**IMPLEMENTATION OF LIST USING LINKED LISTS**

**LAB # 07**



**Data Structures & Algorithms**

Submitted by: **Shah Raza**

Registration No: **18PWCSE1658**

Class Section: **B**

“On my honor, as a student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to: **Dr. Khurram Shehzad Khattak**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Lab Objectives:**

Objectives of this lab are:

* To learn about the data structure, lists and their useful operations.
* Singly Linked List
* Doubly Linked List
* Circular Linked List

**Task # 1 and 2:**

Implement a singly linked list.

**Code:**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*next\_node;

};

class LinkedList

{

public:

Node \*head;

LinkedList()

{

head=NULL;

}

void InsertAtStart(int data)

{

Node \*temp,\*p;

temp= new Node;

temp->data=data;

if(head==NULL)

{

head=temp;

head->next\_node=NULL;

}

else

{

p=head;

head=temp;

head->next\_node=p;

}

cout<<"Element inserted at the start.\n";

}

void InsertAtEnd(int data)

{

Node \*temp,\*p;

temp= new Node;

temp->data=data;

p=head;

while(p->next\_node!=NULL)

{

p=p->next\_node;

}

temp->next\_node=NULL;

p->next\_node=temp;

cout<<"Element inserted at the last.\n";

}

void Insert(int data, int pos)

{

int lastpos=0;

Node \*temp,\*p,\*s;

temp= new Node;

temp->data=data;

p=head;

while(p!=NULL)

{

p=p->next\_node;

lastpos++;

}

if(pos==1)

{

InsertAtStart(data);

}

else if(pos>1 && pos<=lastpos)

{

p=head;

for(int i=1;i<pos;i++)

{

s=p;

p=p->next\_node;

}

s->next\_node=temp;

temp->next\_node=p;

}

else

cout<<"Position out of range.\n";

}

void DeleteFromStart()

{

if(head==NULL)

cout<<"List is empty.\n";

else

{

Node \*s;

s=head;

head=head->next\_node;

delete s;

cout<<"Element Deleted.\n";

}

}

void DeleteFromEnd()

{

int counter=0;

Node \*s;

s=head;

while(s!=NULL)

{

s=s->next\_node;

counter++;

}

if(counter==1)

{

head=NULL;

}

else

{

s=head;

while(s->next\_node->next\_node!=NULL)

{

s=s->next\_node;

}

delete (s->next\_node);

s->next\_node=NULL;

}

cout<<"Element Deleted.\n";

}

void DeleteAtPosition(int pos)

{

int counter=0;

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

if(pos==1)

{

DeleteFromStart();

return;

}

Node \*s,\*p;

s=head;

while(s!=NULL)

{

s=s->next\_node;

counter++;

}

if(pos>1 && pos<=counter)

{

s=head;

for(int i=1;i<pos;i++)

{

p=s;

s=s->next\_node;

}

p->next\_node=s->next\_node;

delete s;

cout<<"Element Deleted.\n";

}

else

cout<<"Position out of Range.\n";

}

void Search(int data)

{

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

int pos=0,counter=0;

Node \*s;

s=head;

while(s!=NULL)

{

pos++;

if(s->data==data)

{

cout<<"Element "<<data<<" fount at position"<<pos<<endl;

counter++;

}

s=s->next\_node;

}

if(counter==0)

cout<<"Element not found.\n";

}

void UpdateList(int data, int pos)

{

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

Node \*s;

s=head;

if(pos==1)

{

s->data=data;

}

else

{

for(int i=1;i<pos;i++)

{

if(s==NULL)

{

cout<<"There are less than "<<pos<<" elements\n";

return;

}

s=s->next\_node;

}

s->data=data;

}

cout<<"Node Updated"<<endl;

}

void EmptyList()

{

if(head==NULL)

{

cout<<"List is already Empty.\n";

return;

}

Node \*current,\*ptr;

current=head;

while(current->next\_node!=NULL)

{

ptr=current;

current=current->next\_node;

free(ptr);

}

free(current);

head==NULL;

cout<<"List deleted.\n";

}

void DisplaySize()

{

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

int counter=0;

