## 实验1

学号：2022217587姓名：党存远专业：物联网工程

|  |  |  |  |
| --- | --- | --- | --- |
| **“数据结构课程实验”验收细则** | | | |
| **成绩等级** | **具体表现** | **教师评分**  **（百分制）** | |
| A（100-90] | 1）能较快完成实验；2）编码正确；3）能完整正确地回答教师提问； | □ |  |
| B（90-80] | 1）能够在规定时间内完成实验；2）编码正确；3）能正确地回答教师提问； | □ |  |
| C（80-70] | 1）能够在规定时间内完成实验； | □ |  |
| D（70-60] | 1）虽完成实验但未能在规定时间内完成验收的； | □ |  |
| E（<60） | 1）未能进行验收的； | □ |  |

**１.1 实验目的**

熟练掌握线性表的顺序存储结构。

熟练掌握顺序表的有关算法设计。

根据具体问题的需要，设计出合理的表示数据的顺序结构，并设计相关算法。

**1.2 实验要求**

顺序表结构和运算定义，算法的实现以库文件方式实现，不得在测试主程序中直接实现；

比如存储、算法实现放入文件：seqList.h

实验程序有较好可读性，各运算和变量的命名直观易懂，符合软件工程要求；

程序有适当的注释。

**1.3 实验任务**

编写算法实现下列问题的求解。

<1>求顺序表中第i个元素（函数），若不存在，报错。

实验测试数据基本要求：

第一组数据：顺序表长度n≥10，i分别为5，n，0，n+1，n+2

第二组数据：顺序表长度n=0，i分别为0，2

<2>在第i个结点前插入值为x的结点。

实验测试数据基本要求：

第一组数据：顺序表长度n≥10，x=100, i分别为5,n,n+1,0,1,n+2

第二组数据：顺序表长度n=0，x=100，i=5

<3>删除顺序表中第i个元素结点。

实验测试数据基本要求：

第一组数据：顺序表长度n≥10，i分别为5,n,1,n+1,0

第二组数据：顺序表长度n=0， i=5

<4>在一个递增有序的顺序表L中插入一个值为x的元素，并保持其递增有序特性。

实验测试数据基本要求：

顺序表元素为（10,20,30,40,50,60,70,80,90,100）,

x分别为25，85，110和8

<5>将顺序表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数），分别放入新的顺序表中，然后原表和新表元素同时输出到屏幕上，以便对照求解结果。

实验测试数据基本要求：

第一组数据：顺序表元素为（1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）

第二组数据：顺序表元素为（10,20,30,40,50,60,70,80,90,100）

<6>求两个递增有序顺序表L1和L2中的公共元素，放入新的顺序表L3中。

实验测试数据基本要求：

第一组

第一个顺序表元素为（1，3，6，10，15，16，17，18，19，20）

第二个顺序表元素为（1，2，3，4，5，6，7，8，9，10，18，20，30）

第二组

第一个顺序表元素为（1，3，6，10，15，16，17，18，19，20）

第二个顺序表元素为（2，4，5，7，8，9，12，22）

第三组

第一个顺序表元素为（）

第二个顺序表元素为（1，2，3，4，5，6，7，8，9，10）

<7>删除递增有序顺序表中的重复元素，并统计移动元素次数，要求时间性能最好。

实验测试数据基本要求：

第一组数据：顺序表元素为（1,2,3,4,5,6,7,8,9）

第二组数据：顺序表元素为（1,1,2,2,2,3,4,5,5,5,6,6,7,7,8,8,9）

第三组数据：顺序表元素为（1,2,3,4,5,5,6,7,8,8,9,9,9,9,9）

**1.4\* 顺序表扩展实验**

非必做内容，有兴趣的同学选做，

<1>递增有序顺序表L1、L2，对2表进行合并，并使得合并后成为一个集合，集合的元素放回L1表中保存，要求时间性能最好。

<2>（递增有序）顺序表表示集合A、B，实现：

C=A∩B，C=A∪B，C=A-B

A=A∩B，A=A∪B，A=A-B

<3>（递增有序）顺序表表示集合A、B，判定A是否B的子集。

<4>(2011)（15 分）一个长度为L（L≥1）的升序序列S，处在第L/2个位置的数称为S 的中位数。例如，若序列S1=(11, 13, 15, 17, 19)，则S1 的中位数是15。两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S2=(2, 4, 6, 8, 20)，则S1 和S2 的中位数是11。

