#include<string.h>  
#include<ctype.h>  
#include<malloc.h> /\* malloc()等 \*/  
#include<limits.h> /\* INT\_MAX等 \*/  
#include<stdio.h> /\* EOF(=^Z或F6),NULL \*/  
#include<stdlib.h> /\* atoi() \*/  
#include<io.h> /\* eof() \*/  
#include<math.h> /\* floor(),ceil(),abs() \*/  
#include<process.h> /\* exit() \*/  
#define TRUE 1  
#define FALSE 0  
#define OK 1  
#define ERROR 0  
#define INFEASIBLE -1  
  
#define LIST\_INIT\_SIZE 100  
#define LISTINCREMRNT 10  
  
typedef int ElemType;  
typedef int Status;   
typedef int Boolean;   
  
typedef struct   
{  
 ElemType \* elem; //储存空间基地址

int length; // 记录当前链表长度  
 int listsize; //链表规模  
} SqList;  
  
  
Status InitList(SqList \*L)  
{  
(\*L).elem = (ElemType\*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));  
if(!(\*L).elem)  
exit(OVERFLOW);  
(\*L).length = 0;  
(\*L).listsize = LIST\_INIT\_SIZE;  
return OK;  
}  
  
Status DestroyList(SqList \*L)  
{ /\* 操作结果：三元组T被销毁 \*/  
free((\*L).elem);  
 (\*L).elem=NULL;  
 (\*L).length =0;  
 return OK;  
}  
  
void ClearList(SqList \*L)

{  
(\*L).length = 0;  
}  
  
Status ListEmpty(SqList L) //值拷贝  
{  
return L.length ==0?1:0;   
  
}  
  
  
Status ListLength(SqList L)  
{  
 return L.length ;  
}  
  
Status GetEle(SqList L,int i,int \*e)  
{  
if(i<1||i>L.length)  
 return ERROR;  
\*e = L.elem[i-1];  
return OK;  
}  
  
Status ListInsert(SqList \*L ,int i, int e)  
{  
int \*newbase;  
int \*p, \*q;  
if(i<1||i>(\*L).length+1)  
return ERROR;  
if((\*L).length>(\*L).listsize)  
{  
newbase = (ElemType\*)realloc((\*L).elem, ((\*L).listsize + LISTINCREMRNT) \* sizeof(ElemType));  
if(!newbase)  
exit(OVERFLOW);  
(\*L).elem = newbase;  
(\*L).listsize += LISTINCREMRNT;  
}  
q = &((\*L).elem[i-1]);  
for(p=&(\*L).elem[(\*L).length - 1]; p>=q; --p)  
{  
\*(p+1) = \*p;  
}  
\*q = e;  
++(\*L).length;  
return OK;  
}  
  
Status LocateElem(SqList \*L,int e)   
{  
int i = 1;  
while (i<(\*L).length && (\*L).elem[i-1])  
i++;  
if(i<(\*L).length)  
return i;  
else  
return ERROR;  
}  
  
Status ListDelete(SqList \*L,int i,int \*e)  
{  
int \*p;  
if(i<1||i>(\*L).length)  
return ERROR;  
\*e = (\*L).elem[i-1];  
for(p=&(\*L).elem[i-1];p<&(\*L).elem[(\*L).length-2];p--)  
{  
\*p = \*(p+1);  
}  
(\*L).length--;  
return OK;  
}  
  
Status ListTraverse(SqList L)  
{  
for(int i=0;i<L.length-1;i++)  
{  
printf(" %d ",L.elem [i]);  
}  
return OK;  
}  
void MergeList(SqList La, SqList Lb,SqList \*Lc)  
{  
InitList(Lc);  
int i=1, j=1, k=0;  
int La\_length = ListLength(La);  
int Lb\_length = ListLength(Lb);  
int ai,bj;  
while ((i<=La\_length) && (j<=Lb\_length))  
{  
  
GetEle(La, i, &ai);  
GetEle(Lb, j, &bj);  
if(ai<=bj)  
{  
ListInsert(Lc, ++k, ai);  
++i;  
}  
else  
{  
ListInsert(Lc, ++k, bj);  
++j;  
}  
}  
while(i<=La\_length)  
{  
GetEle(La, i++, &ai);  
ListInsert(Lc, ++k, ai);  
}  
  
while(j<=Lb\_length)  
{  
GetEle(Lb, j++, &bj);  
ListInsert(Lc, ++k, bj);  
}  
  
   
}  
  
void mergelist\_sq(SqList la,SqList lb,SqList &lc);  
int findMin(SqList \*L) ;  
int findMax(SqList \*L) ;  
void main()  
{  
SqList L1,L2,L3;  
SqList la,lb,lc;  
//int e;  
int i;   
if(InitList(&L1)==1 && InitList(&L2)==1)  
{  
 printf("顺序表初始化成功\n");   
 }   
  
for(i=1;i<10;i++)  
ListInsert(&L1,i, 2\*i);  
for(i=1;i<10;i++)  
ListInsert(&L2,i, (2\*i+1));  
  
findMin(&L1);  
findMax(&L1);  
MergeList(L1, L2, &L3);  
ListTraverse(L3);  
DestroyList(&L1);  
DestroyList(&L2);  
DestroyList(&L3);  
  
  
}  
  
int findMax(SqList \*L) {  
int index = 0, max = L->elem[0];  
for (int i = 0; i < L->length; i++) {  
if (max < L->elem[i]) {  
index = i;  
max = L->elem[i];  
}  
}  
printf("%d,",max);  
return max;  
}  
  
int findMin(SqList \*L) {  
int index = 0, min = L->elem[0];  
for (int i = 0; i < L->length; i++) {  
if (min > L->elem[i]) {  
index = i;  
min = L->elem[i];  
}  
}  
printf("%d,",min);   
return min;  
}  
  
void mergelist\_sq(SqList la,SqList lb,SqList &lc)   
{   
 int \*pa,\*pb,\*pc,\*pa\_last,\*pb\_last,i;   
 pa=la.elem;pb=lb.elem;   
 lc.listsize=la.length+lb.length;   
 pc=lc.elem=(ElemType\*)malloc(lc.listsize\*sizeof(ElemType));   
 pa\_last=la.elem+la.length-1;   
 pb\_last=lb.elem+lb.length-1;   
 while(pa<=pa\_last&&pb<=pb\_last){   
 if(\*pa<=\*pb)\*pc++=\*pa++;   
 else \*pc++=\*pb++;}   
 while(pa<=pa\_last) \*pc++=\*pa++;   
 while(pb<=pb\_last) \*pc++=\*pb++;   
 for(i=0;i<lc.listsize-1;i++);

for( i=0; i<lc.listsize; i++ )  
 printf("%d,",lc.elem[i]);   
}