**WEEK 1 LAB A**

**9923103023 - F1**

**Q1. You are given an empty singly linked list. Assume that this list can contain**

**whole numbers only. Write functions to**

**a) a. Insert ‘n’ number of data in the singly linked list. Insert from the head.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

int len=0;

public:

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

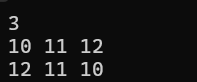
l->inath(a);

}

l->print();

return 0;

}}



**b) b. Find the total number of nodes in the linked list, and give their average.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inath(a);

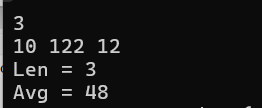
}

cout<<"Len = "<<l->len<<endl;

cout<<"Avg = "<<l->avg();

return 0;

}}



**c. Print first ‘m’ data from the linked list. Assume that ‘m’ is less than ‘n’.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

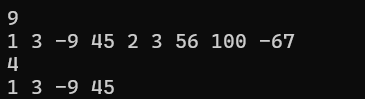
int m;

cin>>m;

l->print(m);

return 0;

}}



**d. Find the middle element of the linked list and check if it’s odd or even. Print**

**an appropriate output.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

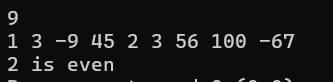
l->inatT(a, tail);

}

l->mid();

return 0;

}}



**e. Find the ‘l’ number from the end of the list.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

void lno(int m){

int n=len-m;

node<T> \* temp=head;

for(int i=0; i<n; i++){

temp=temp->next;

}

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

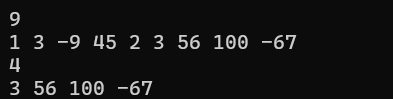
int m;

cin>>m;

l->lno(m);

return 0;

}}



**f. Find if a given number exists in the list. If it does, write function to delete it.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

cout<<endl;

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

void lno(int m){

int n=len-m;

node<T> \* temp=head;

for(int i=0; i<n; i++){

temp=temp->next;

}

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void delnum(T num){

node<T> \* temp=head;

node<T> \* temp2=NULL;

while(temp!=NULL){

if(temp->data==num) break;

temp2=temp;

temp=temp->next;

}

if(temp==NULL){

cout<<"number not found"<<endl;

return;

}

temp2->next=temp->next;

delete temp;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

cout<<"Before deletion"<<endl;

l->print();

int m;

cin>>m;

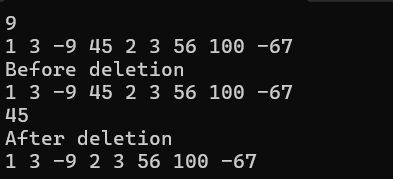
l->delnum(m);

cout<<"After deletion"<<endl;

l->print();

return 0;

}}



**g. Interchange a pair of values with another given pair in the linked list.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

cout<<endl;

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

void lno(int m){

int n=len-m;

node<T> \* temp=head;

for(int i=0; i<n; i++){

temp=temp->next;

}

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void delnum(T num){

node<T> \* temp=head;

node<T> \* temp2=NULL;

while(temp!=NULL){

if(temp->data==num) break;

temp2=temp;

temp=temp->next;

}

if(temp==NULL){

cout<<"number not found"<<endl;

return;

}

temp2->next=temp->next;

delete temp;

}

void interchange(T a, T b, T c, T d){

node<T> \* temp= head;

bool t=false;

while(temp!=NULL){

if(temp->data==a && temp->next->data==b){

t=true;

break;

}

temp=temp->next;

}

if(!t) return;

t=false;

node<T> \*temp2=head;

while(temp2!=NULL){

if(temp2->data==c && temp2->next->data==d){

t=true;

break;

}

temp2=temp2->next;

}

if(!t) return;

temp->data=c;

temp->next->data=d;

temp2->data=a;

temp2->next->data=b;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

cout<<"Enter pair to be swapped : ";

int a,b;

cin>>a>>b;

cout<<"Enter second values : ";

int c,d;

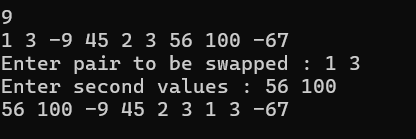
cin>>c>>d;

l->interchange(a,b,c,d);

l->print();

return 0;

