#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;

}

Status max(SqList \*L)

{

int i, max;

max = L->elem[0];

for (i = 0; i < L->length; i++)

{

if (max < L->elem[i + 1])

max = L->elem[i + 1];

}

printf("max=%d\n", max);

return OK;

}

Status t(SqList \*L)

{

int a, min;

min =(\*L).elem[0];

for (a=0;a<(\*L).length&&a>0;a++)

{

if (min > (\*L).elem[a + 1])

{

min = (\*L).elem[a+1];

}

}

printf("min=%d\n", min);

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);

}

}

int main()

{

SqList L1, L2, L3;

//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);

max(&L1);

t(&L1);

for (i = 1; i < 10; i++)

ListInsert(&L2, i, (2 \* i + 1));

max(&L2);

t(&L2);

MergeList(L1, L2, &L3);

ListTraverse(L3);

DestroyList(&L1);

DestroyList(&L2);

DestroyList(&L3);

}

实验总结：1．集合中必存在唯一的一个"第一个元素"；

2．集合中必存在唯一的一个"最后的元素"；

3．除最后元素之外，其它数据元素均有唯一的"后继"；

4．除第一元素之外，其它数据元素均有唯一的"前驱"。

数据结构中线性结构指的是数据元素之间存在着“一对一”的线性关系的数据结构。

更好的认识了线性结构