Node \*s;

s=head;

while(s!=NULL)

{

counter++;

s=s->next\_node;

}

cout<<"Size of List = "<<counter<<endl;

}

void Print()

{

if(head==NULL)

{

cout<<"List is empty.\n";

return;

}

Node \*s;

s=head;

cout<<"Elements of the list are: ";

while(s!=NULL)

{

cout<<s->data<<"-->";

s=s->next\_node;

}

cout<<"NULL\n";

}

};

int main()

{

int choice;

LinkedList L1;

do

{ cout<<endl<<"---------------------------------"<<endl;

cout<<"1.Insert Node at beginning"<<endl;

cout<<"2.Insert node at last"<<endl;

cout<<"3.Insert node at position"<<endl;

cout<<"4.Delete a Particular Node"<<endl;

cout<<"5.Empty the List"<<endl;

cout<<"6.Update Node Value"<<endl;

cout<<"7.Search Element"<<endl;

cout<<"8.Display Linked List"<<endl;

cout<<"9.Display List Size"<<endl;

cout<<"10.Delete from Beginning"<<endl;

cout<<"11.Delete from Last"<<endl;

cout<<"0.Exit "<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Inserting Node at Beginning: "<<endl;

int value;

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

cin>>value;

L1.InsertAtStart(value);

cout<<endl;

break;

case 2:

cout<<"Inserting Node at Last: "<<endl;

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

cin>>value;

L1.InsertAtEnd(value);

cout<<endl;

break;

case 3:

cout<<"Inserting Node at a given position:"<<endl;

int pos;

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

cin>>value;

cout<<"Enter the position at which node is to be inserted: ";

cin>>pos;

L1.Insert(value,pos);

cout<<endl;

break;

case 4:

cout<<"Delete a particular node: "<<endl;

cout<<"Enter the position at which node is to be Deleted: ";

cin>>pos;

L1.DeleteAtPosition(pos);

break;

case 5:

cout<<"Emptying List"<<endl;

L1.EmptyList();

break;

case 6:

cout<<"Update Node Value:"<<endl;

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

cin>>value;

cout<<"Enter the position at which node is to be Updated: ";

cin>>pos;

L1.UpdateList(value,pos);

cout<<endl;

break;

case 7:

cout<<"Search element in Link List: "<<endl;

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

cin>>value;

L1.Search(value);

cout<<endl;

break;

case 8:

cout<<"Display elements of link list"<<endl;

L1.Print();

cout<<endl;

break;

case 9:

cout<<"Size of list: "<<endl;

L1.DisplaySize();

cout<<endl;

break;

case 10:

cout<<"Deleting Element from beginning.\n";

L1.DeleteFromStart();

break;

case 11:

cout<<"Deleting Element from Last.\n";

L1.DeleteFromEnd();

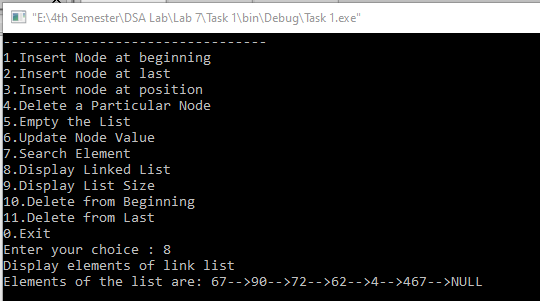
break; }

}while (choice !=0);

return 0;

}

**Output:**

****

**Task # 3:**

Implement a doubly linked list.

**Code:**

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int info;

node \*next;

node \*prev;

}\*start;

class list

{ public:

void create\_list(int value);

void add\_begin(int value);

void add\_at\_a\_position(int value, int position);

void delete\_element(int value);

void search\_element(int value);

void display\_dlist();

void displaysize();

list()

{

start = NULL;

}

};

int main()

{

int choice, element, position;

list dl;

do

{ cout<<endl<<"----------------------------"<<endl;

cout<<"1.Create Node"<<endl;

cout<<"2.Add at beginning"<<endl;

cout<<"3.Add at a given position"<<endl;

cout<<"4.Delete"<<endl;

cout<<"5.Display"<<endl;

cout<<"6.Display size of List"<<endl;

cout<<"0.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch ( choice )

