#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define LEN sizeof(struct Link\_Node)

//带头结点的链表操作

typedef struct Link\_Node

{

int data;

struct Link\_Node \* next;

} node;

void output(node \*p);

void outputs(node \*pHead);

void input(node \*p);

void CreateLinkTail(node \*pHead,int n);

void CreateLinkHead(node \*pHead,int n);

void CreateSortedLink(node \*pHead,int n);

int Search(node \*pHead,int a);

int Update(node \*pHead,int old,int xin);

void Insert(node \*pHead,node \*pNew);

int Dele(node \*pHead,int a);

int length(node \*pHead);

void sort(node \*pHead);

void clear(node \*pHead);

node \* UnionLink(node \*pHead,node \*qHead);

void UnionLink2(node \*pHead,node \*qHead);

void UnionLink3(node \*pHead,node \*qHead);

void DelDua(node \*pHead);

void output(node \*p)

{

printf("%d ",p->data);

}

void outputs(node \*pHead)

{

if(pHead==NULL ) return;

node \*p=pHead->next;

while(p)

{

output(p);

p=p->next;

}

printf("\n");

}

void input(node \*p)

{

scanf("%d",&p->data);

}

void CreateLinkTail(node \*pHead,int n)

{

if(pHead==NULL) return;

node \*pTail=pHead,\*pNew;

int i;

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

{

pNew=(node \*)malloc(LEN);

if(pNew==NULL) exit(1);

input(pNew);

pNew->next=NULL;

pTail->next=pNew;

pTail=pNew;

}

}

void CreateLinkHead(node \*pHead,int n)

{

if(pHead==NULL) return;

node \*pNew;

int i;

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

{

pNew=(node \*)malloc(LEN);

if(pNew==NULL) exit(1);

input(pNew);

pNew->next=pHead->next;

pHead->next=pNew;

}

}

void CreateSortedLink(node \*pHead,int n)

{

int i;

node \* pNew;

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

{

pNew=(node \*)malloc(LEN);

if(pNew==NULL) exit(1);

input(pNew);

Insert(pHead,pNew);

}

}

int Search(node \*pHead,int a)

{

if(pHead==NULL || pHead->next==NULL) return 0;

node \*p=pHead->next;

while(p)

{

if(p->data==a) return 1;

p=p->next;

}

return 0;

}

int Update(node \*pHead,int old,int xin)

{

int n=0;//n表示修改节点的数量

if(pHead==NULL || pHead->next==NULL) return n;

node \*p=pHead->next;

while(p)

{

if(p->data==old)

{

p->data=xin;

n++;

}

p=p->next;

}

return n;

}

void Insert(node \*pHead,node \*pNew)

{

//从大到小

//寻找插入位置，即pLast和p之间，p为第一个指向的值比pNew->data小的，

//也就是当p->data大于pNew->data时，p和pLast前进一个节点。

if(pHead==NULL) return ;

if(pHead->next==NULL) //当前链表为空时

{

pHead->next=pNew;

pNew->next=NULL;

return; //这个千万不能省，否则执行后面的代码将使得最后一个节点的next是其自身。

}

node \*pLast=pHead,\*p=pHead->next;

while(p && p->data>pNew->data)

{

pLast=p;

p=p->next;

}

if(p==NULL)//pNew->data最小，存放在链表的最后

{

pLast->next=pNew;

pNew->next=NULL;

}

else

{

pNew->next=p;

pLast->next=pNew;

}

}

int Dele(node \*pHead,int a)

{

int n=0;

if(pHead==NULL || pHead->next==NULL) return n;

node \*pLast=pHead,\*p=pHead->next;

while(p)

{

if(p->data==a)

{

pLast->next=p->next;

free(p);

p=pLast->next;

n++;

}

else

{

pLast=p;

p=p->next;

}

}

return n;

}

int length(node \*pHead)

{

node \*p;

int n=0;

if(pHead==NULL || pHead->next==NULL) return n;

p=pHead->next;

while(p)

{

n++;

p=p->next;

}

return n;

}

void sort(node \*pHead)

{

//从大到小

node \*pLast,\*p;

int i,j,n=length(pHead),t;

if(n<=1) return ;

