Node; //Create a new node by struct Node

temp->data = n;

temp->next = NULL;

if(head == NULL)

{

head = temp;

tail = temp;

}

else{

tail->next = temp;

tail = tail->next;

}

}

void delnode() //Delete Node

{

int n;

Node \*prev = new Node; //Previous node tracer

Node \*curr = new Node; //Current node under value comparision

prev = NULL;

curr = head;

cout<<"Enter the value to be deleted : ";

cin>>n;

while(curr != NULL)

{

if(curr->data == n)

{

prev->next = curr->next; //Skip the address of node to be deleted by giving address of next node

prev = prev->next;

return;

}

prev = curr;

curr = curr->next;

}

}

void searchnode() //Search a node in linked list by value

{

int n,c=0;

Node \*temp = new Node;

temp = head;

cout<<"Enter the value to be searched :";

cin>>n;

while(temp != NULL)

{

if(temp->data == n)

{

cout<<"The element found at position "<<c<<endl;

}

temp = temp->next; //Traversal in linked list

c = c + 1;

}

}

void insertnode() //Insert node inserts node at the beginning of linked list

{

Node \*temp = new Node;

int n;

cout<<"Enter the value :";

cin>>n;

temp->data = n;

temp->next = NULL;

temp->next = head;

head = temp;

}

void display() //Display function

{

Node \*temp = new Node;

temp = head;

while(temp!=NULL)

{

cout<<temp->data<<"\t";

temp = temp->next;

}

cout<<endl;

}

};

int main()

{

linkedlist L; //declare an object of class linked list

int ch,n;

start :

cout<<"Enter the choice\n1.Add\t2.Delete\t3.Search\t4.Insert\t5.Display\t";

cin>>ch; //Ask for choice

switch(ch)

{

case 1 :

cout<<"Enter the value : ";

cin>>n;

L.addnode(n);

break;

case 2 :

L.delnode();

break;

case 3 :

L.searchnode();

break;

case 4 :

L.insertnode();

break;

case 5 :

L.display();

break;

default :

cout<<"Enter correct choice\n";

}

goto start;

}

**Output :**

****

****

**Analysis :**

* In insert function the value is added at the beginning of linked list always instead of having a flexibility for that.
* Since the program requires many nodes and address for each will be held by previous node any mislead in address may lead linked list to fail.