**Question 1: Write a program to delete the Ith node of the given linked list. Be sure that such a node exists.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void deleteinode(int);

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void SLList<T>::deleteinode(int i)

{

int count=0;

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

node<T> \*prev ,\*cur;

prev=head;

cur=head->next;

if(i==1)

{

head=head->next;

delete (prev);

}

else if(i>1 && i<=count)

{

for(int j=1;j<(i-1);j++)

{

prev=cur;

cur=cur->next;

}

prev->next=cur->next;

if(cur==tail)

tail=prev;

delete(cur);

}

else

cout<<"\n Element not found";

}

void main()

{

SLList<int> L, L2, LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. DELETE I th NODE";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n DELETE I th NODE";

cout<<"\n enter node to be deleted";

cin>>x;

L.deleteinode(x);

break;

default: cout<<"\n Wrong Choice";

}

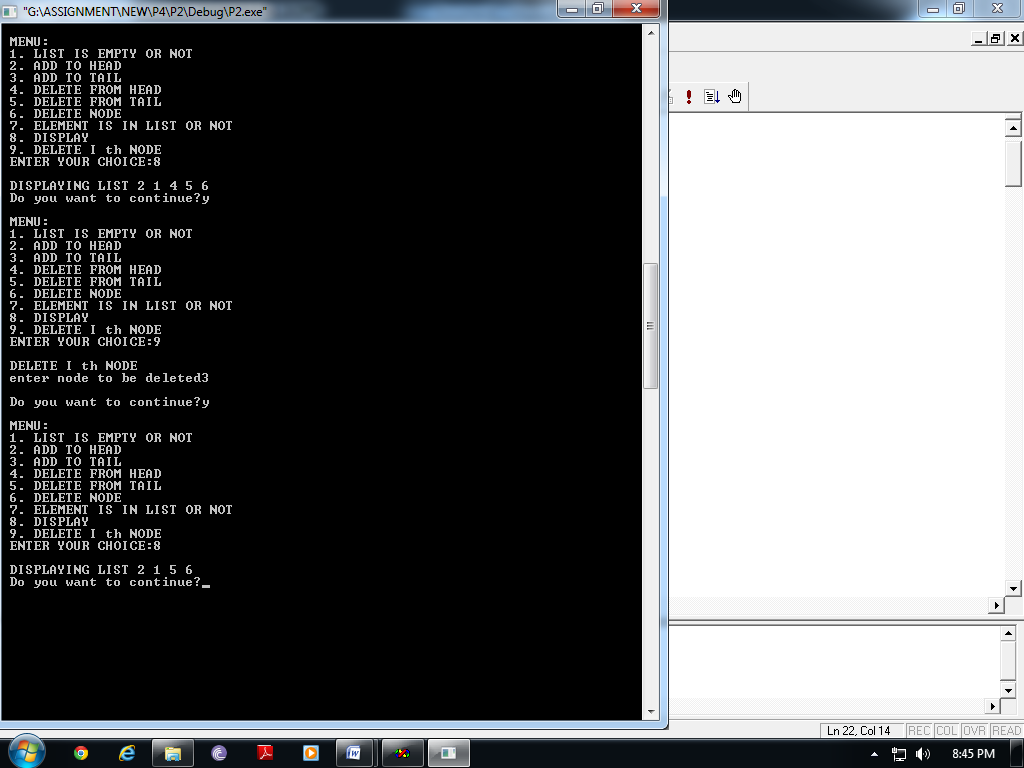
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 2: Delete from list l1 , nodes whose positions are to be found in an ordered list l2. For instance if l1=(a b c d e) and l2=(2 4 8) , then the second and the fourth nodes are to be deleted form list l1(the eighth node does not exist), and after deletion l1=(a c e).**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void deleteinode(int);

void deleteele(SLList<int>);

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void SLList<T>::deleteinode(int i)

{

int count=0;

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

node<T> \*prev ,\*cur;

prev=head;

cur=head->next;

if(i==1)

{

head=head->next;

delete (prev);

}

else if(i>1 && i<=count)

{

for(int j=1;j<(i-1);j++)

{

prev=cur;

cur=cur->next;

}

prev->next=cur->next;

if(cur==tail)

tail=prev;

delete(cur);

}

else

cout<<"\n Element not found";

}

template <class T>

void SLList<T>::deleteele(SLList<int> L2)

{

T x;

while(!(L2.isempty()))

{

x=L2.deletefromtail();

deleteinode(x);

}

}

void main()

{

SLList<char> L, L2, LREV;

SLList<int> L3;

int ch;

char x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. DELETE POSITIONS IN LIST 2";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

ch=L.isempty();

if(ch==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n DELETE POSITIONS IN LIST 2";

op='y';

while(op=='y' || op=='Y')

{

cout<<"\n Enter element to be added=";

cin>>ch;

L3.addtotail(ch);

cout<<"\n Want to enter more?";

cin>>op;

}

L.deleteele(L3);

cout<<"\n LIST AFTER DELETION :";

L.display();

break;

default: cout<<"\n Wrong Choice";

}

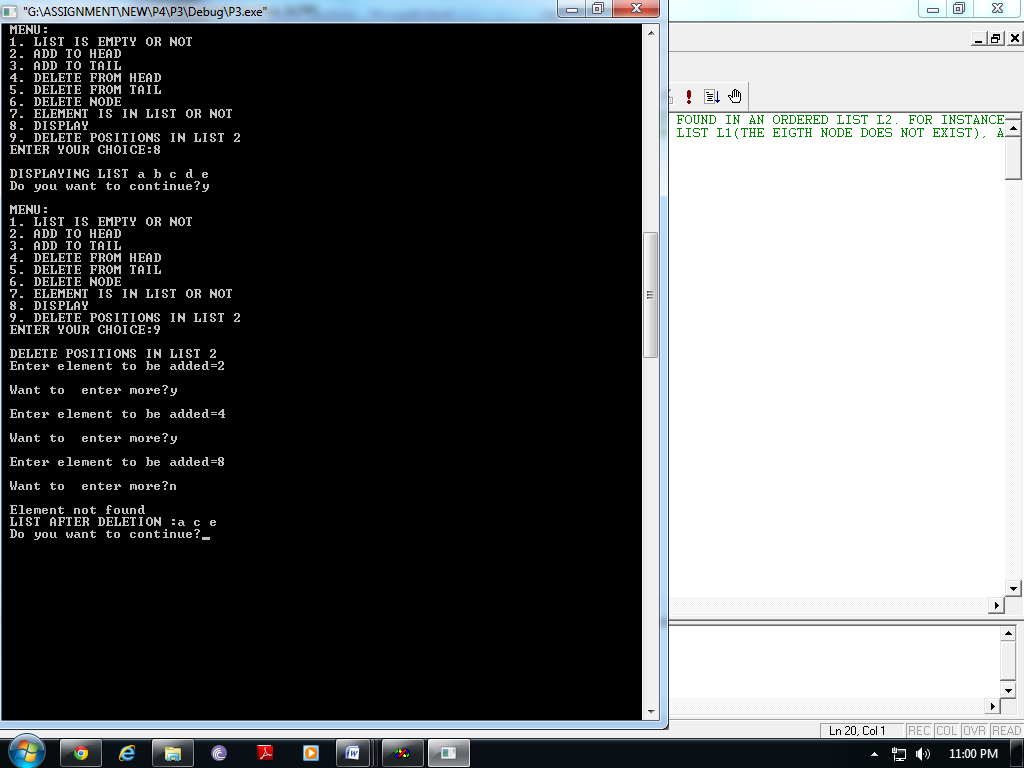
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 3: Delete from list l1, nodes occupying positions indicated in ordered lists l2 and l3. For instance, if l1=(a b c d e), l2=(2 4 8) and l3=(2 5) , then after deletion , l1=(a c)**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void deleteinode(int);

void deleteele(SLList<int>, SLList<int>);

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void SLList<T>::deleteinode(int i)

{

int count=0;

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

node<T> \*prev ,\*cur;

prev=head;

cur=head->next;

if(i==1)

{

head=head->next;

delete (prev);

}

else if(i>1 && i<=count)

{

for(int j=1;j<(i-1);j++)

{

prev=cur;

cur=cur->next;

}

prev->next=cur->next;

if(cur==tail)

tail=prev;

delete(cur);

}

else

cout<<"\n Element not found";

}

template <class T>

void SLList<T>::deleteele(SLList<int> L2, SLList<int> L3)

{

T x,y;

x=L2.deletefromtail();

y=L3.deletefromtail();

while(!L2.isempty() && !L3.isempty())

{

if(x>y)

{

deleteinode(x);

x=L2.deletefromtail();

}

else if(y>x)

{

deleteinode(y);

y=L3.deletefromtail();

}

else

{

deleteinode(x);

x=L2.deletefromtail();

y=L3.deletefromtail();

}

}

if(!L2.isempty())

{

deleteinode(x);

x=deletefromtail();

}

else if(!L3.isempty())

{

deleteinode(y);

y=deletefromtail();

}

}

void main()

{

SLList<char> L, L2, LREV;

SLList<int> L3,L4;

int ch;

char x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. DELETE POSITIONS IN LIST 2 AND LIST 3";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

ch=L.isempty();

if(ch==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n DELETE POSITIONS IN LIST 2 AND LIST 3";

cout<<"\n Enter elements for LIST 2";

if(L3.isempty())

{ op='y';

while(op=='y' || op=='Y')

{

cout<<"\n Enter element to be added=";

cin>>ch;

L3.addtotail(ch);

cout<<"\n Want to enter more?";

cin>>op;

}

}

cout<<"\n Enter elements for LIST 3";

if(L4.isempty())

{ op='y';

while(op=='y' || op=='Y')

{

cout<<"\n Enter element to be added=";

cin>>ch;

L4.addtotail(ch);

cout<<"\n Want to enter more?";

cin>>op;

}

}

L.deleteele(L3,L4);

cout<<"\n LIST AFTER DELETION :";

L.display();

break;

default: cout<<"\n Wrong Choice";