}}



**h. Check whether a given sub-list exists in the given linked list. If it exists, give**

**its position (i.e., the staring position of the sub-list in the master linked list).**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

cout<<endl;

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node<int>(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

void lno(int m){

int n=len-m;

node<T> \* temp=head;

for(int i=0; i<n; i++){

temp=temp->next;

}

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void delnum(T num){

node<T> \* temp=head;

node<T> \* temp2=NULL;

while(temp!=NULL){

if(temp->data==num) break;

temp2=temp;

temp=temp->next;

}

if(temp==NULL){

cout<<"number not found"<<endl;

return;

}

temp2->next=temp->next;

delete temp;

}

void interchange(T a, T b, T c, T d){

node<T> \* temp= head;

bool t=false;

while(temp!=NULL){

if(temp->data==a && temp->next->data==b){

t=true;

break;

}

temp=temp->next;

}

if(!t) return;

t=false;

node<T> \*temp2=head;

while(temp2!=NULL){

if(temp2->data==c && temp2->next->data==d){

t=true;

break;

}

temp2=temp2->next;

}

if(!t) return;

temp->data=c;

temp->next->data=d;

temp2->data=a;

temp2->next->data=b;

}

int sub(LL <T>\* & l){

node<T> \* temp=head;

node<T> \* temp2=l->head;

bool t=true;

int cnt=1;

int i=1;

while(temp!=NULL){

cnt++;

if(temp->data==temp2->data){

if(t) {

i=cnt;

t=false;

}

temp2=temp2->next;

}

else{

temp2=l->head;

t=true;

}

if(temp2==NULL){

return i-1;

}

temp=temp->next;

}

return -1;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

cin>>n;

if(n>0){

cin>>a;

node<int> \* head2=new node<int>(a);

node<int> \* tail2=head2;

LL<int>\* l2= new LL<int>(head2);

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

int a;

cin>>a;

l2->inatT(a, tail2);

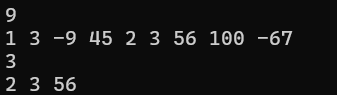
}

cout<<"present at "<<l->sub(l2);

return 0;

}}

}



**i. Reverse a sub-list in the given linked list.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

public:

int len=0;

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

cout<<endl;

}

void inatT(T d, node<T> \*& tail){

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i, node<T>\* & tail){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

float avg(){

float a=0;

node<T> \* temp=head;

while(temp!=NULL){

a+=temp->data;

temp=temp->next;

}

return a/len;

}

void print(int n){

if(n>len){

cout<<"Invalid m"<<endl;

return;

}

node<T> \* temp=head;

for(int i=0; i<n ; i++){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void mid(){

int m=len/2;

node<T> \* temp=head;

for(int i=0; i<m; i++){

temp=temp->next;

}

cout<<temp->data<<" is ";

if(temp->data%2) cout<<"odd";

else cout<<"even";

}

void lno(int m){

int n=len-m;

node<T> \* temp=head;

for(int i=0; i<n; i++){

temp=temp->next;

}

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void delnum(T num){

node<T> \* temp=head;

node<T> \* temp2=NULL;

while(temp!=NULL){

if(temp->data==num) break;

temp2=temp;

temp=temp->next;

}

if(temp==NULL){

cout<<"number not found"<<endl;

return;

}

temp2->next=temp->next;

delete temp;

}

void rev(LL \*& l){

int lenn=l->len;

node<T> \* temp2=head;

node<T>\* temp=NULL;

bool t=true;

while(temp2->data!=l->head->data){

temp=temp2;

temp2=temp2->next;

t=false;

}

node<T> \* temp4=temp2;

for(int i=0; i<lenn; i++){

temp4=temp4->next;

}

{

node<T> \*tempp=NULL;

node<T> \*tempp2=l->head;

node<T> \* tempp3=NULL;

for(int i=0; i<l->len; i++){

tempp3=tempp2->next;

tempp2->next=tempp;

tempp=tempp2;

tempp2=tempp3;

}

l->head=tempp;

}

if(!t)temp->next=l->head;

else{

node<T> \* temp=l->head;

while(temp->next!=NULL){

temp=temp->next;