现有两个等长升序序列A 和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A 和B 的中位数。要求：

（1）给出算法的基本设计思想。

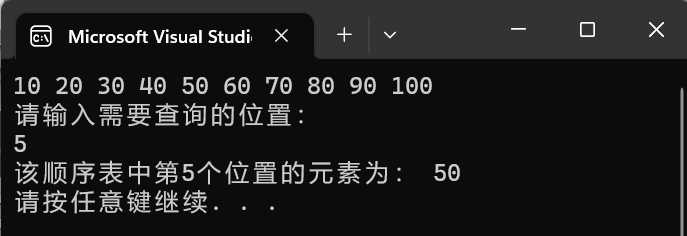
（2）根据设计思想，采用C 或C++语言描述算法，关键之处给出注释。

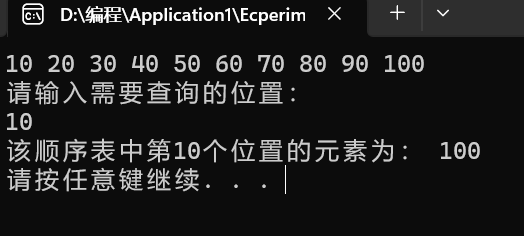
（3）说明你所设计算法的时间复杂度和空间复杂度。

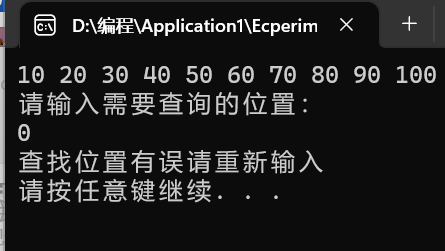
**1.5 验收内容：对线性表的理解及算法具体实现**

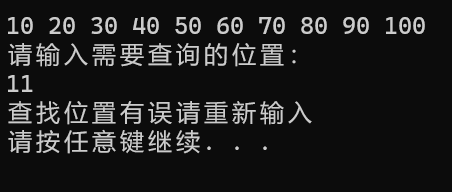
**1.6 运行结果截图及说明**

**A.求出顺序表中第i个元素并输出**

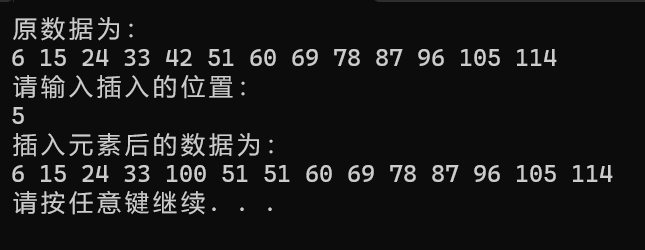


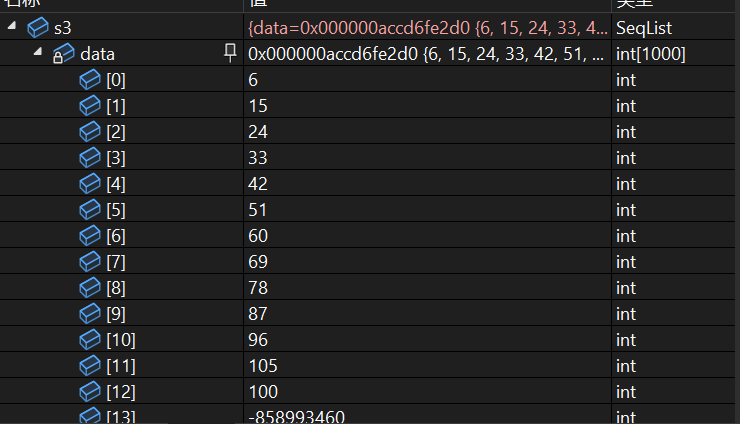


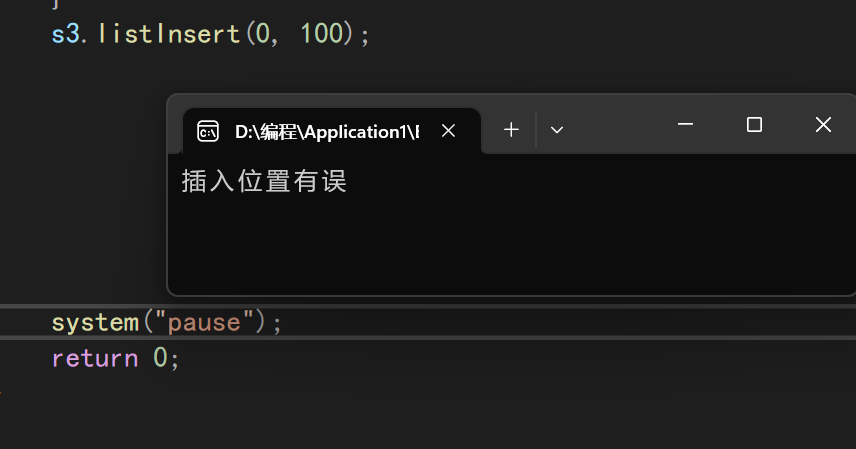


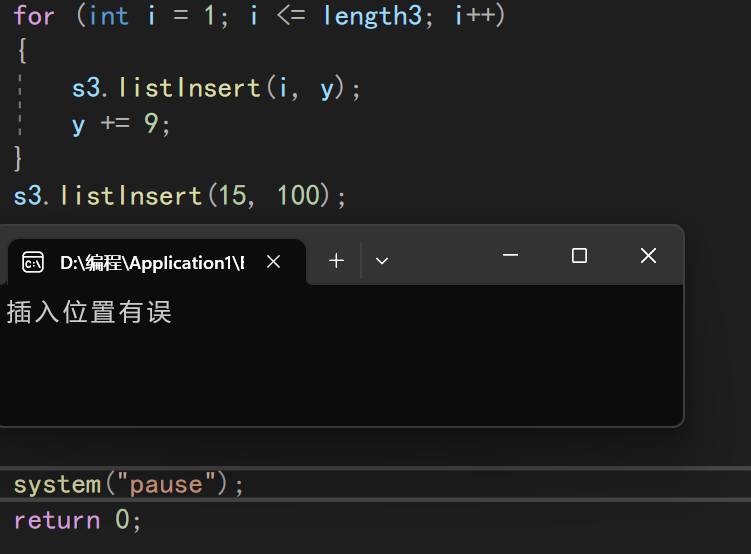


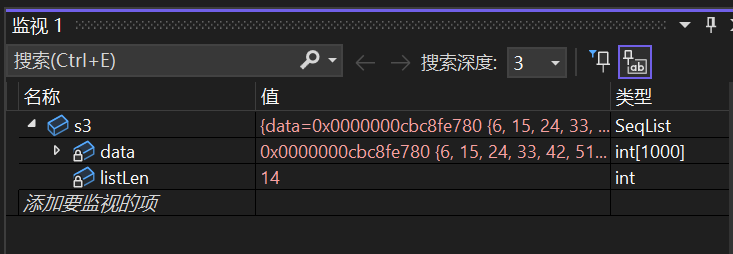
图一：求顺序表中第i个元素

B.