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

{

pLast=pHead->next;

p=pLast->next;

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

{

if(pLast->data<p->data)

{

t=pLast->data;

pLast->data=p->data;

p->data=t;

}

pLast=p;

p=p->next;

}

}

}

void clear(node \*pHead)

{

if(pHead==NULL) return;

node \*p;

while(pHead)

{

p=pHead->next;

free(pHead);

pHead=p;

}

}

node \* UnionLink(node \*pHead,node \*qHead)

{

//每一个都加到pHead最后

node \*uHead=NULL,\*uNew=NULL,\*uTail=NULL,\*p;

if(pHead==NULL && qHead==NULL) return uHead;

p=pHead->next;

while(p)

{

if(Search(uHead,p->data)==0)//当前链表中不存在

{

uNew=(node \*)malloc(LEN);

if(uNew==NULL) exit(1);

uNew->data=p->data;

uNew->next=NULL;

if(uHead==NULL)

{

uHead=(node \*)malloc(LEN);//创建一个空的头结点

if(uHead==NULL) exit(1);

uHead->next=uNew;

uTail=uNew;

}

else

{

uTail->next=uNew;

uTail=uNew;

}

}

p=p->next;

}

p=qHead->next;

while(p)

{

if(Search(uHead,p->data)==0)//当前链表中不存在

{

uNew=(node \*)malloc(LEN);

if(uNew==NULL) exit(1);

uNew->data=p->data;

uNew->next=NULL;

if(uHead==NULL)

{

uHead=uNew;

uTail=uNew;

}

else

{

uTail->next=uNew;

uTail=uNew;

}

}

p=p->next;

}

return uHead;

}

void UnionLink2(node \*pHead,node \*qHead)

{

//把qHead中的内容加到pHead链表的尾部

node \*p,\*pTail=NULL,\*pNew=NULL;

if(pHead==NULL || qHead==NULL) return ;

pTail=pHead->next;

while(pTail->next)

{

pTail=pTail->next;

}

p=qHead->next;

while(p)

{

if(Search(pHead,p->data)==0)

{

pNew=(node \*)malloc(LEN);

if(pNew==NULL) exit(1);

pNew->data=p->data;

pNew->next=NULL;

pTail->next=pNew;

pTail=pNew;

}

p=p->next;

}

}

void UnionLink3(node \*pHead,node \*qHead)

{

//把qHead中的内容加到pHead链表的首部

node \*p,\*pNew=NULL;

if(pHead==NULL || qHead==NULL) return ;

p=qHead->next;

while(p)

{

if(Search(pHead,p->data)==0)

{

pNew=(node \*)malloc(LEN);

if(pNew==NULL) exit(1);

pNew->data=p->data;

pNew->next=pHead->next;

pHead->next=pNew;

}

p=p->next;

}

}

void DelDua(node \*pHead)

{

//去除重复节点

if(pHead==NULL || pHead->next==NULL) return;

node \*p, \*q,\*qLast;

p=pHead->next;

while(p->next!=NULL)

{

qLast=p;

q=p->next;

while(q)

{

if(q->data==p->data)

{

qLast->next=q->next;

free(q);

q=qLast->next;

}else

{

qLast=q;

q=q->next;

}

}

p=p->next;

}

}

int main()

{

node \* pHead=NULL,\*qHead=NULL,\*uHead=NULL;

int n,i;

pHead=(node \*)malloc(LEN);

//qHead=(node \*)malloc(LEN);

if(pHead==NULL) exit(1);

//if(qHead==NULL) exit(1);

pHead->next=NULL;

//qHead->next=NULL;

scanf("%d",&n);

// CreateSortedLink(pHead,n);

// CreateLinkHead(pHead,n);

CreateLinkTail(pHead,n);

//CreateLinkTail(qHead,n);

outputs(pHead);

//outputs(qHead);

//sort(pHead);

//uHead=UnionLink(pHead,qHead);

//UnionLink2(pHead,qHead);

//UnionLink3(pHead,qHead);

DelDua(pHead);

outputs(pHead);

clear(pHead);

//clear(qHead);

//clear(uHead);

return 0;

}