}

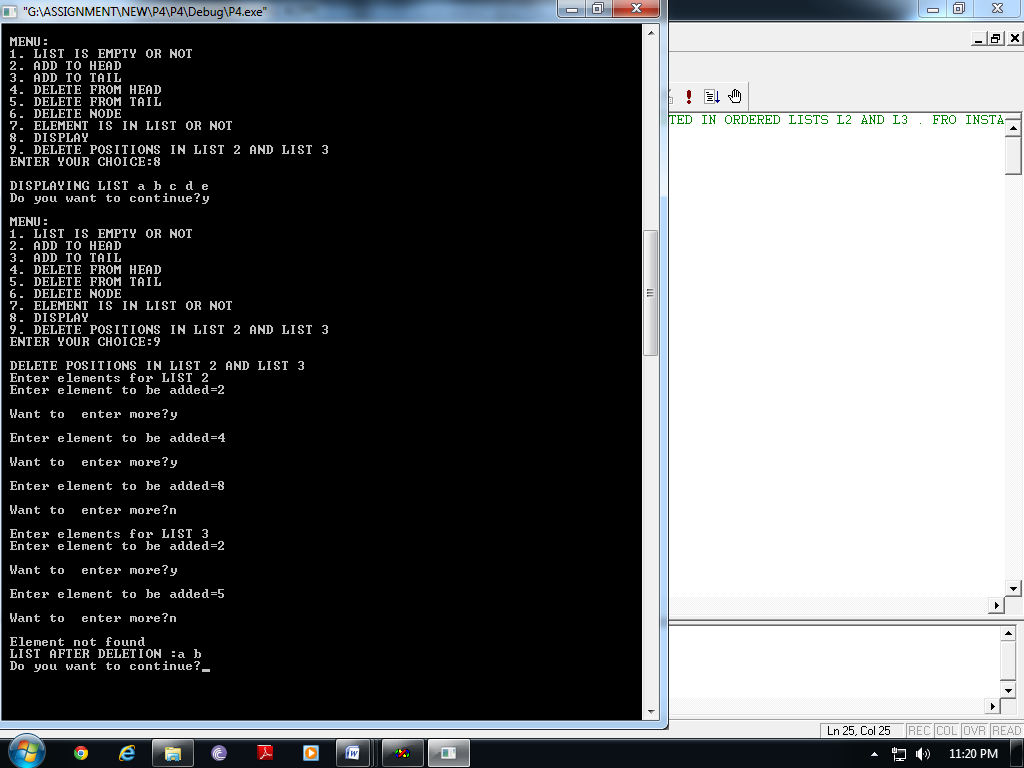
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 4: Delete from an ordered list L nodes occupying positions indicated in list L itself. For instance, if L (1 3 5 7 8) , then after deletion, L=(3,7).**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void deleteinode(int);

void deleteele();

node<T>\* ret\_hd();

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void SLList<T>::deleteinode(int i)

{

int count=0;

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

node<T> \*prev ,\*cur;

prev=head;

cur=head->next;

if(i==1)

{

head=head->next;

delete (prev);

}

else if(i>1 && i<=count)

{

for(int j=1;j<(i-1);j++)

{

prev=cur;

cur=cur->next;

}

prev->next=cur->next;

if(cur==tail)

tail=prev;

delete(cur);

}

else

cout<<"\n Element not found";

}

template <class T>

void SLList<T>::deleteele()

{

T x;

SLList<T> L;

node<T> \* temp=head;

while(temp!=NULL)

{

L.addtotail(temp->info);

temp=temp->next;

}

node<T> \* tmp=L.ret\_hd();

while(tmp!=NULL)

{

x=L.deletefromtail();

deleteinode(x);

tmp=L.ret\_hd();

}

}

template<class T>

node<T>\* SLList<T>::ret\_hd()

{

return head;

}

void main()

{

SLList<int> L, L2, LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. DELETE POSITIONS IN LIST ";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n DELETE POSITIONS IN LIST";

L.deleteele();

cout<<"\n LIST AFTER DELETION :";

L.display();

break;

default: cout<<"\n Wrong Choice";

}

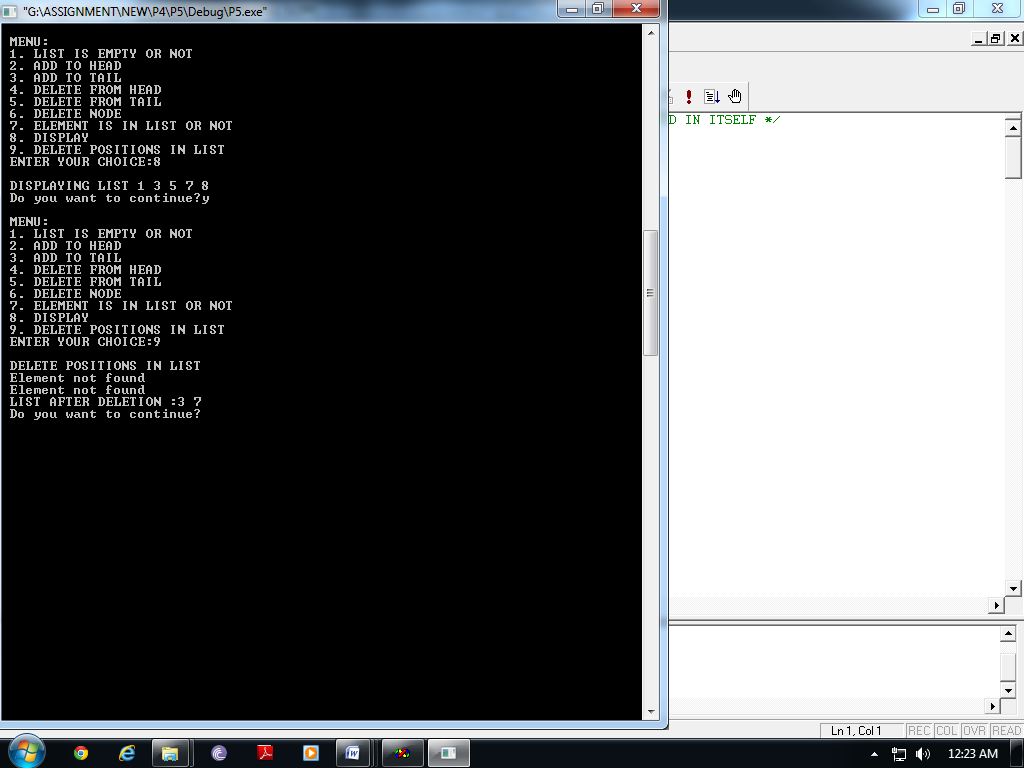
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 5:** **Write a member function to check whether two single linked lists have the same content.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

int check(SLList<T>);

node<T>\* ret\_hd();

int count();

};

template<class T>

node<T>\* SLList<T>::ret\_hd()

{

return head;

}

template<class T>

int SLList<T>::count()

{

node<T>\*tmp;

int c=0;

if(!isempty())

{

c=1;

tmp=head;

while(tmp!=tail)

{

c++;

tmp=tmp->next;

}

return c;

}

return c;

}

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

int SLList<T>::check(SLList<T> L1)

{

node<T>\* temp;

node<T>\* temp1;

temp=head;

temp1=L1.ret\_hd();

int n, n1;

n=count();

n1=L1.count();

if(n==n1)

{

while(temp!=NULL)

{

if(temp->info!=temp1->info)

return 1;

temp=temp->next;

temp1=temp1->next;

}

return 0;

}

return 1;

}

void main()

{

SLList<int> L, L2,L3, LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. CHECK EQUALITY";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n CHECK EQUALITY ";

if(L.isempty()==0)

{

cout<<"\n Enter elements of the list 2:";

op='y';

while(op=='y' || op=='Y')

{

cout<<"\n Enter value:";

cin>>x;

L3.addtotail(x);

cout<<"\n Enter more?";

cin>>op;

}

x=L.check(L3);

if(x==0)

cout<<"\n List are equal";

else

cout<<"\n List are unequal";

}

else

cout<<"\n NO ELEMENTS TO BE COMPARED";

break;

default: cout<<"\n Wrong Choice";

}

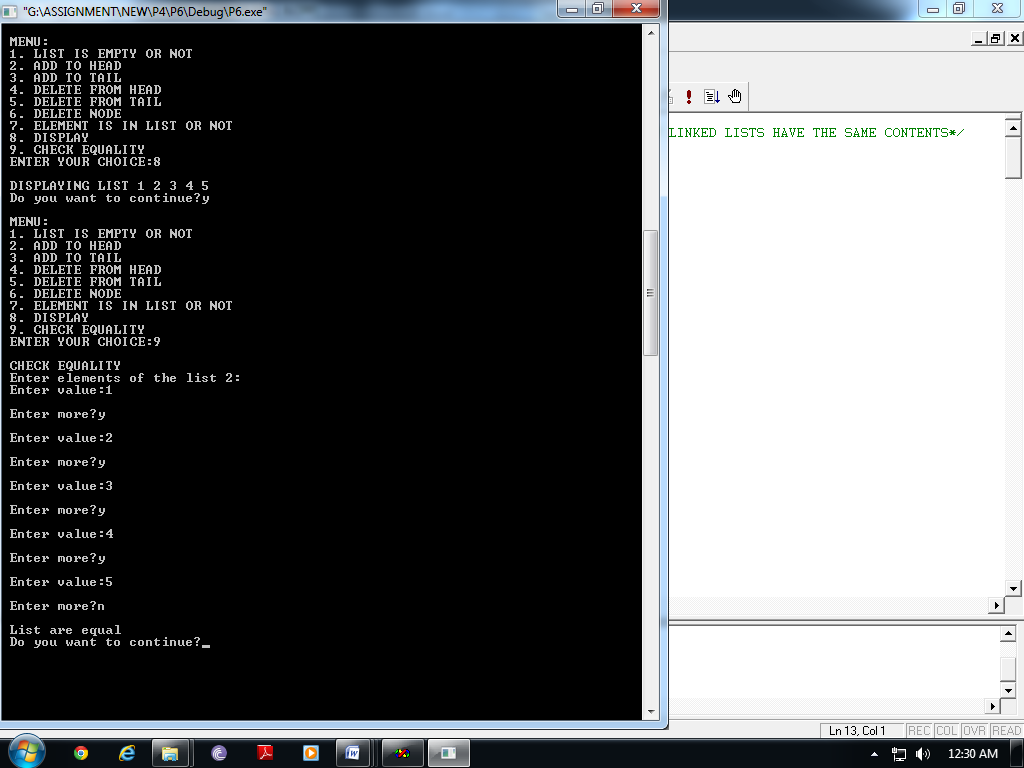
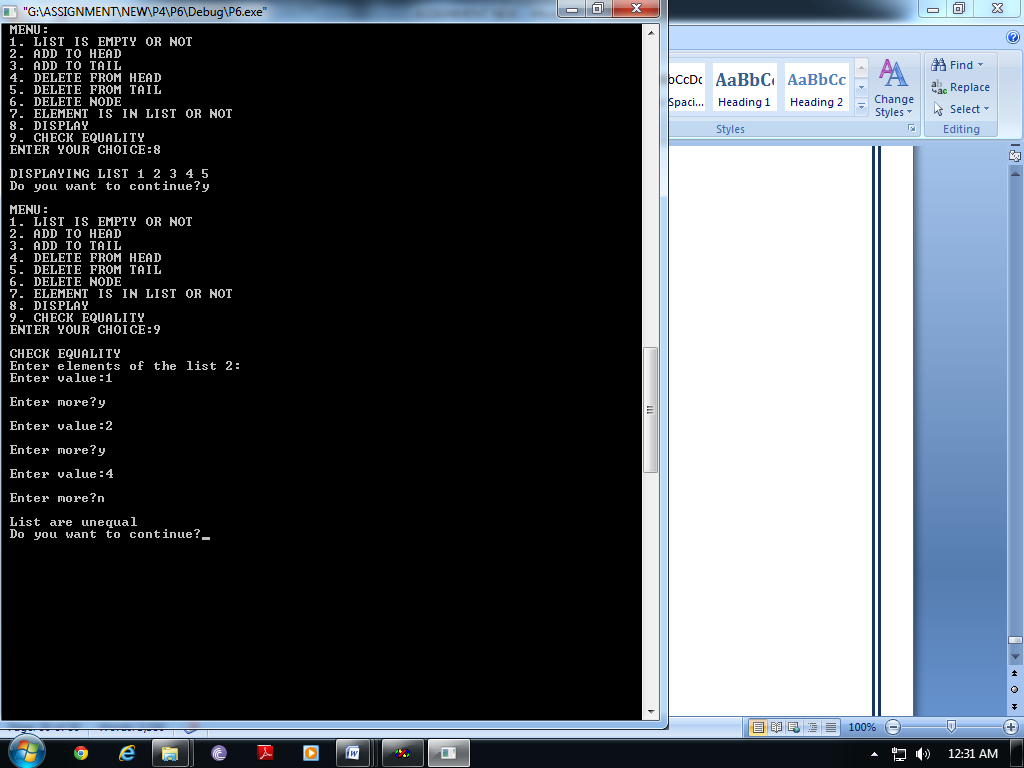
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

