#include<iostream>

using namespace std;

#include<stdlib.h>

#define MAXSIZE 100

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define MAXSIZE 100

typedef int ElemType;

typedef int Status;

typedef struct LNode

{

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

//初始化

Status InitList(LinkList &L);

//输出

void OutList(LinkList &L);

//插入

Status ListInsert(LinkList &L,int i,ElemType e);

//删除

Status ListDelete(LinkList &L,int i);

//排序

void Sort(LinkList &L,LinkList &LB);

//取值

int GetElem(LinkList L,int min);

//有序插入

void Fun(LinkList &L,int e);

int main()

{

LNode \*L,\*LB;

int i,e,flag;

InitList(L);

InitList(LB);

//输入输出-------------------------------------------

cout<<"Please Input Some numbers:"<<endl;

for(i=1;i<=6;i++)

{

cin>>e;

ListInsert(L,i,e);

}

OutList(L);

//依次插入3、21、15、99四个数，分别插入在第1、8、4和12位置，每插入一次都要输出一次单链表。

e=3,i=1;

cout<<"将3插入到第1个位置"<<endl;

flag=ListInsert(L,i,e);

if(flag==ERROR) cout<<"插入失败"<<endl;

OutList(L);

e=21,i=8;

cout<<"将21插入到第8个位置"<<endl;

flag=ListInsert(L,i,e);

if(flag==ERROR) cout<<"插入失败"<<endl;

OutList(L);

e=15,i=4;

cout<<"将15插入到第4个位置"<<endl;

flag=ListInsert(L,i,e);

if(flag==ERROR) cout<<"插入失败"<<endl;

OutList(L);

e=99,i=12;

cout<<"将99插入到第12个位置"<<endl;

flag=ListInsert(L,i,e);

if(flag==ERROR) cout<<"插入失败"<<endl;

OutList(L);

//------------------------------------------------------------------

//5．删除第1，第9和第12个位置上的元素，每删除一个元素都要输出一次单链表

i=1;

cout<<"删除第1个位置上的元素"<<endl;

flag=ListDelete(L,i);

if(flag==ERROR) cout<<"删除失败"<<endl;

OutList(L);

i=9;

cout<<"删除第9个位置上的元素"<<endl;

flag=ListDelete(L,i);

if(flag==ERROR) cout<<"删除失败"<<endl;

OutList(L);

i=12;

cout<<"删除第12个位置上的元素"<<endl;

flag=ListDelete(L,i);

if(flag==ERROR) cout<<"删除失败"<<endl;

OutList(L);

//编写一个排序算法，对链表中元素从小到大排列。

cout<<"由小到大排序为："<<endl;

Sort(L,LB);

//向有序链表分别插入20和50，插入后表仍然有序

cout<<"有序插入20，结果为："<<endl;

Fun(LB,20);

OutList(LB);

cout<<"有序插入50，结果为："<<endl;

Fun(LB,50);

OutList(LB);

return 0;

}

//初始化

Status InitList(LinkList &L)

{

L=new LNode;

L->next=NULL;

return OK;

}

//输出

void OutList(LinkList &L)

{

LNode \*p;

p=L->next;

while(p!=NULL)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl<<endl;

}

//插入

Status ListInsert(LinkList &L,int i,ElemType e)

{

LNode \*p,\*s;

int j;

p=L;j=0;

while(p&&(j<i-1))

{p=p->next;j++;}

if(!p||j>i-1) return ERROR;

s=new LNode;

s->data=e;

s->next=p->next;

p->next=s;

return OK;

}

//删除

Status ListDelete(LinkList &L,int i)

{

LNode \*p,\*q;

int j;

p=L;j=0;

while((p->next)&&(j<i-1))

{p=p->next;j++;}

if(!(p->next)||j>i-1) return ERROR;

q=p->next;

p->next=q->next;

delete q;

return OK;

}

//排序

void Sort(LinkList &L,LinkList &LB)

{

LNode \*p;

InitList(LB);

int i=1,t,min;

p=L->next;

while(p!=NULL)

{

min=p->data;

while(p!=NULL)

{

if(p->data<min)

{

min=p->data;

}

p=p->next;

}

p=L->next;

ListDelete(L,GetElem(L,min));

ListInsert(LB,i,min);

i++;

p=L->next;

}

OutList(LB);

}

//取值

int GetElem(LinkList L,int min)

{

LNode \*p;

int j;

p=L->next;

j=1;

while(p!=NULL)

{

if(p->data==min) break;

j++;

p=p->next;

}

return j;

}

//有序插入

void Fun(LinkList &LB,int e)

{

LNode \*p;

int i=0,flag;

p=LB->next;

while(p)

{

if(e>=p->data) flag=i+1;

i++;

p=p->next;

}

ListInsert(LB,flag+1,e);

}