}

temp->next=temp4;

this->head=l->head;

}

for(int i=0; i<lenn; i++){

temp=temp->next;

}

temp->next=temp4;

}

};

int main(){

int n;

cin>>n;

int a;

if(n>0){ cin>>a;

node<int> \* head=new node<int>(a);

node<int> \*tail=head;

LL<int> \* l=new LL<int>(head);

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

int a;

cin>>a;

l->inatT(a, tail);

}

int m;

cin>>m;

if(m>0){

int a;

cin>>a;

node<int> \* head2= new node<int>(a);

node<int> \* tail2= head2;

LL<int> \* ll=new LL<int>(head2);

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

cin>>a;

ll->inatT(a, tail2);

}

cout<<"Before "<<endl;

l->print();

l->rev(ll);

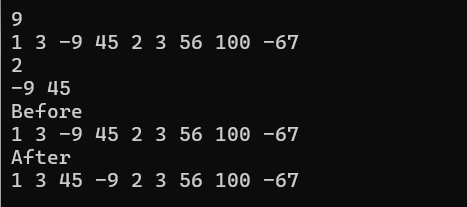
}

cout<<"After"<<endl;

l->print();

return 0;

}}



**Q2. Assume that you have a linked list that can contain strings, i.e., each node can**

**contain a string. Write a function to:**

Given below is the implementation of linked list

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

node<T> \* tail=NULL;

int len=0;

public:

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

if(head==NULL){

head=temp;

tail=head;

return;

}

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d){

if(tail==NULL){

inath(d);

return;

}

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

};

int main(){

return 0;

}

Only functions and main function will be typed in question parts as implementation for all sub parts

is same

**a. Print all the nodes in the linked list**

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

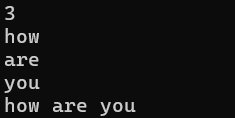
s->inatT(st);

}

s->print();

return 0;

}



b. **Print all the strings (node values) that start with a particular alphabet.**

template<class T>

void LL<T>:: print(char a){

node<T> \* temp=head;

while(temp!=NULL){

if(temp->data[0]==a) cout<<temp->data<<' ';

temp=temp->next;

}

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

s->inatT(st);

}

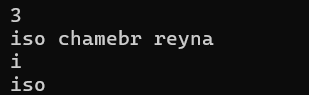
char a;

cin>>a;

s->print(a);

return 0;

}



**c. Find if a given string exists in the linked list or not. Give appropriate output**

**message.**

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

node<T> \* tail=NULL;

int len=0;

public:

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

if(head==NULL){

head=temp;

tail=head;

return;

}

temp->next=head;

this->head=temp;

len++;

}

void print(){

node<T> \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void inatT(T d){

if(tail==NULL){

inath(d);

return;

}

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

tail->next=temp;

tail=temp;

len++;

}

void inati(T d, int i){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

void print(char a);

void exist(T a);

};

template<class T>

void LL<T>:: print(char a){

node<T> \* temp=head;

while(temp!=NULL){

if(temp->data[0]==a) cout<<temp->data<<' ';

temp=temp->next;

}

}

template<class T>

void LL<T> :: exist(T a){

node<T> \* temp=head;

while(temp!=NULL){

if(temp->data==a){

cout<<"String exists";

return;

}

temp=temp->next;

}

cout<<"string does not exist";

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

s->inatT(st);

}

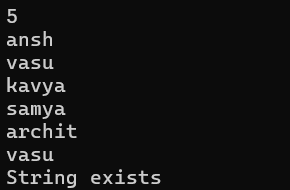
string st;

cin>>st;

s->exist(st);

return 0;

}



**d. Find the string with maximum length.**

template<class T>

void LL<T> :: maxx(){

node<T> \* temp=head;

string s=temp->data;;

int len=temp->data.length();

while(temp!=NULL){

if(temp->data.length()>len){

len=temp->data.length();

s=temp->data;

}

temp=temp->next;

}

cout<<"max length : "<<s;

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

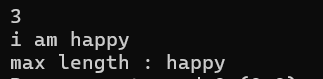
s->inatT(st);

}

s->maxx();

return 0;