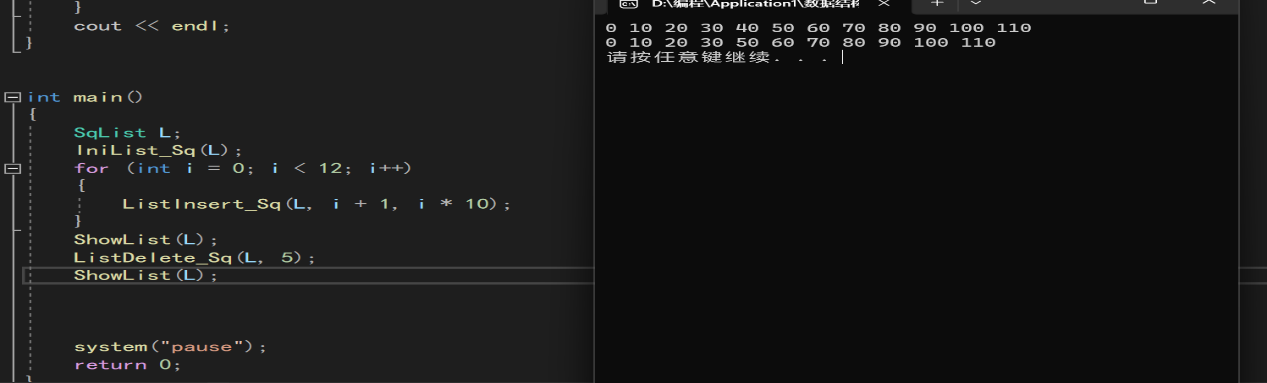




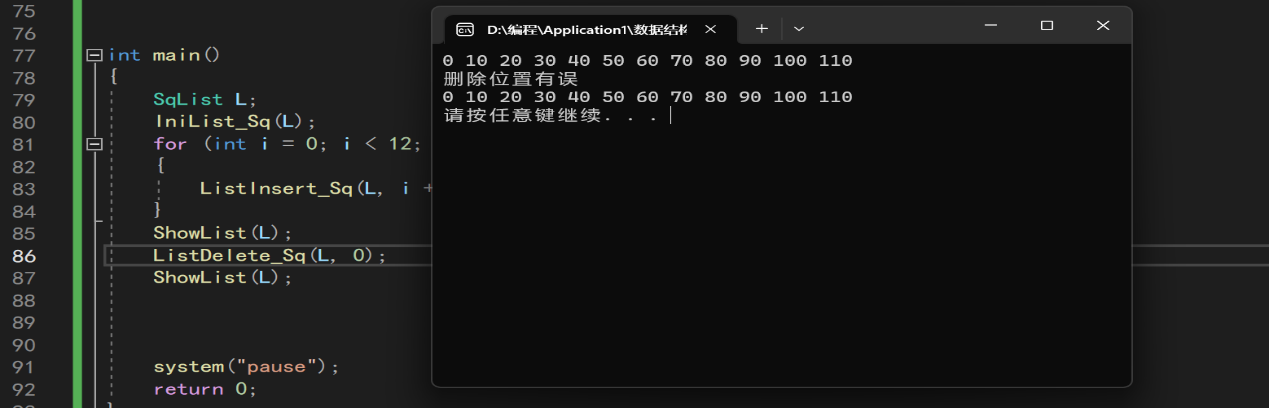


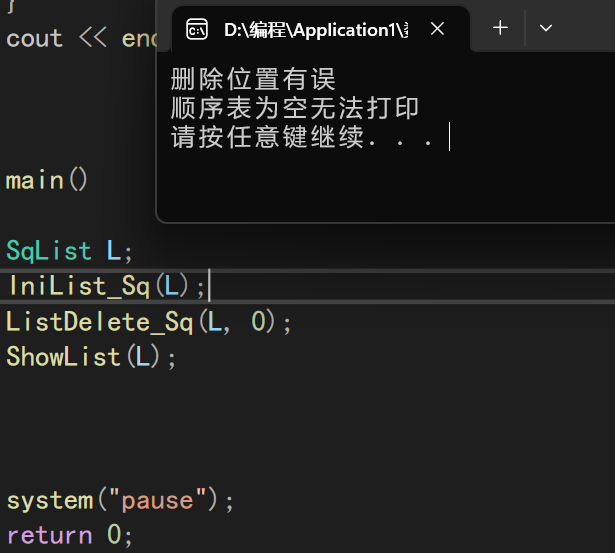
图二：在第i个结点前插入值为x的结点

C.