** **

**Question 6:** **Write a member function to reverse a single linked list using only one pass through the list.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

int c;

node<T>\* temp1[50];

node<T> \*head,\*tail;

public:

SLList()

{

c=-1;

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void reverse();

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int SLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void SLList<T>::reverse()

{

node<T>\* temp=head;

int i;

while(temp!=0)

{

temp1[++c]=temp;

temp=temp->next;

}

for(i=c;i>0;i--)

{

temp1[i]->next=temp1[i-1];

}

temp1[i]->next=NULL;

head=temp1[c];

}

void main()

{

SLList<int> L, L2;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. REVERSE";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"REVERSE";

L.reverse();

L.display();

break;

default: cout<<"\n Wrong Choice";

}

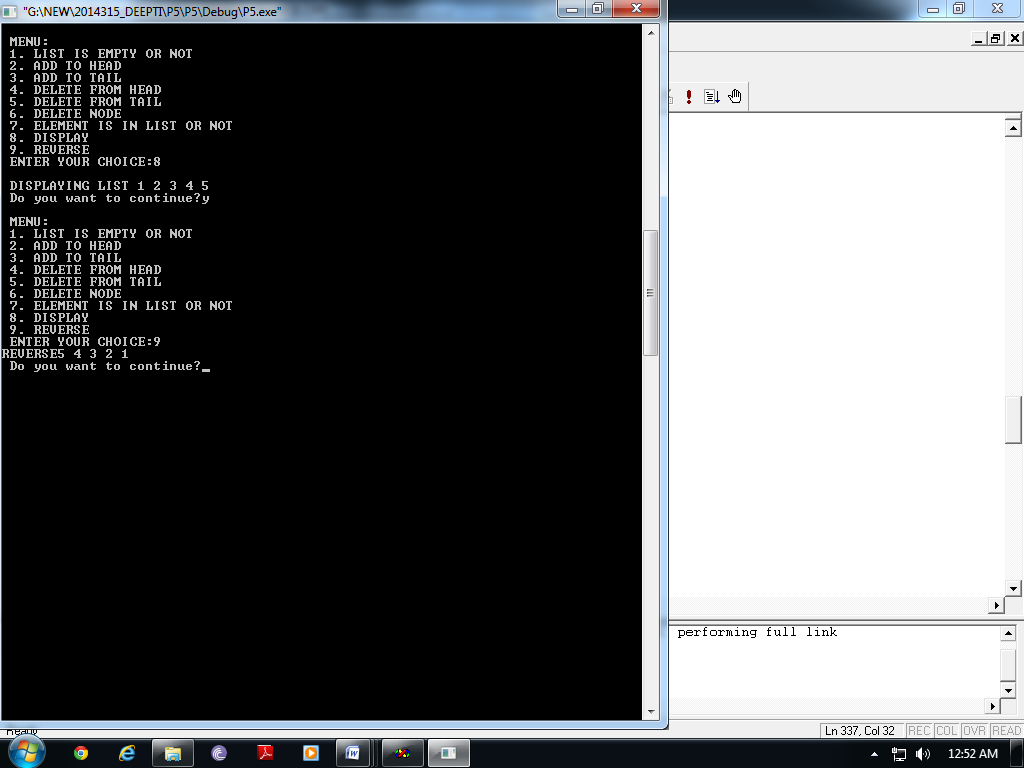
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 7: Write a program to insert a node in the middle of a double linked list.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next,\*prev;

node(T x, node \*n=0, node \*p=0)

{

info=x;

next=n;

prev=p;

}

};

template<class T>

class DLList

{

node<T> \*head,\*tail;

public:

DLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

void insert\_mid(T);

};

template<class T>

int DLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void DLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head->prev=temp;

head=temp;

}

}

template<class T>

void DLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

temp->prev=tail;

tail=temp;

}

}

template<class T>

T DLList<T>::deletefromhead()

{

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

head=head->next;

delete(head->prev);

head->prev=NULL;

}

return x;

}

template <class T>

T DLList<T>::deletefromtail()

{

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

tail=tail->prev;

delete(tail->next);

tail->next=NULL;

}

return x;

}

template <class T>

void DLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*temp;

temp=head->next;

while(temp!=0 && temp->info!=x)

{

temp=temp->next;

}

if(temp!=0)

{

temp->prev->next=temp->next;

if(temp=tail)

{

tail=tail->prev;

}

else

{

temp->next->prev=temp->prev;

delete temp;

}

}

else

cout<<"\n Element Not FOUND";

}

}

template<class T>

void DLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

int DLList<T>::isinlist(T x)

{

node<T> \*temp;

temp=head;

while(temp!=NULL)

{

if(temp->info==x)

{

return 1;

}

temp=temp->next;

}

return 0;

}

template<class T>

void DLList<T>::insert\_mid(T x)

{

node<T> \*temp=head;

node<T> \*temp1=new node<T>(x);

int i,count=0;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

if(head==NULL)

{

tail=head=temp1;

}

else if(head==tail)

{

tail=temp1;

head->next=temp1;

temp1->prev=head;

}

else

{

temp=head;

for(i=1;i<(count/2);i++)

temp=temp->next;

temp1->next=temp->next;

temp1->prev=temp;

temp->next->prev=temp1;

temp->next=temp1;

}

}

void main()

{

DLList<int> L,L2,LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. ELEMENT IS IN LIST OR NOT";

cout<<"\n 8. DISPLAY";

cout<<"\n 9. INSERT IN MIDDLE";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n ELEMENT IS IN LIST OR NOT?";

if(L.isempty()==0)

{cout<<"\n Enter a value you want to search in list:";

cin>>x;

int status;

status=L.isinlist(x);

if(status==1)

cout<<"\n Element found";

else

cout<<"\n Element not found";

}

else

{

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

}

break;

case 8: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 9: cout<<"\n Enter element to be inserted in middle ";

cin >>x;

L.insert\_mid(x);

break;

default: cout<<"\n Wrong Choice";

}

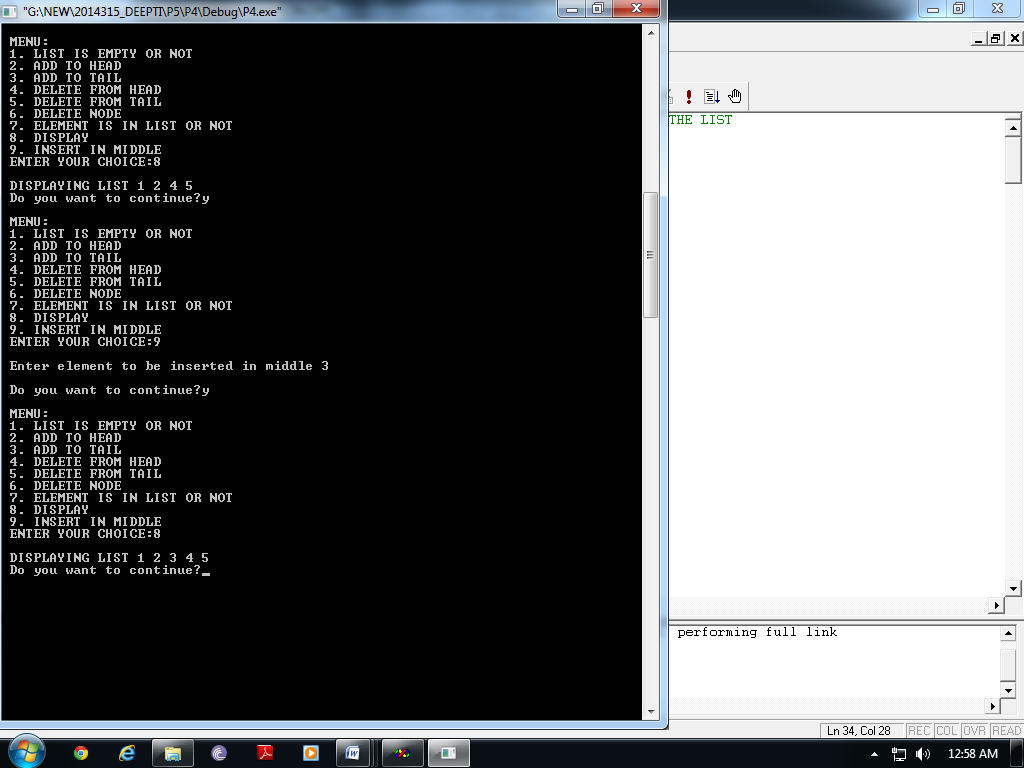
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 8: Write a program to reverse a stack using one additional stack and some additional variables.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template<class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=NULL)

{

info=x;

next=n;

}

};

template<class T>

class stack

{

node<T> \*top;

public:

stack()

{

top=NULL;

}

void push(T);

T pop();

void display();

T topel();

void clear();

int isempty();

void reverse();

};

template<class T>

void stack<T>::reverse()

{

stack<node<T>\*> S2;

node<T>\* temp;

if(!isempty())

{

temp=top;

while(temp!=NULL)

{

S2.push(temp);

temp=temp->next;

}

}

top=S2.pop();

temp=top;

while(!S2.isempty())

{

temp->next=S2.pop();

temp=temp->next;

}

temp->next=NULL;

}

template<class T>

void stack<T>::push(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

top=temp;

else

{

temp->next=top;

top=temp;

}

}

template<class T>

int stack<T>::isempty()

{

if(top==NULL)

return 1;

else

return 0;

}

template<class T>

T stack<T>::pop()

{

node<T> \*temp;

T x;

x=top->info;

if(top->next==NULL)

{

delete top;

top=NULL;

}

else

{

temp=top;

top=top->next;

delete temp;

}

return x;

}

template<class T>

void stack<T>::display()

{

node<T> \*temp;

temp=top;

while(temp!=NULL)

{

cout<<temp->info;

temp=temp->next;

}

}

template<class T>

T stack<T>::topel()

{

T x;

x=top->info;

return(x);

}

template<class T>

void stack<T>::clear()

{

T x;

while(top!=NULL)

x=pop();

top=NULL;

}

void main()

{

stack<int> S1;

int x, ch;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. PUSH ELEMENT";

cout<<"\n 2. POP ELEMENT";

cout<<"\n 3. TOP MOST ELEMENT";

cout<<"\n 4. DISPLAY";

cout<<"\n 5. CLEAR";

cout<<"\n 6. REVERSE";

cout<<"\n ENTER A CHOICE";

cin>>ch;

switch(ch)