}



**e. Find if a node contains “xyz” as a sub-string or not. Give appropriate output**

**message.**

template<class T>

int LL<T> :: sub(string s){

node<T> \* temp=head;

while(temp!=NULL){

string dataa=temp->data;

if(dataa.find(s)!= string::npos){

cout<<"Substring exists in : "<<dataa;

return 1;

}

}

cout<<"Substring does not exist";

return 0;

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

s->inatT(st);

}

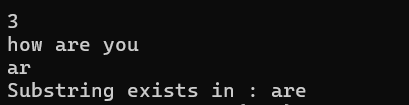
string st;

cin>>st;

s->sub(st);

return 0;

}



**f. Interchange the strings given in the positions p1, p2. These positions are**

**user input. Check conditions that both p1 and p2 position exist in the given**

**linked list, eg: suppose that your linked list consists of 4 strings only, and if**

**user given p1=7, p2 = 10, then error message must be generated.**

template<class T>

void LL<T> :: exchange(int a, int b){

if(a>this->len || b>this->len || a<0 || b<0){

cout<<"Index out of range";

return;

}

node<T> \* temp=head;

string o,t;

int n;

if(a>b) n=a;

else n=b;

for(int i=0; i<n+1; i++){

if(i==a){

o=temp->data;

}

if(i==b){

t=temp->data;

}

temp=temp->next;

}

temp=head;

for(int i=0; i<n+1; i++){

if(i==a){

temp->data=t;

}

if(i==b){

temp->data=o;

}

temp=temp->next;

}

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

s->inatT(st);

}

cout<<"Before "<<endl;

s->print();

int a,b;

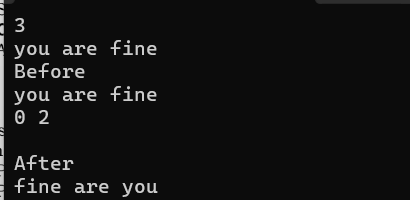
cin>>a>>b;

s->exchange(a,b);

cout<<endl<<"After"<<endl;

s->print();

}



**g. Delete a given node (either by value or by position).**

template<class T>

void LL<T>:: deletee(int n){

node<T> \* temp=head;

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

temp=temp->next;

}

node<T> \* temp2=temp->next->next;

delete temp->next;

temp->next=temp2;

}

int main(){

LL<string> \* s=new LL<string>();

int n;

cin>>n;

for(int i=0; i<n; i++){

string st;

cin>>st;

s->inatT(st);

}

cout<<"Before "<<endl;

s->print();

int a;

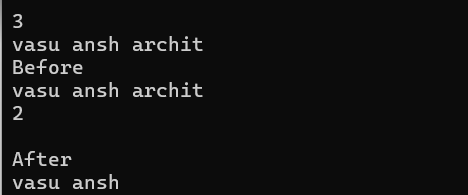
cin>>a;

s->deletee(a);

cout<<endl<<"After"<<endl;

s->print();

}



**Q3. Implement a circular linked list that can contain integer elements. Add**

**functions to:**

Given below

is the implementation of CLL that will remain same for all sub parts

#include<iostream>

using namespace std;

template <class T>

class LL;

template <class T>

class node{

bool ishead=false;

T data;

node<T> \* next=NULL;

public:

node<T>(){}

node<T>(T a) {data=a;}

friend class LL<T>;

};

template <class T>

class LL{

node<T> \* head=NULL;

node<T> \* tail=NULL;

int len=0;

public:

LL(){}

LL(node<T> \* &h){

head=h;

len=1;

}

void inath(T d){

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

temp->ishead=true;

if(head==NULL){

head=temp;

tail=head;

len++;

return;

}

head->ishead=false;

temp->next=head;

this->head=temp;

tail->next=head;

len++;

}

void print(){

node<T> \* temp=head;

while(!temp->next->ishead){

cout<<temp->data<<' ';

temp=temp->next;

}

cout<<temp->data;

}

void inatT(T d){

if(tail==NULL){

inath(d);

return;

}

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

tail->next=temp;

tail=temp;

tail->next=head;

len++;

}

void inati(T d, int i){

if(i>len || i<0) {

cout<<"Invalid index"<<endl;

return;

}

if(i==0) {inath(d);

return ;