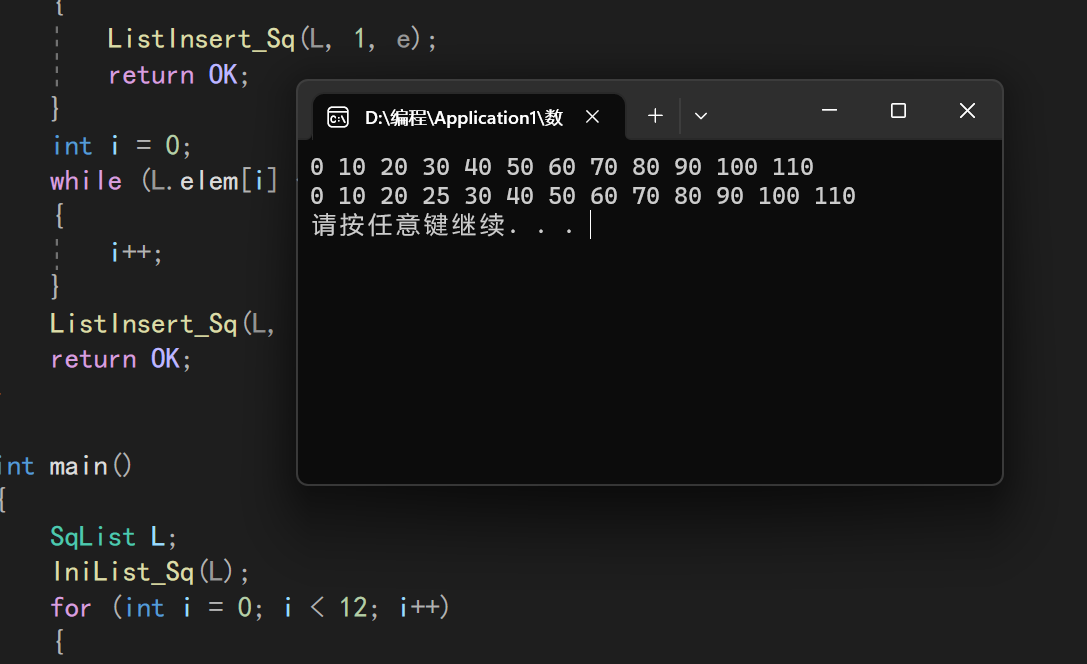


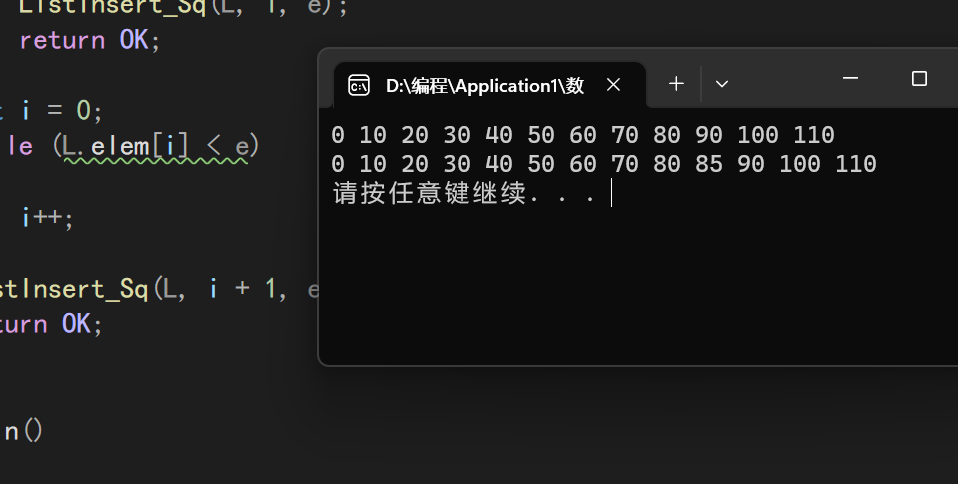


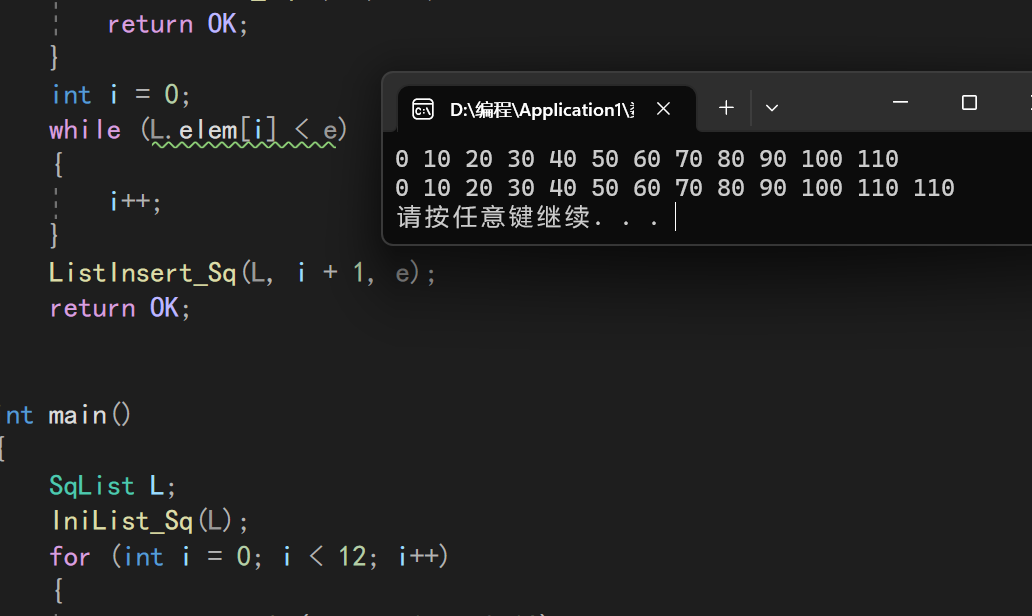


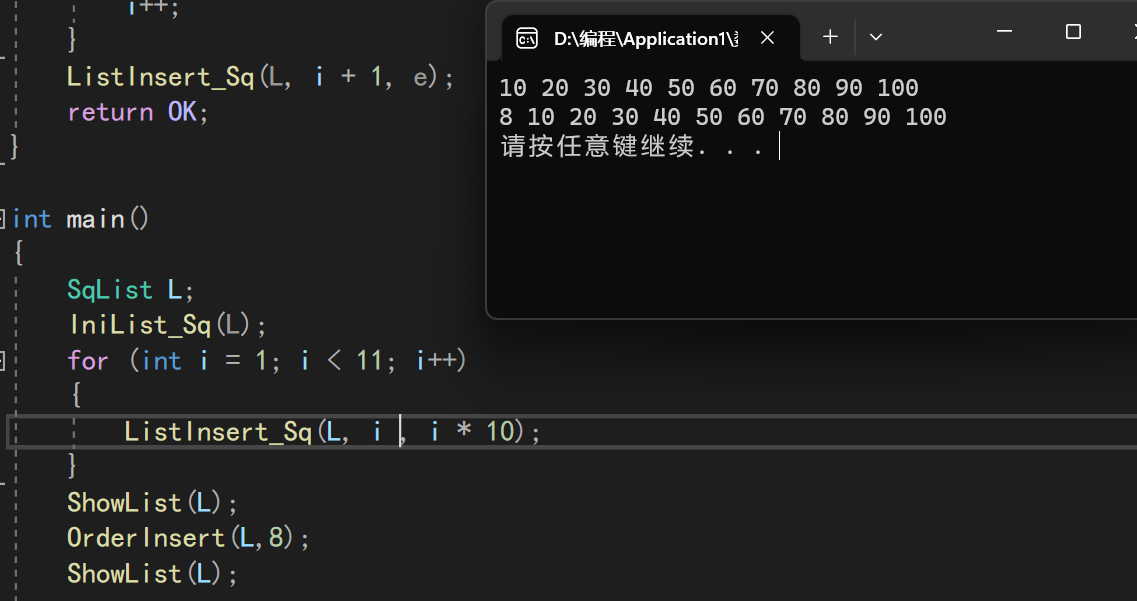
图三：删除表中第i个结点

D.