{

case 1:cout<<"\n PUSH";

cout<<"\n Enter an element";

cin>>x;

S1.push(x);

break;

case 2:cout<<"n POP";

if(S1.isempty()==0)

{

x=S1.pop();

cout<<"\n Element popped:"<<x<<endl;

}

else

cout<<"\n UNDERFLOW";

break;

case 3:x=S1.topel();

cout<<"\n TOP MOST ELEMNT:"<<x;

break;

case 4:if(S1.isempty()==0)

{

S1.display();

}

else

cout<<"\n No element to be displayed";

break;

case 5:S1.clear();

break;

case 6:S1.reverse();

S1.display();

break;

}

cout<<"\n Do you want to continue?";

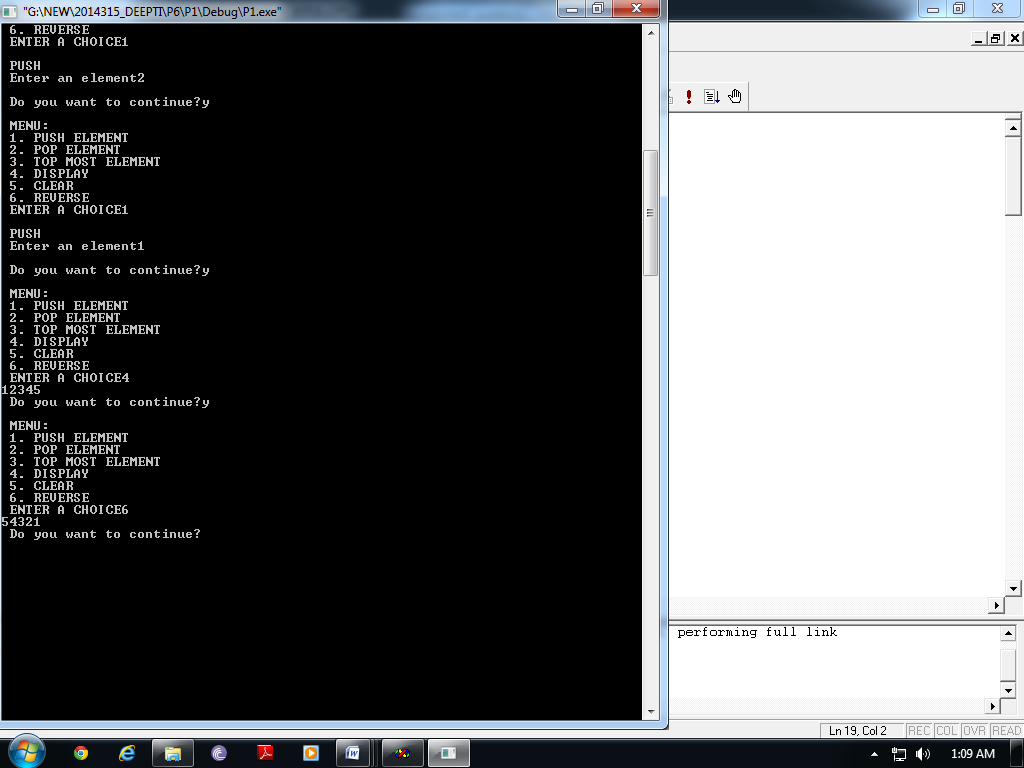
cin>>op;

}

while(op=='Y' || op=='y');

}

**Output:**

****

**Question 9: Write a recursive method that for a positive integer n print odd numbers**

1. **Between 1 and n**
2. **Between n and 1**

***Solution:***

***//1.Between 1 and n***

***#include<iostream>***

***using namespace std;***

class prn

{

public:

void prnodd(int n)

{

if(n%2==0)

n--;

if(n>=1 && n%2==1)

{

prnodd(n-2);

cout<<" "<<n;

}

}

};

void main()

{

int max;

prn P;

cout<<"\n Enter the number =";

cin>>max;

P.prnodd(max);

}

***//2. Between N to 1***

***#include<iostream>***

***using namespace std;***

class prn

{

public:

void prnodd(int n)

{

if(n%2==0)

n--;

if(n>=1 && n%2==1)

{

cout<<" "<<n;

prnodd(n-2);

}

}

};

void main()

{

int max;

prn P;

cout<<"\n Enter the number =";

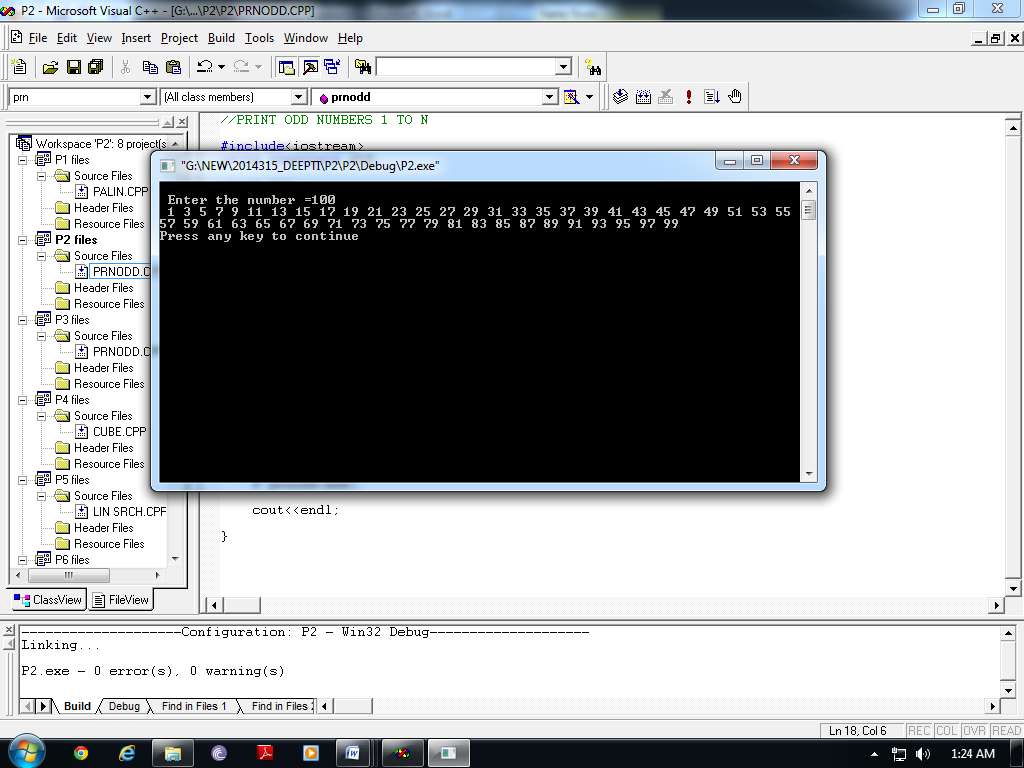
cin>>max;

P.prnodd(max);

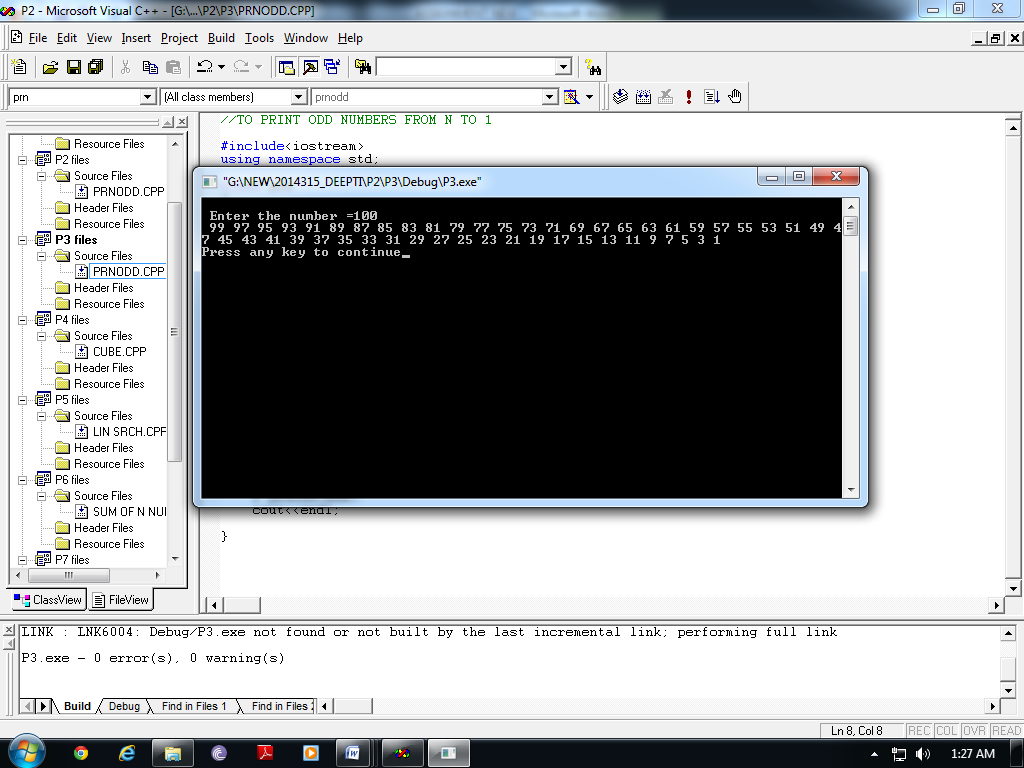
}

**Output:**

1. **Between 1 and N**



1. **Between N and 1**

****

**Question 10: Write a function**

1. **Count the number of right children.**
2. **To count the number of nodes in binary tree.**
3. **Delete all leaves from a binary tree.**
4. **Check whether binary tree is binary search tree or not.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template<class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=NULL)

{

info=x;

next=n;

}

};

//STACK CLASS

template<class T>

class stack

{

node<T> \*top;

public:

stack()

{

top=NULL;

}

void push(T);

T pop();

void display();

int isempty();

};

template<class T>

void stack<T>::push(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

top=temp;

else

{

temp->next=top;

top=temp;

}

}

template<class T>

int stack<T>::isempty()

{

if(top==NULL)

return 1;

else

return 0;

}

template<class T>

T stack<T>::pop()

{

node<T> \*temp;

T x=top->info;

if(top->next==NULL)

{

delete top;

top=NULL;

}

else

{

temp=top;

top=top->next;

delete temp;

}

return x;

}

template<class T>

void stack<T>::display()

{

node<T> \*temp;

temp=top;

while(temp!=NULL)

{

cout<<temp->info;

temp=temp->next;

}

}

// QUEUE CLASS

template<class T>

class queue

{

public:

node<T> \*front,\*rear;

queue()

{

front=rear=NULL;

}

void enqueue(T);

T dequeue();

void display();

int isempty();

int sort();

};

template<class T>

void queue<T>::enqueue(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

front=rear=temp;

else

{

rear->next=temp;

rear=temp;

}

}

template<class T>

int queue<T>::isempty()

{

if(front==NULL)

return 1;

else

return 0;