}

else if(i==len){

inatT(d,tail);

return ;

}

node<T>\* temp=head;

node<T>\* temp2=new node(d);

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

temp=temp->next;

}

temp2->next=temp->next;

temp->next=temp2;

}

};

**a. Insert elements.**

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

}

return 0;

}



**b. Print elements**

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

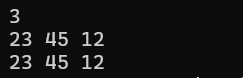
s->inatT(st);

}

s->print();

return 0;

}



c. **Count the number of elements**

template<class T>

void LL<T> :: length(){

cout<<len;

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

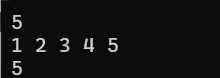
s->inatT(st);

}

s->length();

return 0;

}



d. **Find if any element has a negative value.**

template<class T>

void LL<T> :: neg(){

node<T> \* temp=head;

while(!temp->next->ishead){

if(temp->data<0){

cout<<"Negative present";

return;

}

temp=temp->next;

}

cout<<"Negative not present";

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

}

s->neg();

return 0;

}



**e. Find the number of nodes having a value greater than 15.**

template<class T>

void LL<T> :: fif(){

node<T> \* temp=head;

int cnt=0;

do{

if(temp->data>15){

cnt++;

}

temp=temp->next;

}while(!temp->ishead);

cout<<"Greater than 15 : "<<cnt;

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

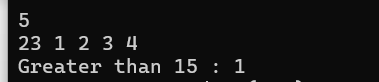
s->inatT(st);

}

s->fif();

return 0;

}



**f. Delete a particular element from the list.**

template<class T>

void LL<T>:: deletee(T n){

node<T> \* temp=head;

do{

if(temp->next->data==n) break;

temp=temp->next;

}while(!temp->ishead);

node<T> \* temp2=temp->next;

temp->next=temp2->next;

delete temp2;

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

}

cin>>n;

s->deletee(n);

s->print();

return 0;

}



**g. Update the value of a particular element.**

template<class T>

void LL<T>:: updatee(T n,T m){

node<T> \* temp=head;

do{

if(temp->data==n) {

temp->data=m;

break;}

temp=temp->next;

}while(!temp->ishead);

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

}

cin>>n;

int m;

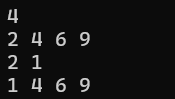
cin>>m;

s->updatee(n, m);

s->print();

return 0;

}



h. **Insert a value at a given position.**

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

}

cin>>n;

int m;

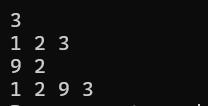
cin>>m;

s->inati(n, m);

s->print();

return 0;

}



**i. Delete all nodes that have a prime number as their value.**

template<class T>

void LL<T>:: deletee(T n){

node<T> \* temp=head;

do{

if(temp->next->data==n) break;

temp=temp->next;

}while(!temp->ishead);

node<T> \* temp2=temp->next;

temp->next=temp2->next;

if(temp2->ishead) temp2->next->ishead=true;

delete temp2;

len--;

}

bool isprime(int n){

if(!n%2 && n!=2) return false;

if(n==1) return false;

for(int i=3; i<=pow(n, 0.5); i+=2){

if(n%i==0) return false;

}

return true;

}

template<class T>

void LL<T> :: delprm(){

node<T>\* temp=head;

do{

node<T> \* temp2=temp->next;

if(isprime(temp->data)){

this->deletee(temp->data);

}

temp=temp2;

}while(!temp->ishead);

}

int main(){

LL<int> \* s=new LL<int>();

int n;

cin>>n;

for(int i=0; i<n; i++){

int st;

cin>>st;

s->inatT(st);

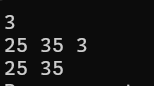
}

s->delprm();

s->print();

return 0;

}



**j. Remove all the nodes from the list which contains Fibonacci data values.**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

template<class T>

struct node {

T data;

node<T>\* next;

bool ishead;

node(T d) : data(d), next(nullptr), ishead(false) {}

};

template<class T>

class LL {

public:

node<T>\* head;

int len;

LL() : head(nullptr), len(0) {}

void inatT(T n);

void print();

void deletee(T n);

void delFibonacci();