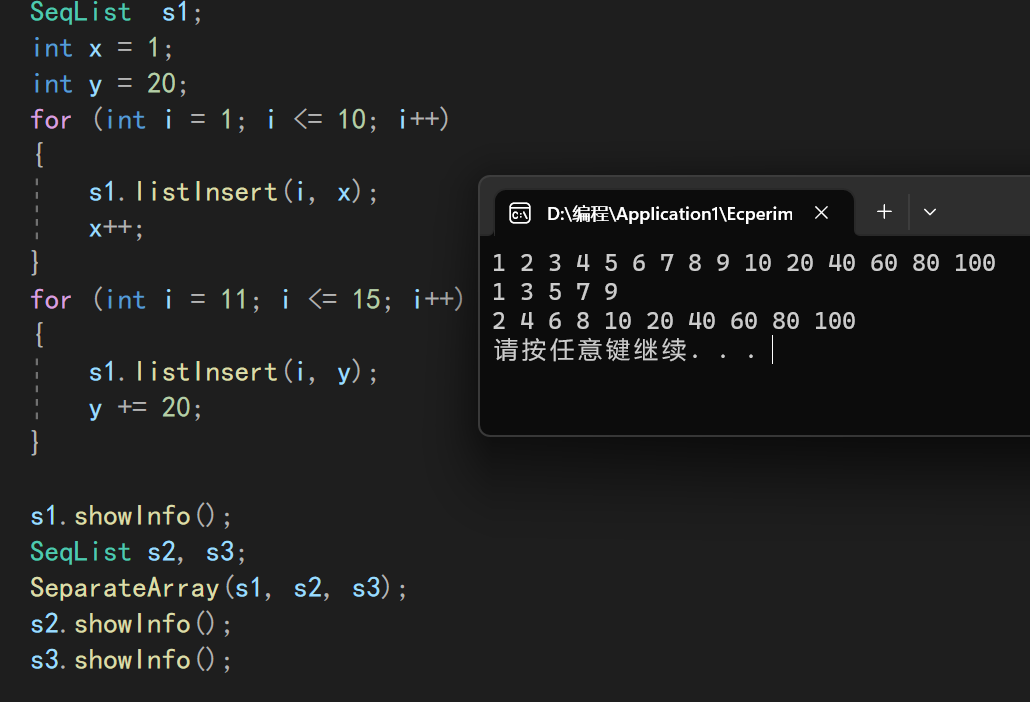


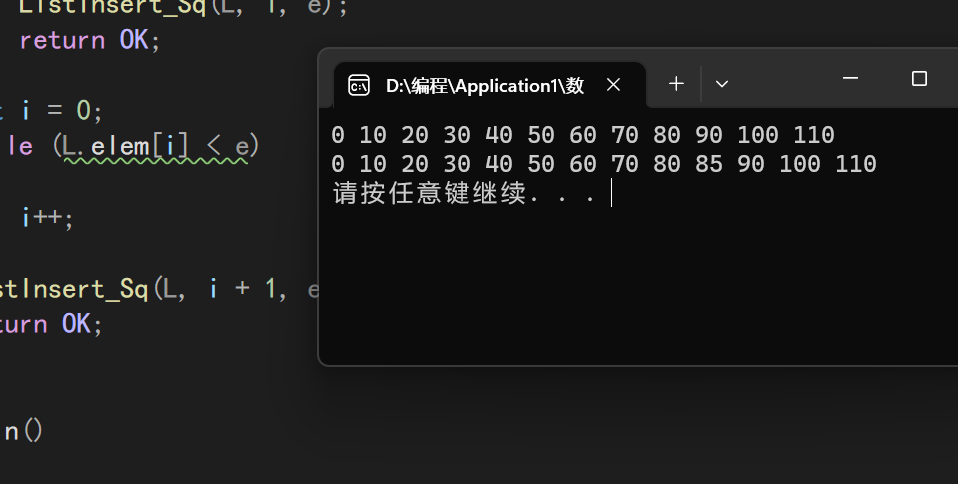




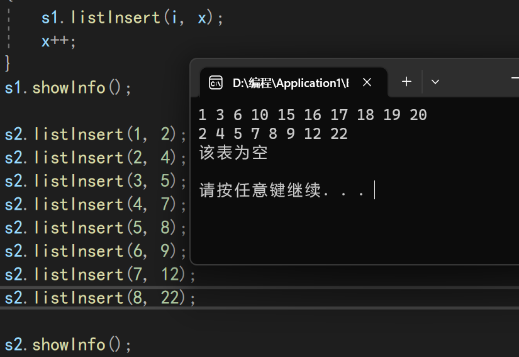
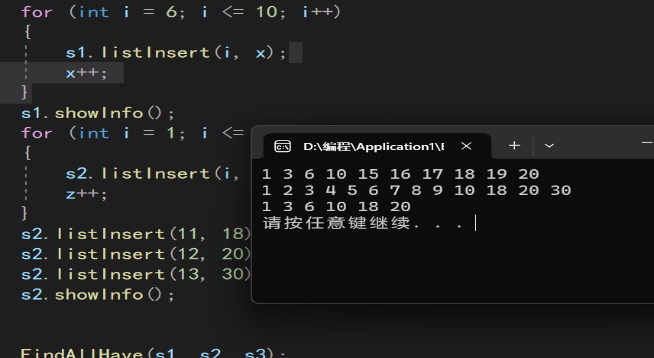
图四：在递增有序的顺序表中插入元素使其仍递增有序