}

template<class T>

T queue<T>::dequeue()

{

node<T> \*temp;

T x=front->info;

if(front==rear)

{

delete front;

front=rear=NULL;

}

else

{

temp=front;

front=front->next;

delete temp;

}

return x;

}

template<class T>

void queue<T>::display()

{

node<T> \*temp;

temp=front;

while(temp!=NULL)

{

cout<<temp->info;

temp=temp->next;

}

}

template<class T>

int queue<T>::sort()

{

node<T> \*tmp;

tmp=front;

while(tmp->next!=NULL)

{

if(tmp->info>tmp->next->info)

return 0;

tmp=tmp->next;

}

return 1;

}

//BST NODE CLASS

template<class T>

class BSTnode

{

public:

T info;

BSTnode \*left,\*right;

BSTnode(T x, BSTnode \*l=0,BSTnode \*r=0)

{

info=x;

left=l;

right=r;

}

};

//TREE CLASS

template<class T>

class Tree

{

BSTnode<T> \*root;

public:

Tree()

{

root=NULL;

h=0;

}

int isempty()

{

if(root==NULL)

return 1;

else

return 0;

}

void insertion(T x);

int search(T x);

void preorder();

void postorder();

void inorder();

void preorder(BSTnode<T>\*);

void postorder(BSTnode<T>\*);

void inorder(BSTnode<T>\*);

void iterativepreorder();

void iterativepostorder();

void iterativeinorder();

void Breadthfirst();

void findanddeletebymerge(T x);

void findanddeletebycopy(T x);

void deletebymerge(BSTnode<T>\*&);

void deletebycopy(BSTnode<T>\*&);

int count\_right();

void count\_nodes();

void del\_leaves();

int check();

};

template<class T>

void Tree<T>::insertion(T x)

{

BSTnode<T> \*temp=new BSTnode<T>(x);

BSTnode<T> \*p,\*prev;

prev=NULL;

p=root;

while(p!=NULL)

{

prev=p;

if(p->info>x)

p=p->left;

else

p=p->right;

}

if(prev==0)

root=temp;

else if(prev->info>x)

prev->left=temp;

else

prev->right=temp;

}

template<class T>

int Tree<T>::search(T x)

{

BSTnode<T> \*p;

p=root;

while(p!=NULL)

{

if (p->info==x)

return 1;

else if(p->info>x)

p=p->left;

else

p=p->right;

}

return 0;

}

template<class T>

void Tree<T>::Breadthfirst()

{

queue<BSTnode<T>\*> Q;

BSTnode<T> \*p=root;

Q.enqueue(p);

while(!Q.isempty())

{

p=Q.dequeue();

cout<<p->info<<" ";

if(p->left!=0)

Q.enqueue(p->left);

if(p->right!=0)

Q.enqueue(p->right);

}

}

template<class T>

void Tree<T>::preorder()

{

preorder(root);

}

template<class T>

void Tree<T>::preorder(BSTnode<T>\* p)

{

if(p!=0)

{

cout<<p->info<<" ";

preorder(p->left);

preorder(p->right);

}

}

template<class T>

void Tree<T>::postorder()

{

postorder(root);

}

template<class T>

void Tree<T>::postorder(BSTnode<T>\* p)

{

if(p!=0)

{

postorder(p->left);

postorder(p->right);

cout<<p->info<<" ";

}

}

template<class T>

void Tree<T>::inorder()

{

inorder(root);

}

template<class T>

void Tree<T>::inorder(BSTnode<T>\* p)

{

if(p!=0)

{

inorder(p->left);

cout<<p->info<<" ";

inorder(p->right);

}

}

template<class T>

void Tree<T>::iterativepreorder()

{

stack<BSTnode<T>\*> S1;

BSTnode<T> \*p,\*V;

p=root;

if(p!=NULL)

{

S1.push(p);

while(!S1.isempty())

{

V=S1.pop();

cout<<V->info<<" ";

if(V->right!=0)

S1.push(V->right);

if(V->left!=0)

S1.push(V->left);

}

}

}

template<class T>

void Tree<T>::iterativeinorder()

{

stack<BSTnode<T>\*> S1;

BSTnode<T> \*p;

p=root;

while(p!=NULL)

{

while(p!=0)

{

if(p->right)

S1.push(p->right);

S1.push(p);

p=p->left;

}

p=S1.pop();

while(!S1.isempty() && p->right==0)

{

cout<<p->info<<" ";

p=S1.pop();

}

cout<<p->info;

if(!S1.isempty())

p=S1.pop();

else

p=0;

}

}

template<class T>

void Tree<T>::iterativepostorder()

{

stack<BSTnode<T>\*> S1;

BSTnode<T> \*p,\*q;

p=q=root;

while(p!=NULL)

{

for(;p->left!=0;p=p->left)

S1.push(p);

while(p!=0 && (p->right==0 || p->right==q))

{

cout<<p->info<<" ";

q=p;

if(S1.isempty())

return;

p=S1.pop();

}

S1.push(p);

p=p->right;

}

}

template<class T>

void Tree<T>::findanddeletebymerge(T x)

{

BSTnode<T> \*p,\*prev;

p=root;

while(p!=NULL)

{

if(p->info==x)

break;

prev=p;

if(p->info<x)

p=p->right;

else

p=p->left;

}

if(p!=0 && p->info==x)

{

if(p==root)

deletebymerge(root);

else if(prev->left==p)

deletebymerge(prev->left);

else

deletebymerge(prev->right);

}

else if(root!=0)

cout<<"\n "<<x<<" is not element in the tree";

else

cout<<"\n The tree is empty";

}

template<class T>

void Tree<T>::deletebymerge(BSTnode<T>\*& p)

{

BSTnode<T> \*tmp=p;

if(p!=0)

{

if(p->right==0)

p=p->left;

else if(p->left==0)

p=p->right;

else

{

tmp=p->left;

while(tmp->right!=0)

tmp=tmp->right;

tmp->right=p->right;

tmp=p;

p=p->left;

}

delete tmp;

}

}

template<class T>

void Tree<T>::findanddeletebycopy(T x)

{

BSTnode<T> \*p,\*prev;

p=root;

while(p!=NULL)

{

if(p->info==x)

break;

prev=p;

if(p->info<x)

p=p->right;

else

p=p->left;

}

if(p!=0 && p->info==x)

{

if(p==root)

deletebycopy(root);

else if(prev->left==p)

deletebycopy(prev->left);

else

deletebycopy(prev->right);

}

else if(root!=0)

cout<<"\n "<<x<<" is not element in the tree";

else

cout<<"\n The tree is empty";

}

template<class T>

void Tree<T>::deletebycopy(BSTnode<T>\*& p)

{

BSTnode<T> \*prev,\*tmp;

tmp=p;

if(p->right==0)

p=p->left;

else if(p->left==0)

p=p->right;

else

{

tmp=p->left;

prev=p;

while(p->right!=0)

{

prev=tmp;

tmp=tmp->right;

}

p->info=tmp->info;

if(prev==p)

prev->left=tmp->left;

else

prev->right=tmp->left;

}

delete tmp;

}

template<class T>

void Tree<T>::count\_nodes()

{

int c=0;

queue<BSTnode<T>\* > Q1;

BSTnode<T>\*p=root;

if(p!=0)

{

Q1.enqueue(root);

while(!Q1.isempty())

{

p=Q1.dequeue();

c++;

if(p->left!=0)

Q1.enqueue(p->left);

if(p->right!=0)

Q1.enqueue(p->right);

}

}

cout<<"\n NODES IN TREE:"<<c;

}

template<class T>

int Tree<T>::count\_right()

{

queue<BSTnode<T>\*> Q;

BSTnode<T> \*p=root;

int count=0;

if(p!=0)

{

Q.enqueue(p);

while(!Q.isempty())

{

p=Q.dequeue();

if(p->right!=0)

count++;

if(p->left!=0)

Q.enqueue(p->left);

if(p->right!=0)

Q.enqueue(p->right);

}

}

return count;

}

template<class T>

void Tree<T>::del\_leaves()

{

queue<BSTnode<T>\*> Q;

BSTnode<T> \*p=root;

if(p!=0)

{

Q.enqueue(p);

while(!Q.isempty())

{

p=Q.dequeue();

if(p->right==0 && p->left==0)

findanddeletebymerge(p->info);

else

{

if(p->left!=0)

Q.enqueue(p->left);

if(p->right!=0)

Q.enqueue(p->right);

}

}

}

}

template<class T>

int Tree<T>::check()

{

int status;

queue<T> Q1;

stack<BSTnode<T>\*> S1;

BSTnode<T> \*p;

p=root;

while(p!=NULL)

{

while(p!=0)

{

if(p->right)

S1.push(p->right);

S1.push(p);

p=p->left;

}

p=S1.pop();

while(!S1.isempty() && p->right==0)

{

Q1.enqueue(p->info);

p=S1.pop();

}

Q1.enqueue(p->info);

if(!S1.isempty())

p=S1.pop();

else

p=0;

}

status=Q1.sort();

return(status);

}

void main()

{

Tree<int> T1,T2;

int x,op,res;

char ch='y';

do

{

cout<<"\n MENU:";

cout<<"\n 1. INSERTION ";

cout<<"\n 2. SEARCH";

cout<<"\n 3. DELETION:";

cout<<"\n 4. TRAVERSAL";

cout<<"\n 5. COUNT RIGHT CHILD";

cout<<"\n 6. COUNT TOTAL NODES";

cout<<"\n 7. DELETE ALL LEAVES";

cout<<"\n 8. CHECK WHETHER BST OR NOT";

cout<<"\n Enter your choice:";

cin>>op;

switch(op)

{

case 1:cout<<"\n INSERTION";

cout<<"\n Enter values to be inserted into the tree=";

while(ch=='y')

{

cin>>x;

T1.insertion(x);

cout<<"\n Enter more?";

cin>>ch;

}

break;

case 2: cout<<"\n SEARCH AN ELEMENT ";

cout<<"\n Enter values to be searched into the tree=";

cin>>x;

if(!T1.isempty())

{

res= T1.search(x);

if(res==1)

cout<<"\n Elemnt found ";

else

cout<<"\n Elemnt not found";

}

else

cout<<"\n The Tree is empty";

break;

case 3:cout<<"\n DELETION";

cout<<"\n 1. DELETION BY COPYING ";

cout<<"\n 2. DELETION BY MERGING ";

cout<<"\n Enter option";

cin>>op;

if(!T1.isempty())

{

cout<<"\n Enter an element to be deleted ";

cin>>x;

if(op==1)

{

T1.findanddeletebycopy(x);

cout<<"\n After deletion";

T1.Breadthfirst();

}

else

{

T1.findanddeletebymerge(x);

cout<<"\n After deletion";

T1.Breadthfirst();

}

}

else

cout<<"\n The Tree is empty";

break;

case 4:cout<<"\n TRAVERSAL";

cout<<"\n 1. BREADTHFIRST";

cout<<"\n 2. PREORDER ";

cout<<"\n 3. POSTORDER";

cout<<"\n 4. INORDER";

cout<<"\n Enter option";

cin>>op;

if(!T1.isempty())

{

if(op==1)