{

case 1:

cout<<"Enter the element: ";

cin>>element;

dl.create\_list(element);

cout<<endl;

break;

case 2:

cout<<"Enter the element: ";

cin>>element;

dl.add\_begin(element);

cout<<endl;

break;

case 3:

cout<<"Enter the element: ";

cin>>element;

cout<<"Insert Element after postion: ";

cin>>position;

dl.add\_at\_a\_position(element, position);

cout<<endl;

break;

case 4:

if (start == NULL)

{

cout<<"List empty,nothing to delete"<<endl;

break;

}

cout<<"Enter the element for deletion: ";

cin>>element;

dl.delete\_element(element);

cout<<endl;

break;

case 5:

dl.display\_dlist();

cout<<endl;

break;

case 6:

dl.displaysize();

break;

}

}

while(choice != 0);

return 0;

}

void list::create\_list(int value)

{

struct node \*s, \*temp;

temp = new(struct node);

temp->info = value;

temp->next = NULL;

if (start == NULL)

{ temp->prev = NULL;

start = temp;

}

else

{ s = start;

while (s->next != NULL)

{ s = s->next; }

s->next = temp;

temp->prev = s;

} }

void list::add\_begin(int value)

{ if (start == NULL)

{ cout<<"List does not exist"<<endl; }

struct node \*temp;

temp = new(struct node);

temp->prev = NULL;

temp->info = value;

temp->next = start;

start->prev = temp;

start = temp;

cout<<"Element Inserted"<<endl;

}

void list::add\_at\_a\_position(int value, int pos)

{ if (start == NULL)

{ cout<<"List does not exist"<<endl; }

struct node \*tmp, \*q;

int i;

q = start;

for (i = 0;i < pos - 1;i++)

{

q = q->next;

if (q == NULL)

{ cout<<"Position out of range"<<endl;

}

}

tmp = new(struct node);

tmp->info = value;

if (q->next == NULL)

{ q->next = tmp;

tmp->next = NULL;

tmp->prev = q;

}

else

{ tmp->next = q->next;

tmp->next->prev = tmp;

q->next = tmp;

tmp->prev = q;

}

cout<<"Element Inserted"<<endl;

}

void list::delete\_element(int value)

{ struct node \*tmp, \*q;

if (start->info == value)

{ tmp = start;

start = start->next;

start->prev = NULL;

cout<<"Element Deleted"<<endl;

free(tmp);

return;

}

q = start;

while (q->next->next != NULL)

{ if (q->next->info == value)

{ tmp = q->next;

q->next = tmp->next;

tmp->next->prev = q;

cout<<"Element Deleted"<<endl;

free(tmp);

return;

}

q = q->next; }

if (q->next->info == value)

{ tmp = q->next;

free(tmp);

q->next = NULL;

cout<<"Element Deleted"<<endl;

return;

}

cout<<"Element "<<value<<" not found"<<endl;

}

void list::display\_dlist()

{ struct node \*q;

if (start == NULL)

{ cout<<"List empty,nothing to display"<<endl;

return;

}

q = start;

cout<<"The Doubly Link List is :"<<endl;

while (q != NULL)

{ cout<<q->info<<" <-> ";

q = q->next;

}

cout<<"NULL"<<endl;

}

void list::displaysize()

{

struct node \*q = start;

int cnt = 0;

while (q != NULL)

{ q = q->next;

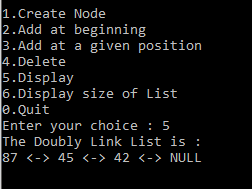
cnt++;

}

cout<<"Number of elements are: "<<cnt<<endl;

}

**Output:**

****

**Task # 4:**

Implement a circular singly linked list.