E.



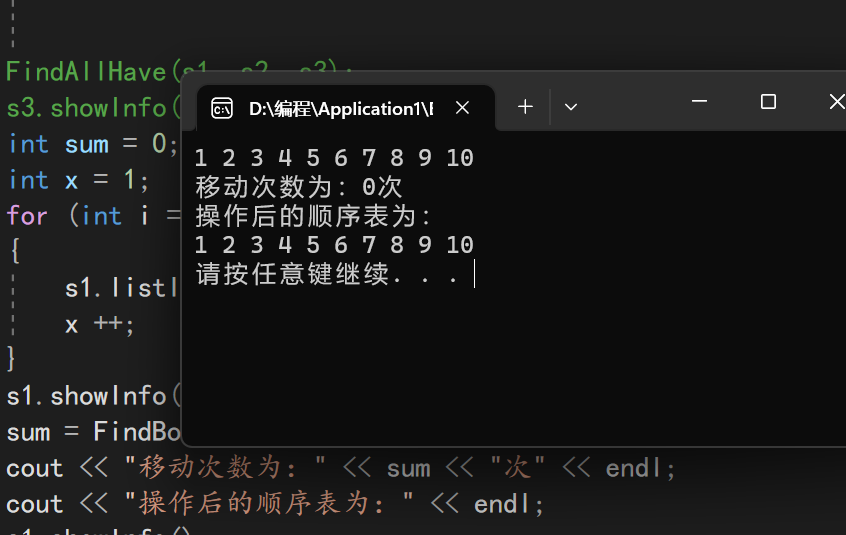


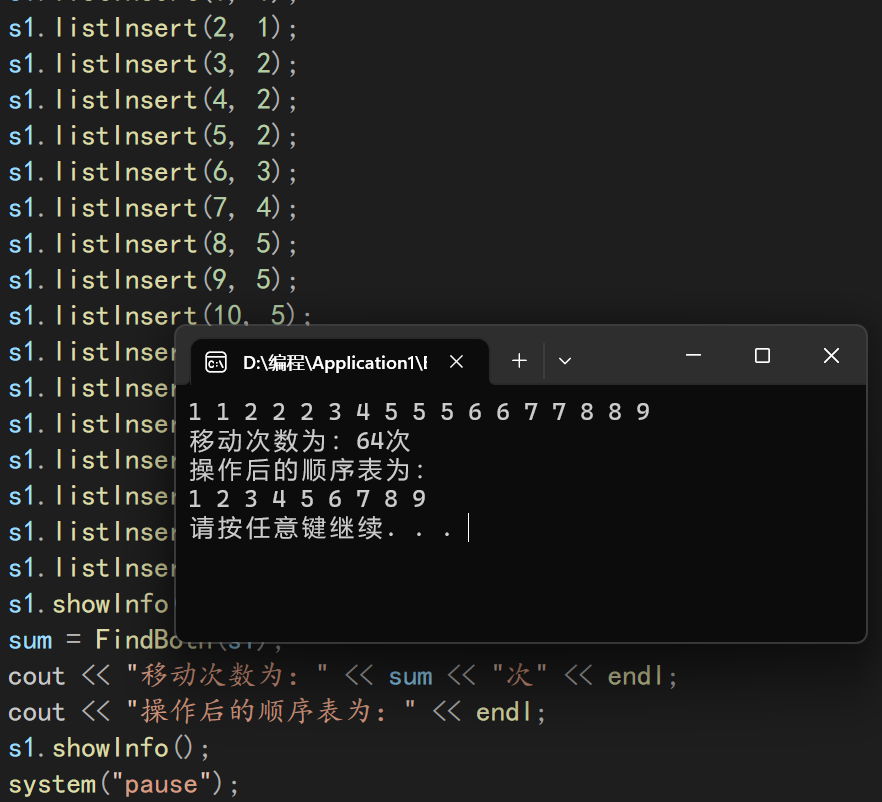
图五：将顺序表中的奇数项与偶数项分开

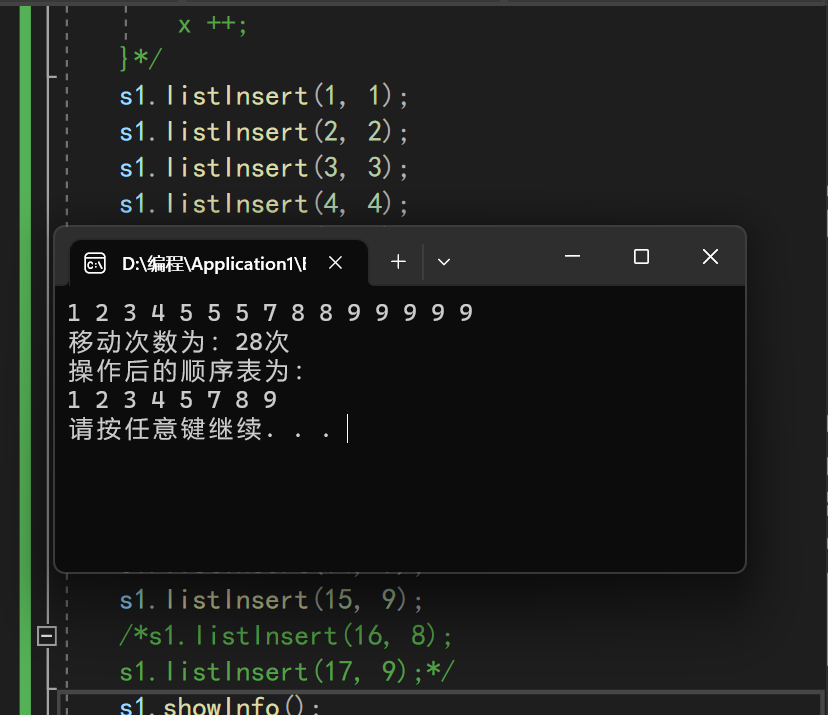