{

T1.Breadthfirst();

}

else if(op==2)

{

cout<<"\n Through recursion";

T1.preorder();

cout<<"\n Through Iteration";

T1.iterativepreorder();

}

else if(op==3)

{

cout<<"\n Through recursion";

T1.postorder();

cout<<"\n Through Iteration";

T1.iterativepostorder();

}

else if(op==4)

{

cout<<"\n Through recursion";

T1.inorder();

cout<<"\n Through Iteration";

T1.iterativeinorder();

}

else

cout<<"\n Wrong Options";

}

else

cout<<"\n The Tree is empty";

break;

case 5:cout<<"\n COUNT NUMBER OF RIGHT CHILDREN";

x=T1.count\_right();

cout<<"\n NUMBER OF RIGHT CHILDREN="<<x;

break;

case 6:cout<<"\n COUNT TOTAL NODES";

if(!T1.isempty())

{

T1.count\_nodes();

}

else

cout<<"\n Tree is empty";

break;

case 7:if(! T1.isempty())

{

T1.del\_leaves();

}

break;

case 8: cout<<"\n CHECK WHETHER THE TREE IS BINARY SEARCH TREE OR NOT";

x=T1.check();

if(x==1)

cout<<"\n IT IS A BST";

else

cout<<"\n IT IS NOT A BST";

break;

}

cout<<"\n Do you want to continue";

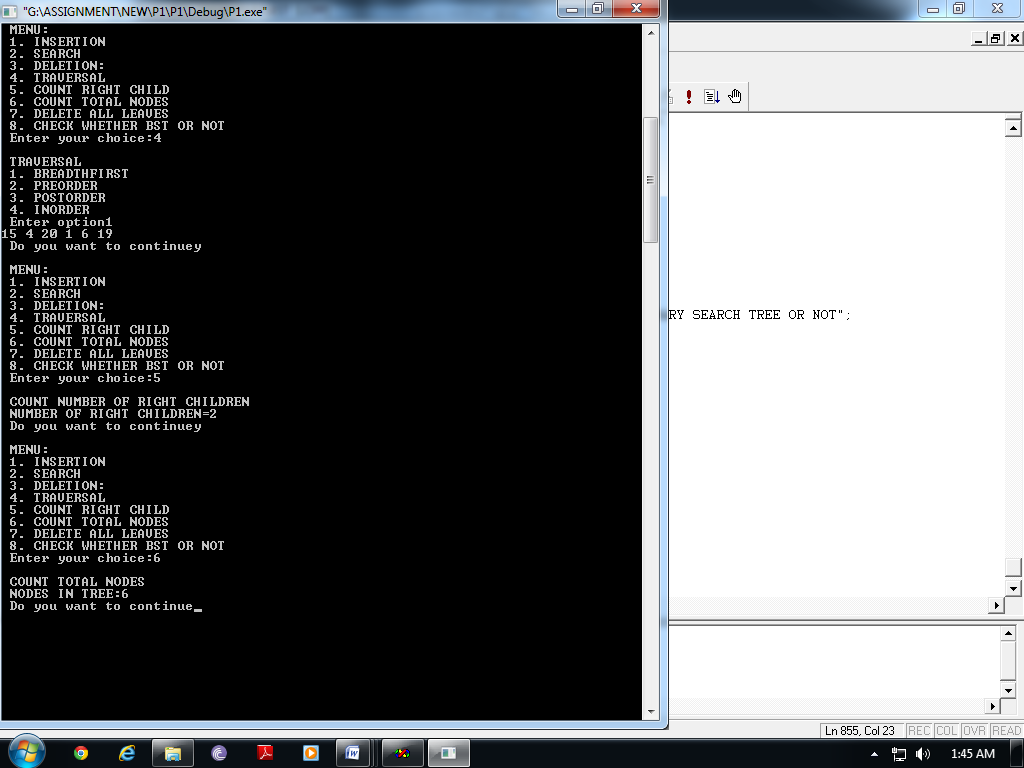
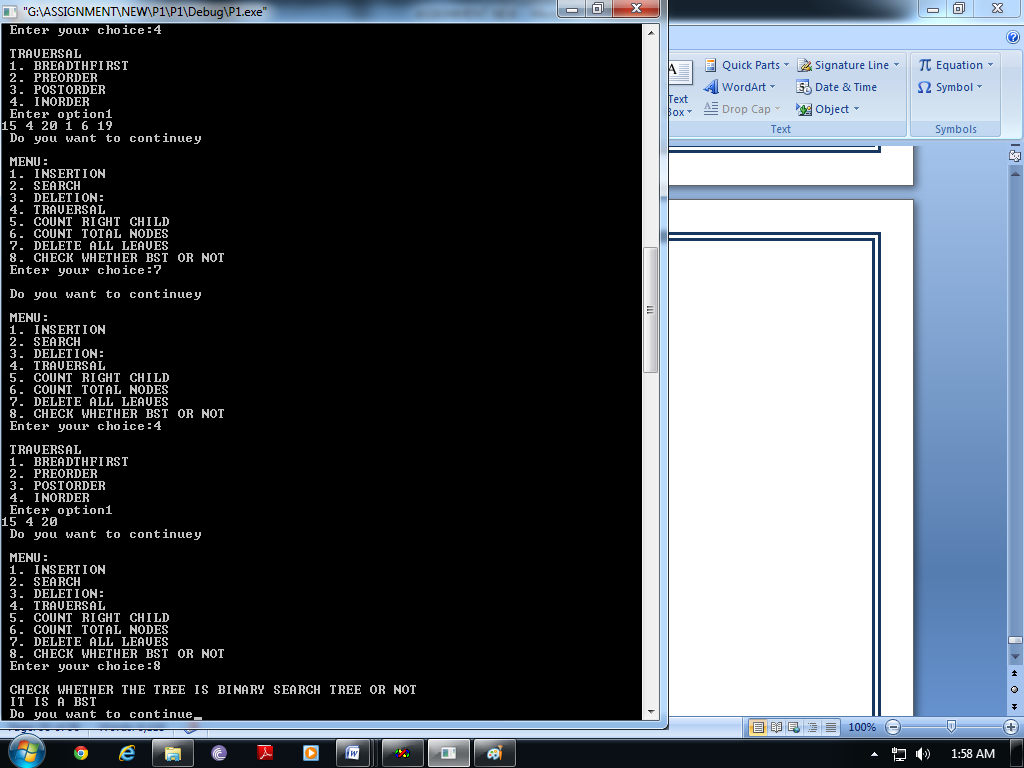
cin>>ch;

}

while(ch=='y' || ch=='Y');

}

**Output:**

** **

**Question 11: WAP to implement delimiter matching algorithm.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template<class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=NULL)

{

info=x;

next=n;

}

};

//STACK CLASS

template<class T>

class stack

{

node<T> \*top;

public:

stack()

{

top=NULL;

}

void push(T);

T pop();

void display();

T topel();

void clear();

int isempty();

int delim(char[]);

};

template<class T>

void stack<T>::push(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

top=temp;

}

else

{

temp->next=top;

top=temp;

}

}

template<class T>

int stack<T>::isempty()

{

if(top==NULL)

return 1;

else

return 0;

}

template<class T>

T stack<T>::pop()

{

node<T> \*temp;

T x;

x=top->info;

if(top->next==NULL)

{

delete top;

top=NULL;

}

else

{

temp=top;

top=top->next;

delete temp;

}

return x;

}

template<class T>

void stack<T>::display()

{

node<T> \*temp;

temp=top;

while(temp!=NULL)

{

cout<<temp->info;

temp=temp->next;

}

}

template<class T>

T stack<T>::topel()

{

T x;

x=top->info;

return(x);

}

template<class T>

void stack<T>::clear()

{

T x;

while(top!=NULL)

{

x=pop();

}

top=NULL;

}

template<class T>

int stack<T>::delim(char str[])

{

int i=0;

char ch;

while(str[i]!='\0')

{

if(str[i]=='(' || str[i]=='{' || str[i]=='[')

{

push(str[i]);

}

else if(str[i]==')' || str[i]=='}' || str[i]==']')

{

ch=pop();

if(ch=='(' && str[i]!=')')

return 0;

else if(ch=='[' && str[i]!=']')

return 0;

else if(ch=='{' && str[i]!='}')

return 0;

}

else if(str[i]=='/')

{

ch=str[i+1];

if(ch=='\*')

{

int j;

for(j=i+1; str[j]!='\*' && str[j]!='\0' ;j++);

if(str[j]!='\0')

{

if(str[j+1]=='\0')

return 0;

else if(str[j+1]!='/')

return 0;

i=j+1;

}

else

return 0;

}

else

{

i++;

continue;

}

}

i++;

}

if(isempty())

return 1;

else

return 0;

}

void main()

{

char str[50];

stack<char> S1;

int i;

cout<<"\n Enter expresion:";

cin>>str;

i=S1.delim(str);

if(i==1)

cout<<"\n Well Formed Formula";