**Code:**

#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int info;

struct node \*next;

}\*last;

class list

{

public:

void create\_node(int value);

void add\_begin(int value);

void add\_at\_a\_position(int value, int position);

void delete\_element(int value);

void search\_element(int value);

void display\_list();

void update();

list()

{

last = NULL;

}

};

int main()

{ int choice, element, position;

list cl;

do

{

cout<<endl<<"---------------------------"<<endl;

cout<<"1.Create Node"<<endl;

cout<<"2.Add at beginning"<<endl;

cout<<"3.Add at a given position"<<endl;

cout<<"4.Delete"<<endl;

cout<<"5.Search"<<endl;

cout<<"6.Display"<<endl;

cout<<"7.Update"<<endl;

cout<<"0.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the element: ";

cin>>element;

cl.create\_node(element);

cout<<endl;

break;

case 2:

cout<<"Enter the element: ";

cin>>element;

cl.add\_begin(element);

cout<<endl;

break;

case 3:

cout<<"Enter the element: ";

cin>>element;

cout<<"Insert element after position: ";

cin>>position;

cl.add\_at\_a\_position(element, position);

cout<<endl;

break;

case 4:

cout<<"Enter the element for deletion: ";

cin>>element;

cl.delete\_element(element);

cout<<endl;

break;

case 5:

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

cin>>element;

cl.search\_element(element);

cout<<endl;

break;

case 6:

cl.display\_list();

break;

case 7:

cl.update();

break;

}

}while(choice != 0);

return 0;

}

void list::create\_node(int value)

{ struct node \*temp;

temp = new(struct node);

temp->info = value;

if (last == NULL)

{ last = temp;

temp->next = last;

}

else

{ temp->next = last->next;

last->next = temp;

last = temp;

}

}

void list::add\_begin(int value)

{

if (last == NULL)

{ cout<<"List does not exit"<<endl; }

struct node \*temp;

temp = new(struct node);

temp->info = value;

temp->next = last->next;

last->next = temp;

}

void list::add\_at\_a\_position(int value, int pos)

{ if (last == NULL)

{ cout<<"List does not exit"<<endl; }

struct node \*temp, \*s;

s = last->next;

for (int i = 0;i < pos-1;i++)

{ s = s->next;

if (s == last->next)

{ cout<<"Position out of range"; }

}

temp = new(struct node);

temp->next = s->next;

temp->info = value;

s->next = temp;

if (s == last)

{ last=temp; }

}

void list::delete\_element(int value)

{ struct node \*temp, \*s;

s = last->next;

if (last->next == last && last->info == value)

{ temp = last;

last = NULL;

free(temp); }

if (s->info == value)

{ temp = s;

last->next = s->next;

free(temp); }

while (s->next != last)

{

if (s->next->info == value)

{ temp = s->next;

s->next = temp->next;

free(temp);

cout<<"Element deleted"<<endl; }

s = s->next;

}

if (s->next->info == value)

{ temp = s->next;

s->next = last->next;

free(temp);

last = s; }

else

{ cout<<"Element not found"; }

}

void list::search\_element(int value)

{

struct node \*s;

int counter = 0;

s = last->next;

while (s != last)

{

counter++;

if (s->info == value)

{

cout<<"Element "<<value;

cout<<" found at position "<<counter<<endl;

return;

}

s = s->next;

}

if (s->info == value)

{

counter++;

cout<<"Element "<<value;

cout<<" found at position "<<counter<<endl;

return;

}

cout<<"Element "<<value<<" not found in the list"<<endl;

}

void list::display\_list()

{

struct node \*s;

if (last == NULL)

{

cout<<"List is empty, nothing to display"<<endl;

return;

}

s = last->next;

cout<<"Circular Link List: "<<endl;

while (s != last)

{ cout<<s->info<<"->";

s = s->next;

}

cout<<s->info<<endl;

}

void list::update()

{

int value, pos, i;

if (last == NULL)

{

cout<<"List is empty, nothing to update"<<endl;

return;

}

cout<<"Enter the node position to be updated: ";

cin>>pos;

cout<<"Enter the new value: ";

cin>>value;

struct node \*s;

s = last->next;

for (i = 0;i < pos - 1;i++)

{

if (s == last)

{

cout<<"There are less than "<<pos<<" elements.";

cout<<endl;

return;

}

s = s->next;

}

s->info = value;

cout<<"Node Updated"<<endl;

}

**Output:**