**F．**

**图六：将两个表进行合并操作**

**G.**

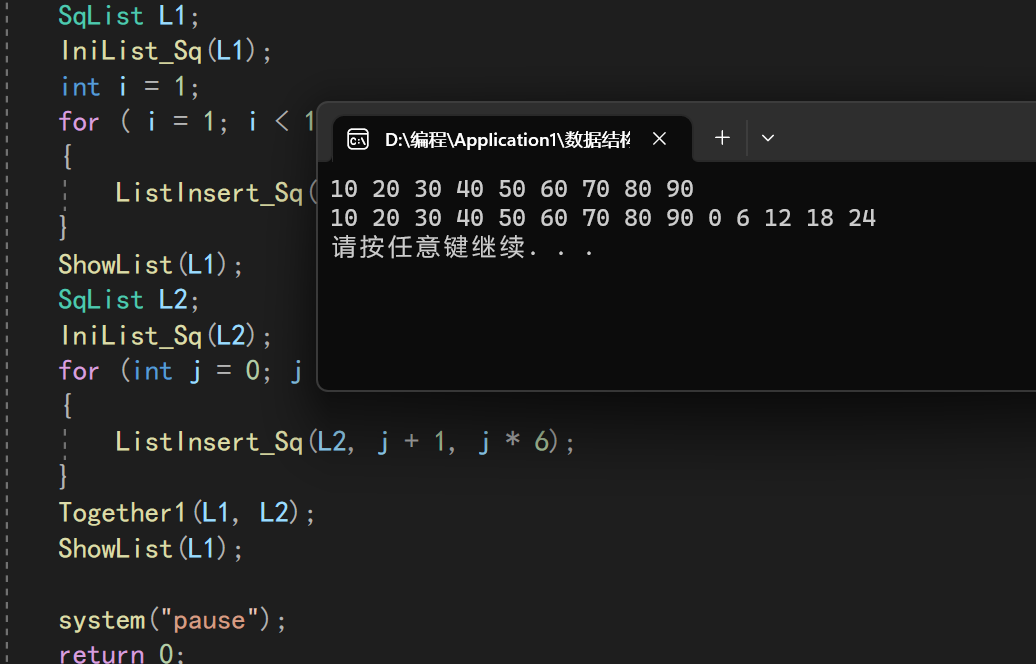


****

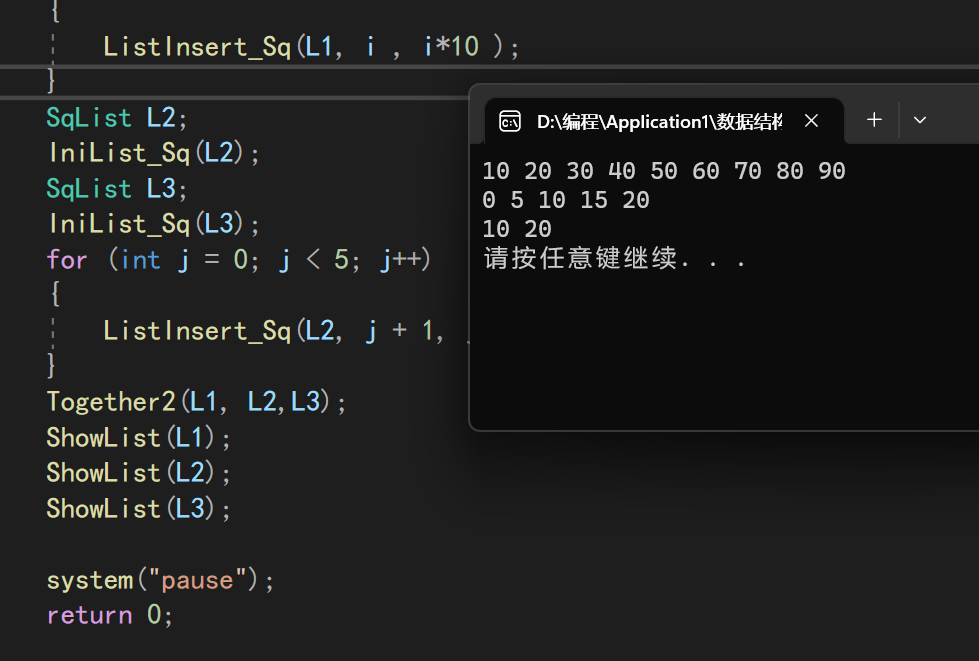