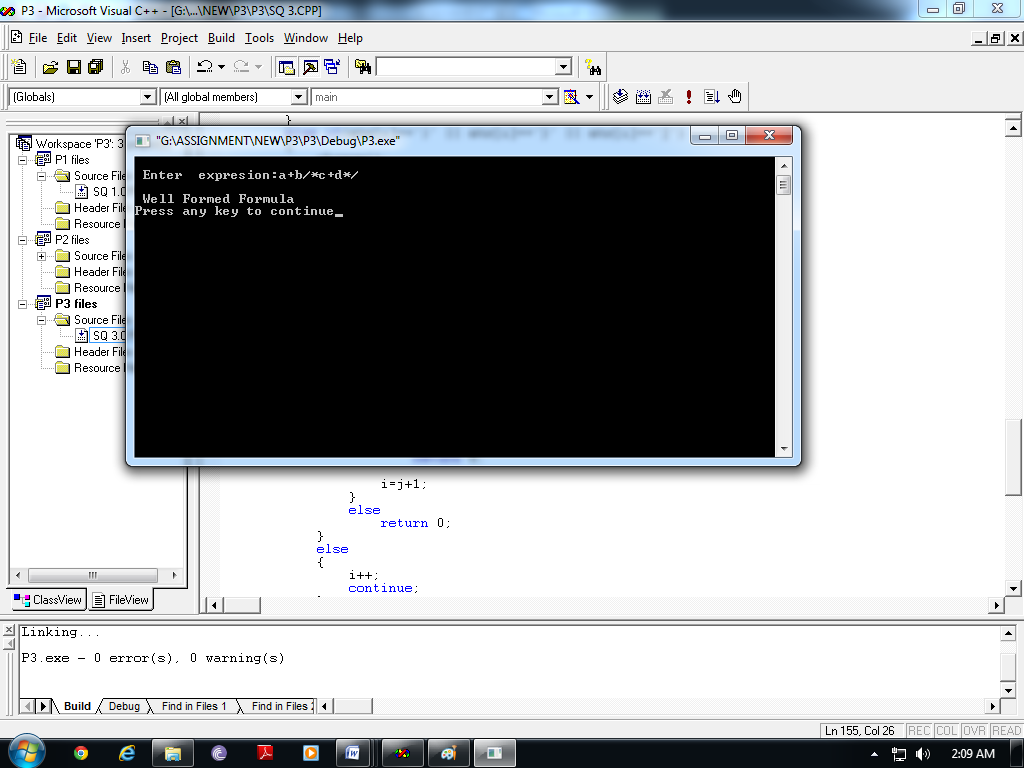
else

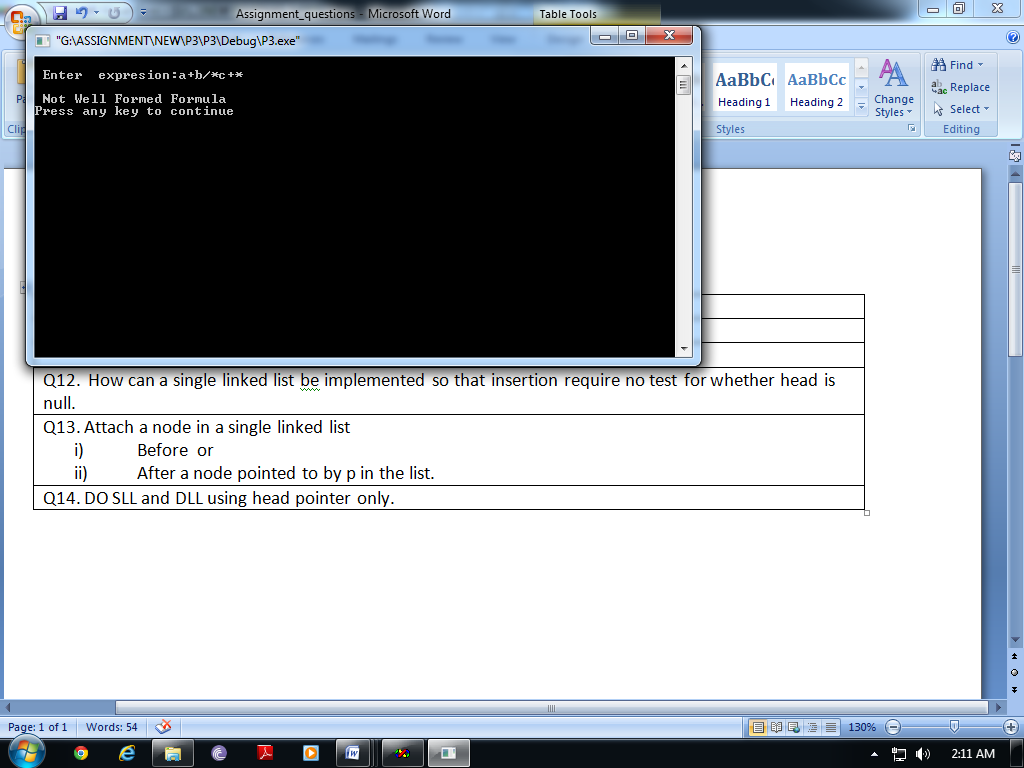
cout<<"\n Not Well Formed Formula";

cout<<endl;

}

**Output:**

****



**Question 12: How can a single linked list be implemented so that insertion require no test for whether head is null.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

void display();

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x,head);

head=temp;

if(tail==NULL)

tail=temp;

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

void main()

{

SLList<int> L, L2;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. DISPLAY";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

default: cout<<"\n Wrong Choice";

}

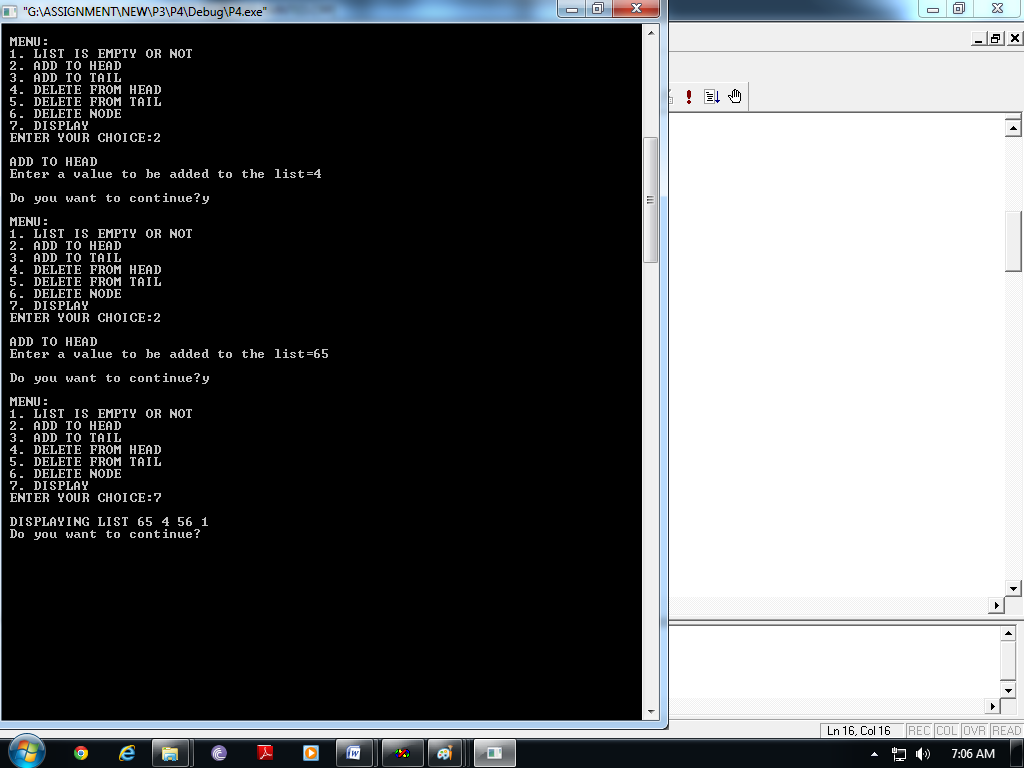
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 13: Attach a node in a single linked list**

1. **Before or**
2. **After a node pointed to by p in the list.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

void display();

void insert\_p(node<T>\*p,int cas, T x);

node<T>\* ret\_hd(){return head;}

node<T>\* ret\_tl(){return tail;}

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

void SLList<T>::insert\_p(node<T>\*p,int cas, T x)

{

node<T>\* temp=new node<T>(x);

node<T>\*temp1=head;

switch(cas)

{

case 1://before p=head

temp->next=p;

head=temp;

break;

case 2://after p=head

temp->next=p->next;

p->next=temp;

break;

case 3://before p=tail

while(temp1->next!=p)

temp1=temp1->next;

temp1->next=temp;

temp->next=p;

break;

case 4://after p=tail

p->next=temp;

tail=temp;

break;

}

if(tail==NULL)

tail=head;

if(head==NULL)

head=tail;

}

void main()

{

SLList<int> L, L2, LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. DISPLAY";

cout<<"\n 8. INSERTION USING P POINTER";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 8: cout<<"\n INSERTION USING P POINTER ";

cout<<"\n 1. BEFORE HEAD";

cout<<"\n 2. AFTER HEAD";

cout<<"\n 3. BEFORE TAIL";

cout<<"\n 4. AFTER TAIL";

cout<<"\n ENTER YOUR CHOICE";

cin>>ch;

cout<<"\n Enter a value to be added";

cin>>x;

if(ch<3)

L.insert\_p(L.ret\_hd(), ch, x);

else

L.insert\_p(L.ret\_tl(), ch, x);

L.display();

break;

default: cout<<"\n Wrong Choice";

}

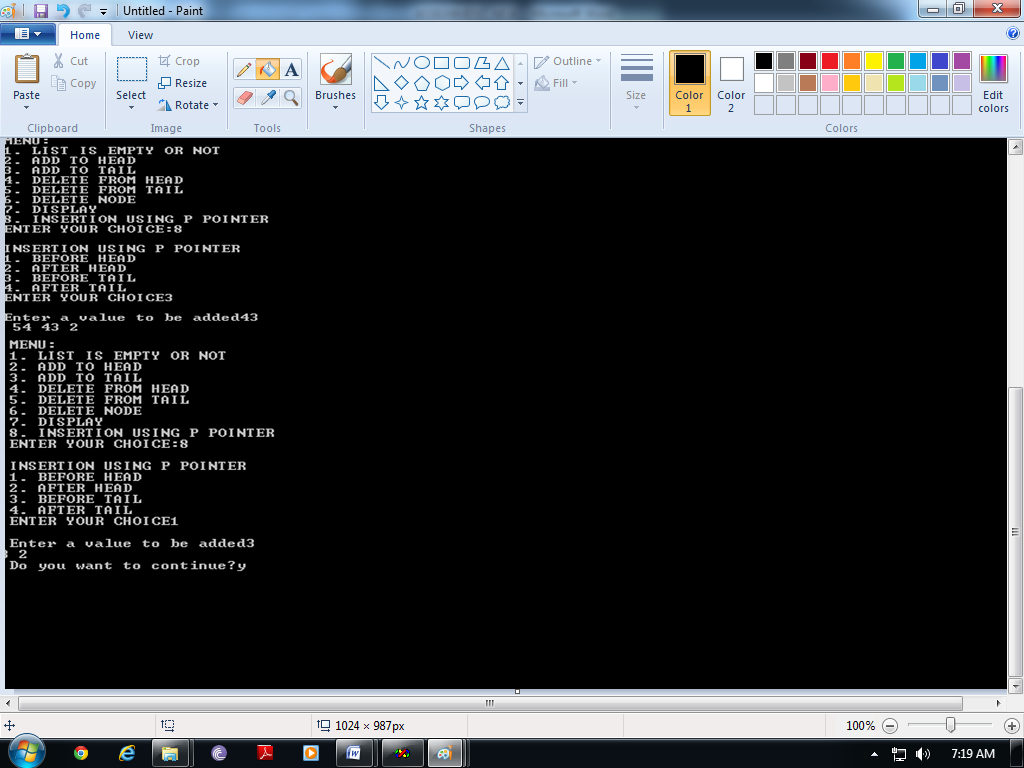
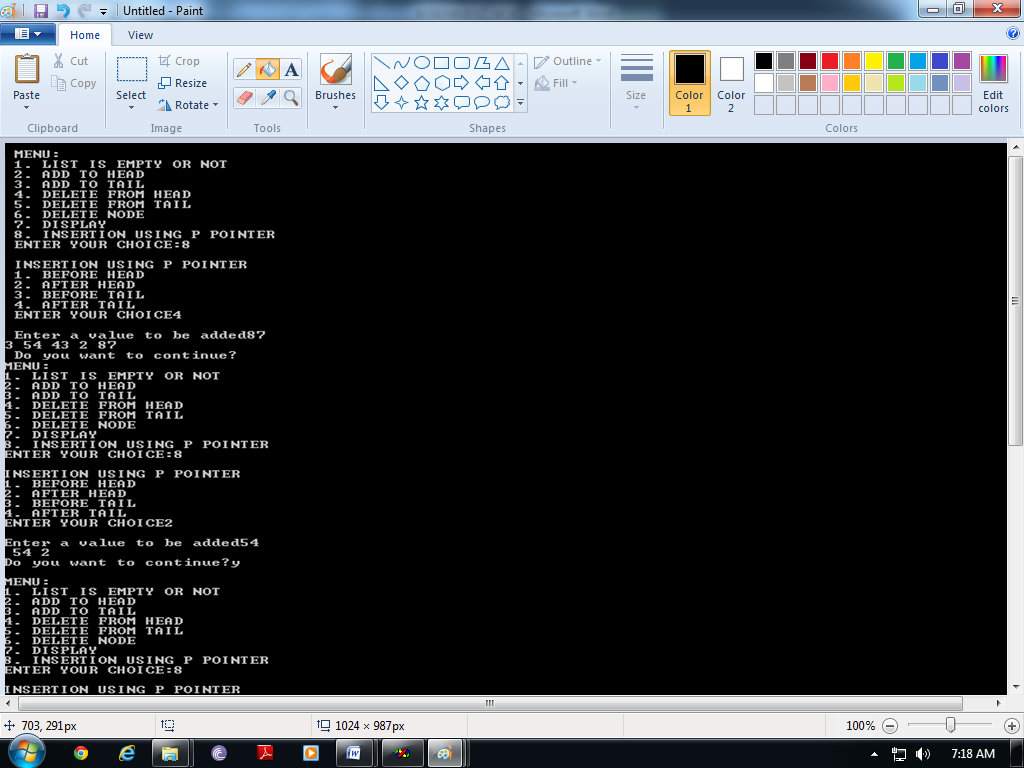
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**



