#include <iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

class linked\_list

{

private:

node \*head,\*tail;

int size;

public:

linked\_list()

{

head = NULL;

tail = NULL;

size = 0;

}

void addlast(int n)

{

node \*tmp = new node;

tmp->data = n;

tmp->next = NULL;

if(head == NULL)

{

head = tmp;

tail = tmp;

}

else

{

tail->next = tmp;

tail = tail->next;

}

size++;

}

void addfirst(int value){

node \*temp = new node;

temp->data = value;

if(head==NULL){

temp->next = NULL;

tail = temp;

head = temp;

}

else{

temp->next = head;

head = temp;

}

size++;

}

node\* gethead()

{

return head;

}

node\* gettail()

{

return tail;

}

int size\_(){

return size;

}

static node\* mid\_node(node\* head, node\* tail){

node\* slow = head;

node\* fast = head;

while(fast!=tail && fast->next!=tail){

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

static void display(node \*head)

{

node\* temp = head;

while(temp!=NULL){

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

temp = temp->next;

}

cout<<endl;

}

static linked\_list mergeTwolist(node \*a,node \*b)

{

linked\_list c;

node\* i = a;

node\* j = b;

while(i!=NULL && j!=NULL){

if(i->data<=j->data){

c.addlast(i->data);

i = i->next;

}

else{

c.addlast(j->data);

j = j->next;

}

}

while(i!=NULL){

c.addlast(i->data);

i = i->next;

}

while(j!=NULL){

c.addlast(j->data);

j = j->next;

}

return c;

}

static linked\_list merge\_sort(node\* head, node\* tail){

if(head==tail){

linked\_list base;

base.addlast(head->data);

// cout<<head->data<<endl;

return base;

}

node\* mid = linked\_list::mid\_node(head,tail);

linked\_list s1 = linked\_list::merge\_sort(head,mid);

linked\_list s2 = linked\_list::merge\_sort(mid->next, tail);

linked\_list res = linked\_list::mergeTwolist(s1.gethead(),s2.gethead());

return res;

}

static linked\_list remove\_duplicate(node\* head){

linked\_list res;

node\* i = head;

node\* j = i->next;

res.addlast(head->data);

while(j!=NULL){

if(i->data==j->data){

j = j->next;

}

else{

res.addlast(j->data);

i = j;

j = j->next;

}

}

return res;

}

static linked\_list oddeven(node \*head){

node\* temp = head;

linked\_list odd;

linked\_list even;

while(temp!=NULL){

if((temp->data%2)==0){

even.addlast(temp->data);

}

else{

odd.addlast(temp->data);

}

temp = temp->next;

}

int o = odd.size\_();

int e = even.size\_();

if(o>0 && e>0){

node\* t = odd.gettail();

node\* h = even.gethead();

t->next = h;

odd.tail = even.tail;

odd.size = o + e;

return odd;

}

else if(o>0){

return odd;

}

else{

return even;

}

}

void reverse\_display(node\* node){

if(node==NULL){

return;

}

reverse\_display(node->next);

cout<<node->data<<" ";

}

void reversePIhelper(node\* node){

if(node==NULL){

return;

}

reversePIhelper(node->next);

if(node==tail){

// return;

}

else{

node->next->next = node;

}

}

void reversePI(node\* head){

reversePIhelper(head);

head->next = NULL;

node\* temp = head;

head = tail;

tail = temp;

}

};

int main()

{

linked\_list a;

// linked\_list b;

int n,m,num1,num2,num;

cin>>n;

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

cin>>num;

a.addlast(num);

}

cin>>num1;

cin>>num2;

// node\* h = a.gethead();

linked\_list::display(a.gethead());

a.reversePI(a.gethead());

linked\_list::display(a.gethead());

// linked\_list::display(c.gethead());

a.addlast(num1);

a.addfirst(num2);

linked\_list::display(a.gethead());

// cout<<c.size\_()<<endl;

return 0;

}