};

template<class T>

void LL<T>::inatT(T n) {

node<T>\* newNode = new node<T>(n);

if (!head) {

head = newNode;

head->next = head;

head->ishead = true;

} else {

node<T>\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

len++;

}

template<class T>

void LL<T>::print() {

if (!head) return;

node<T>\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

cout << endl;

}

template<class T>

void LL<T>::deletee(T n) {

if (!head) return;

node<T>\* current = head;

node<T>\* prev = nullptr;

if (head->data == n) {

if (head->next == head) {

delete head;

head = nullptr;

} else {

node<T>\* last = head;

while (last->next != head) {

last = last->next;

}

node<T>\* toDelete = head;

head = head->next;

last->next = head;

delete toDelete;

}

len--;

return;

}

prev = head;

current = head->next;

while (current != head) {

if (current->data == n) {

prev->next = current->next;

if (current->next == head) {

head->ishead = true;

}

delete current;

len--;

return;

}

prev = current;

current = current->next;

}

}

bool isFibonacci(int n) {

if (n < 0) return false;

int x1 = 5 \* n \* n + 4;

int x2 = 5 \* n \* n - 4;

int a = static\_cast<int>(sqrt(x1));

int b = static\_cast<int>(sqrt(x2));

return (x1 == (a \* a)) || (x2 == (b \* b));

}

template<class T>

void LL<T>::delFibonacci() {

if (!head) return;

vector<node<T>\*> nodesToDelete;

node<T>\* temp = head;

do {

if (isFibonacci(temp->data)) {

nodesToDelete.push\_back(temp);

}

temp = temp->next;

} while (temp != head);

for (node<T>\* nodeToDelete : nodesToDelete) {

this->deletee(nodeToDelete->data);

}

}

int main() {

LL<int>\* s = new LL<int>();

int n;

cin >> n;

for (int i = 0; i < n; i++) {

int st;

cin >> st;

s->inatT(st);

}

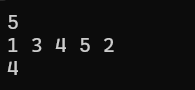
s->delFibonacci();

s->print();

delete s;

return 0;

}



**Q4. Create an empty doubly linked list to store integers. Perform the following by**

**writing appropriate functions to:**

**a. Insert and print elements of the list.**

#include <iostream>

using namespace std;

class DLL;

class node{

int data;

node \* next=NULL;

node \* prev=NULL;

public:

node(){}

node(int a){data=a;}

friend class DLL;

};

class DLL{

node \* head=NULL;

node\* tail=NULL;

public:

DLL(){}

DLL(node \* &a) {head=a;}

void inatT(int t){

if(!head){

head=new node(t);

tail=head;

return;

}

node \* temp= new node(t);

tail->next=temp;

temp->prev=tail;

tail=temp;

return;

}

void print(){

node \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

};

int main() {

DLL \* dll=new DLL();

int n;

cin>>n;

for(int i=0; i<n; i++){

int a;

cin>>a;

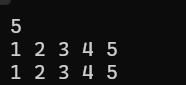
dll->inatT(a);

}

dll->print();

return 0;

}



b. **Traverse all nodes and check if the value is divisible by a number ‘m’.**

#include <iostream>

using namespace std;

class DLL;

class node{

int data;

node \* next=NULL;

node \* prev=NULL;

public:

node(){}

node(int a){data=a;}

friend class DLL;

};

class DLL{

node \* head=NULL;

node\* tail=NULL;

int len=0;

public:

DLL(){}

DLL(node \* &a) {head=a;

len++;

}

void inatT(int t){

len++;

if(!head){

head=new node(t);

tail=head;

return;

}

node \* temp= new node(t);

tail->next=temp;

temp->prev=tail;

tail=temp;

return;

}

void print(){

node \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void divi(int m){

node\* t1=head;

node\* t2=tail;

bool b=true;

while(b){

if(!(t1->data%m)){

cout<<t1->data<<" divisible by "<<m<<endl;

}

if(!(t2->data%m)){

cout<<t2->data<<" divisible by "<<m<<endl;

}

t1=t1->next;

t2=t2->prev;

if(len%2){

b=(t2->next->next!=t1);

}

else b=(t2->next!=t1);

}

if(!t1->data%m){

cout<<t1->data<<" divisible by "<<m<<endl;