**Question 14: . Implement SLL using head pointer only.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next;

node(T x, node \*n=0)

{

info=x;

next=n;

}

};

template<class T>

class SLList

{

node<T> \*head,\*tail;

public:

SLList()

{

head=tail=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

void display();

void insert\_p(node<T>\*p,int cas, T x);

node<T>\* ret\_hd(){return head;}

node<T>\* ret\_tl(){return tail;}

};

template<class T>

int SLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void SLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

temp->next=head;

head=temp;

}

}

template<class T>

void SLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=tail=temp;

}

else

{

tail->next=temp;

tail=temp;

}

}

template<class T>

T SLList<T>::deletefromhead()

{

node<T> \*temp;

T x=head->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

head=head->next;

delete temp;

}

return x;

}

template <class T>

T SLList<T>::deletefromtail()

{

node<T> \*temp;

T x=tail->info;

if(head==tail)

{

delete head;

head=tail=NULL;

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

delete tail;

tail=temp;

temp->next=NULL;

}

return x;

}

template <class T>

void SLList<T>::deletenode(T x)

{

if(head==tail && head->info==x)

{

delete head;

head=tail=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*prev,\*cur;

prev=head;

cur=head->next;

while(cur!=0 && cur->info!=x)

{

prev=cur;

cur=cur->next;

}

if(cur!=0)

{

prev->next=cur->next;

if(tail==cur)

{

tail=prev;

}

delete cur;

}

else

{

cout<<"\n Element not found";

}

}

}

template<class T>

void SLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

template<class T>

void SLList<T>::insert\_p(node<T>\*p,int cas, T x)

{

node<T>\* temp=new node<T>(x);

node<T>\*temp1=head;

switch(cas)

{

case 1://before p=head

temp->next=p;

head=temp;

break;

case 2://after p=head

temp->next=p->next;

p->next=temp;

break;

case 3://before p=tail

while(temp1->next!=p)

temp1=temp1->next;

temp1->next=temp;

temp->next=p;

break;

case 4://after p=tail

p->next=temp;

tail=temp;

break;

}

if(tail==NULL)

tail=head;

if(head==NULL)

head=tail;

}

void main()

{

SLList<int> L, L2, LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. DISPLAY";

cout<<"\n 8. INSERTION USING P POINTER";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

case 8: cout<<"\n INSERTION USING P POINTER ";

cout<<"\n 1. BEFORE HEAD";

cout<<"\n 2. AFTER HEAD";

cout<<"\n 3. BEFORE TAIL";

cout<<"\n 4. AFTER TAIL";

cout<<"\n ENTER YOUR CHOICE";

cin>>ch;

cout<<"\n Enter a value to be added";

cin>>x;

if(ch<3)

L.insert\_p(L.ret\_hd(), ch, x);

else

L.insert\_p(L.ret\_tl(), ch, x);

L.display();

break;

default: cout<<"\n Wrong Choice";

}

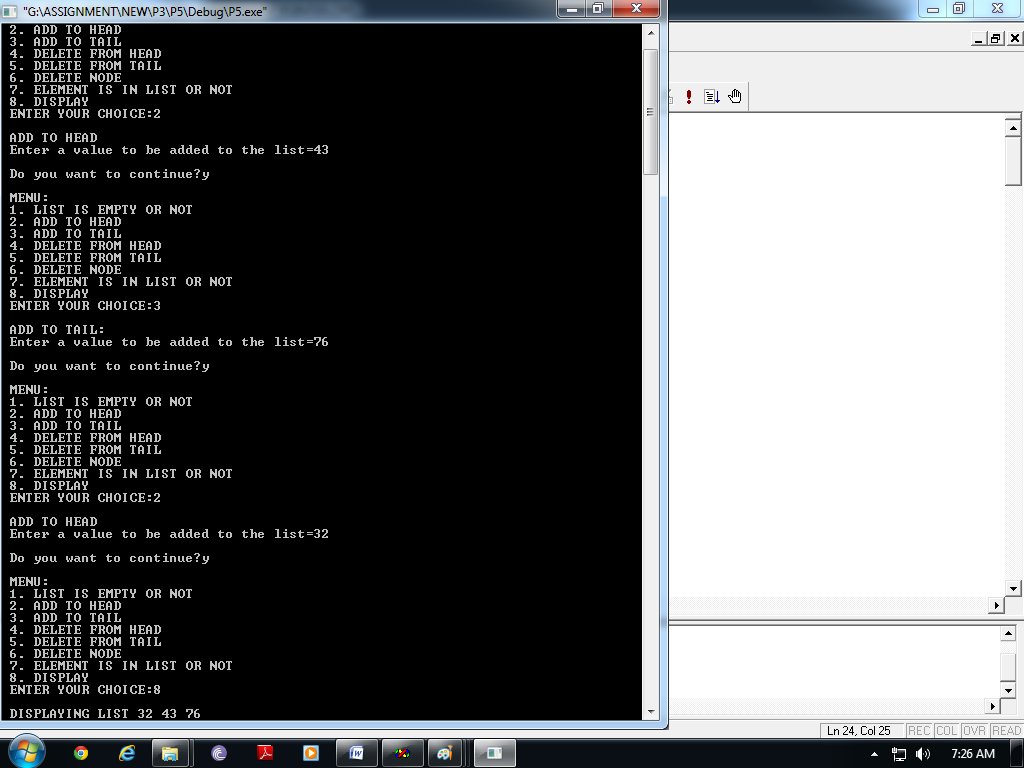
cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

****

**Question 15: Implement DLL using head pointer only.**

***Solution:***

***#include<iostream>***

***using namespace std;***

template <class T>

class node

{

public:

T info;

node \*next,\*prev;

node(T x, node \*n=0, node \*p=0)

{

info=x;

next=n;

prev=p;

}

};

template<class T>

class DLList

{

node<T> \*head;

public:

DLList()

{

head=NULL;

}

int isempty();

void addtohead(T);

void addtotail(T);

T deletefromhead();

T deletefromtail();

void deletenode(T);

int isinlist(T);

void display();

};

template<class T>

int DLList<T>::isempty()

{

if(head==NULL)

return 1;

else

return 0;

}

template<class T>

void DLList<T>::addtohead(T x)

{

node<T> \*temp=new node<T>(x,head);

if(head!=NULL)

head->prev=temp;

head=temp;

}

template<class T>

void DLList<T>::addtotail(T x)

{

node<T> \*temp=new node<T>(x);

if(isempty())

{

head=temp;

}

else

{

node<T>\* temp1;

temp1=head;

while(temp1->next!=NULL)

temp1=temp1->next;

temp1->next=temp;

temp->prev=temp1;

}

}

template<class T>

T DLList<T>::deletefromhead()

{

T x=head->info;

if(head->next==NULL)

{

delete head;

head=NULL;

}

else

{

head=head->next;

delete(head->prev);

head->prev=NULL;

}

return x;

}

template <class T>

T DLList<T>::deletefromtail()

{

T x;

node<T>\* temp;

if(head->next==NULL)

{

delete head;

head=NULL;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

x=temp->info;

temp->prev->next=NULL;

delete temp;

}

return x;

}

template <class T>

void DLList<T>::deletenode(T x)

{

if(head->next!=NULL && head->info==x)

{

delete head;

head=NULL;

}

else if(head->info==x)

{

x=deletefromhead();

cout<<"\n "<<x <<" is deleted";

}

else

{

node<T> \*temp;

temp=head->next;

while(temp!=0 && temp->info!=x)

{

temp=temp->next;

}

if(temp!=0)

{

temp->prev->next=temp->next;

if(temp->next!=NULL)

{

temp->next->prev=temp->prev;

}

delete temp;

}

else

cout<<"\n Element Not FOUND";

}

}

template<class T>

void DLList<T>::display()

{

node<T> \*temp;

for(temp=head;temp!=NULL;temp=temp->next)

{

cout<<temp->info<<" ";

}

};

void main()

{

DLList<int> L,L2,LREV;

int ch,x;

char op;

do

{

cout<<"\n MENU:";

cout<<"\n 1. LIST IS EMPTY OR NOT ";

cout<<"\n 2. ADD TO HEAD";

cout<<"\n 3. ADD TO TAIL";

cout<<"\n 4. DELETE FROM HEAD";

cout<<"\n 5. DELETE FROM TAIL";

cout<<"\n 6. DELETE NODE";

cout<<"\n 7. DISPLAY";

cout<<"\n ENTER YOUR CHOICE:";

cin>>ch;

switch(ch)

{

case 1: cout<<"\n IS LIST EMPTY OR NOT";

x=L.isempty();

if(x==1)

cout<<"\n List is empty ";

else

cout<<"\n List is not empty";

break;

case 2: cout<<"\n ADD TO HEAD";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtohead(x);

break;

case 3: cout<<"\n ADD TO TAIL:";

cout<<"\n Enter a value to be added to the list=";

cin>>x;

L.addtotail(x);

break;

case 4: cout<<"\n DELETE FROM HEAD";

if(L.isempty()==0)

{

x=L.deletefromhead();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 5:cout<<"\n DELETE FROM TAIL";

if(L.isempty()==0)

{

x=L.deletefromtail();

cout<<"\n Delete element is : "<<x;

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 6:cout<<"\n DELETE A PARTICULAR NODE";

if(L.isempty()==0)

{

cout<<"\n Enter a value you want to delete from list:";

cin>>x;

L.deletenode(x);

}

else

cout<<"\n UNDERFLOW!!!!!";

break;

case 7: cout<<"\n DISPLAYING LIST ";

if(L.isempty()==0)

L.display();

else

cout<<"\n No element to display";

break;

default: cout<<"\n Wrong Choice";

}

cout<<"\n Do you want to continue?";

cin>>op;

}while(op=='Y' || op=='y');

}

**Output:**

