**Experiment No.5**

**Title:** Implementation of Linked List

**Problem Statement:**

Write a menu driven C++ program to create

linked list and perform the following functions.

(i) create

(ii) insert

(iii) delete

(iv)search

(v)display

**Algorithm:**

**Step 1**

Declare a class which defines a single node.

**Step 2**

A class which will contain the functions to handle the nodes and

two important pointers i.e. head and tail is created.

**Step 3**

Class which defines the required operations on liked list are defined (viz. create, insert, delete, search, display ) .

**Step 4**

Scan the required operation on linked list and it is performed accordingly by calling the respective functions.

**Code:**

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node\* next;

};

class LinkedList:public Node

{

Node \*head,\*tail;

public:

LinkedList()

//Constructor

{

head=NULL;

tail=NULL;

}

void createNode();

void insert();

void delet();

void display();

void search();

};

int main()

{

LinkedList lst;

int ch;

while(1)

{

cout<<"\n\*\*\*\* MENU\*\*\*\*";

cout<<"\n1:CREATE\t2:INSERT\t3:DELETE\t4:SEARCH\t5:DISPLAY\t6:EXIT\n";

cout<<"\nEnter Your Choice:";

cin>>ch;

switch(ch)

{

case 1:

lst.createNode();

break;

case 2:

lst.insert();

break;

case 3:

lst.delet();

break;

case 4:

lst.search();

break;

case 5:

lst.display();

break;

case 6:

return 0;

}

}

return 0;

}

void LinkedList::createNode() //Creates anode at the tail of thelist

{

Node \*temp=new Node;

int n;

cout<<"\n Enter an Element:";

cin>>n;

temp->data=n;

temp->next=NULL;

if(head==NULL)

{

head=temp;

tail=head;

}

else

{

tail->next=temp;

tail=temp;

}

}

void LinkedList :: insert() // Inserts a node at

//the beginning of the list

{

int n;

Node \*temp=new Node;

cout<<"\nEnter an Element:";

cin>>n;

temp->data=n;

temp->next=NULL;

temp->next=head;

head=temp;

}

void LinkedList::delet()

// Deletes a node at the given position

{

Node \*prev=NULL,\*cur=head;

int count=1,pos,ch;

if(cur!=NULL) //if 1

{

cout<<"\nEnter the Position of Deletion:";

cin>>pos;

while(count!=pos)

{

prev=cur;

cur=cur->next;

count++;

}

if(count==pos) //if2

{

cout<<"\nDeleted Element is: " <<cur->data;

prev->next=cur->

next;

}// end if2

}// end if1

else

cout<<"\nNot Able to Delete";

}

void LinkedList::display() // Displays the whole List

{

Node \*temp=head;

if(temp==NULL)

{

cout<<"\nList is Empty";

}

while(temp!=NULL)

{

cout<<temp->data;

cout<<"-->";

temp=temp->next;

}

cout<<"NULL";

}

void LinkedList::search()

{

int value,pos=0;

bool flag=false;

if(head==NULL)

{

cout<<"List is Empty";

return;

}

cout<<"Enter the Value to be Searched:";

cin>>value;

Node \*temp;

temp=head;

while(temp!=NULL)

{

pos++;

if(temp->data==value)

{

flag=true;

cout<<"Element "<<value<< " is Found at " <<pos<<" Position";

return;

}

temp=temp->next;

}

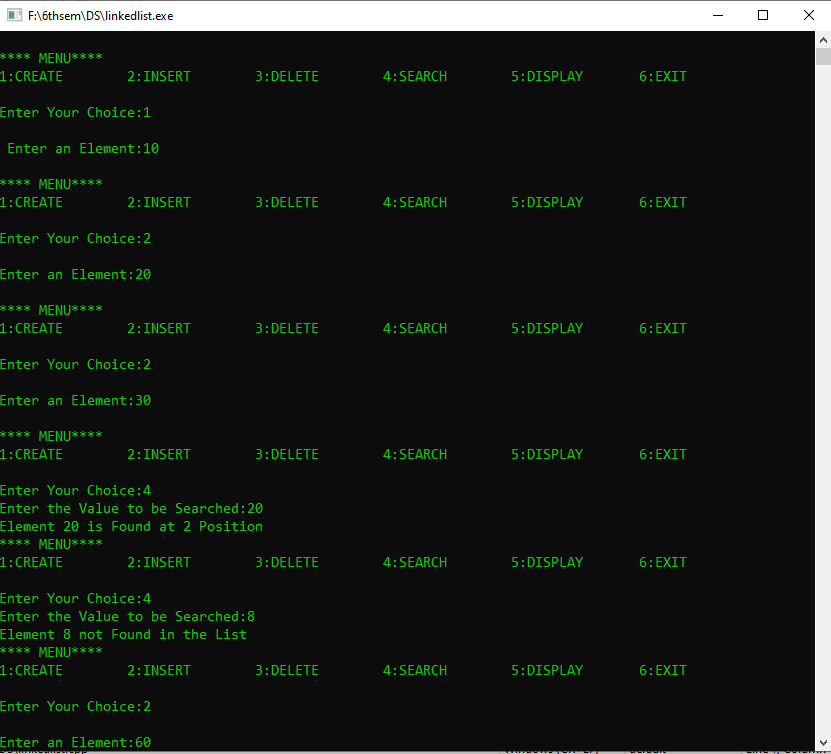
if(!flag)

{

cout<<"Element "<<value<<" not Found in the List";

}

}

**Results:**

**Analysis(Limitations):**

Linked list is a dynamic data structure so it can grow and shrink at runtime by allocating and deallocating memory. But we some limitations:

* More memory is required to store elements in linked list as compared to array. Because in linked list each node contains a pointer and it requires extra memory for itself.
* Elements or nodes traversal is difficult in linked list. We can not randomly access any element as we do in array by index.
* In linked list reverse traversing is really difficult. In case of [doubly linked list](https://www.thecrazyprogrammer.com/2015/09/doubly-linked-list-in-c-and-cpp.html) its easier but extra memory is required for back pointer hence wastage of memory.