}

}};

int main() {

DLL \* dll=new DLL();

int n;

cin>>n;

for(int i=0; i<n; i++){

int a;

cin>>a;

dll->inatT(a);

}

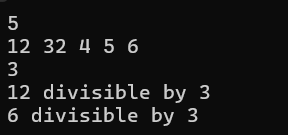
int m;

cin>>m;

dll->divi(m);

return 0;

}



**c. Delete all the nodes from the list that are greater than the given value ‘x’.**

#include <iostream>

#include<vector>

using namespace std;

class DLL;

class node{

int data;

node \* next=NULL;

node \* prev=NULL;

public:

node(){}

node(int a){data=a;}

friend class DLL;

};

class DLL{

node \* head=NULL;

node\* tail=NULL;

int len=0;

public:

DLL(){}

DLL(node \* &a) {head=a;

len++;

}

void inatT(int t){

len++;

if(!head){

head=new node(t);

tail=head;

return;

}

node \* temp= new node(t);

tail->next=temp;

temp->prev=tail;

tail=temp;

return;

}

void print(){

node \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void dele(int x){

node\* t1=head;

node\* t2=tail;

vector<node\*> v;

bool b=true;

while(b){

if((t1->data>x)){

v.push\_back(t1);

}

if(t2->data>x && t1!=t2){

v.push\_back(t2);

}

t1=t1->next;

t2=t2->prev;

if(len%2){

b=(t2->next->next!=t1);

}

else b=(t2->next!=t1);

}

for(int i=0; i<v.size(); i++){

if(v[i]==head){

head=head->next;

head->prev=NULL;

}

else if(v[i]==tail){

tail=tail->prev;

tail->next=NULL;

}

else{

v[i]->prev->next=v[i]->next;

v[i]->next->prev=v[i]->prev;

}

delete v[i];

}

}

};

int main() {

DLL \* dll=new DLL();

int n;

cin>>n;

for(int i=0; i<n; i++){

int a;

cin>>a;

dll->inatT(a);

}

int m;

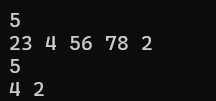
cin>>m;

dll->dele(m);

dll->print();

return 0;

}



d. **Find the number of elements between two duplicate values.**

#include <iostream>

#include<vector>

using namespace std;

class DLL;

class node{

int data;

node \* next=NULL;

node \* prev=NULL;

public:

node(){}

node(int a){data=a;}

friend class DLL;

};

class DLL{

node \* head=NULL;

node\* tail=NULL;

int len=0;

public:

DLL(){}

DLL(node \* &a) {head=a;

len++;

}

void inatT(int t){

len++;

if(!head){

head=new node(t);

tail=head;

return;

}

node \* temp= new node(t);

tail->next=temp;

temp->prev=tail;

tail=temp;

return;

}

void print(){

node \* temp=head;

while(temp!=NULL){

cout<<temp->data<<' ';

temp=temp->next;

}

}

void dele(int x){

node\* t1=head;

node\* t2=tail;

vector<node\*> v;

bool b=true;

while(b){

if((t1->data>x)){

v.push\_back(t1);

}

if(t2->data>x && t1!=t2){

v.push\_back(t2);

}

t1=t1->next;

t2=t2->prev;

if(len%2){

b=(t2->next->next!=t1);

}

else b=(t2->next!=t1);

}

for(int i=0; i<v.size(); i++){

if(v[i]==head){

head=head->next;

head->prev=NULL;

}

else if(v[i]==tail){

tail=tail->prev;

tail->next=NULL;

}

else{

v[i]->prev->next=v[i]->next;

v[i]->next->prev=v[i]->prev;

}

delete v[i];

}

}

void dupli(int n){

node \* temp=head;

int cnt=0;

bool t =false;

while(temp!=NULL){

if(t && temp->data==n) break;

if(temp->data==n){

t=true;

}

if(t) cnt++;

temp=temp->next;

}

cout<<cnt-1;

}

};

int main() {

DLL \* dll=new DLL();

int n;

cin>>n;

for(int i=0; i<n; i++){

int a;

cin>>a;

dll->inatT(a);

}

int m;

cin>>m;

dll->dupli(m);

return 0;

}