****

**图七：将顺序表中相同元素删除并且计算移动次数**

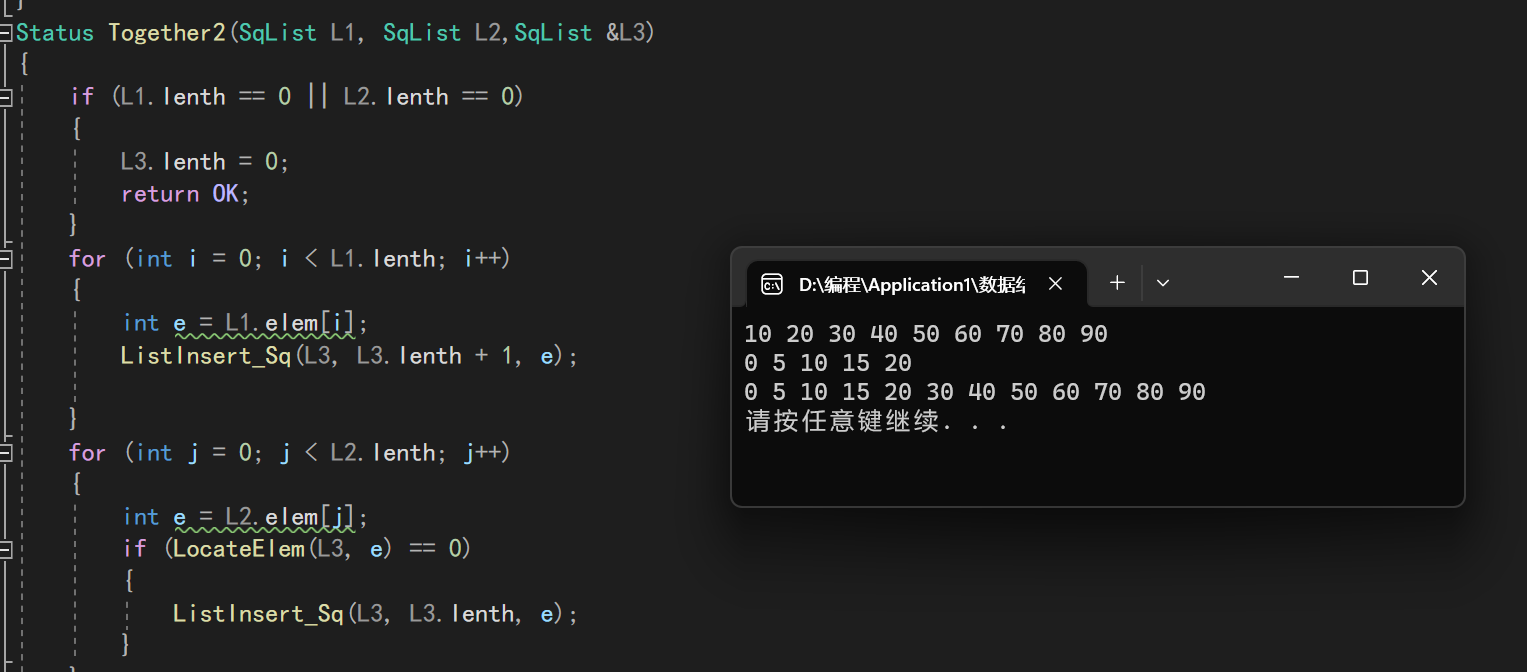
**H.**

****

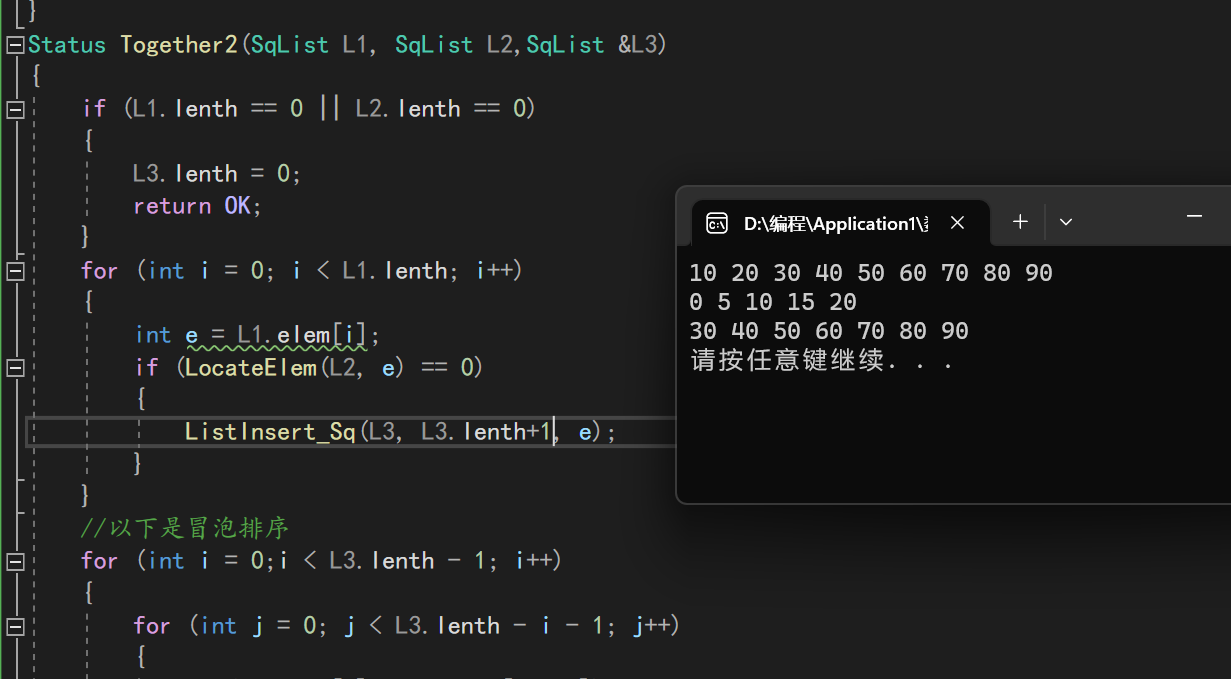
**图一：合并表AB**

****

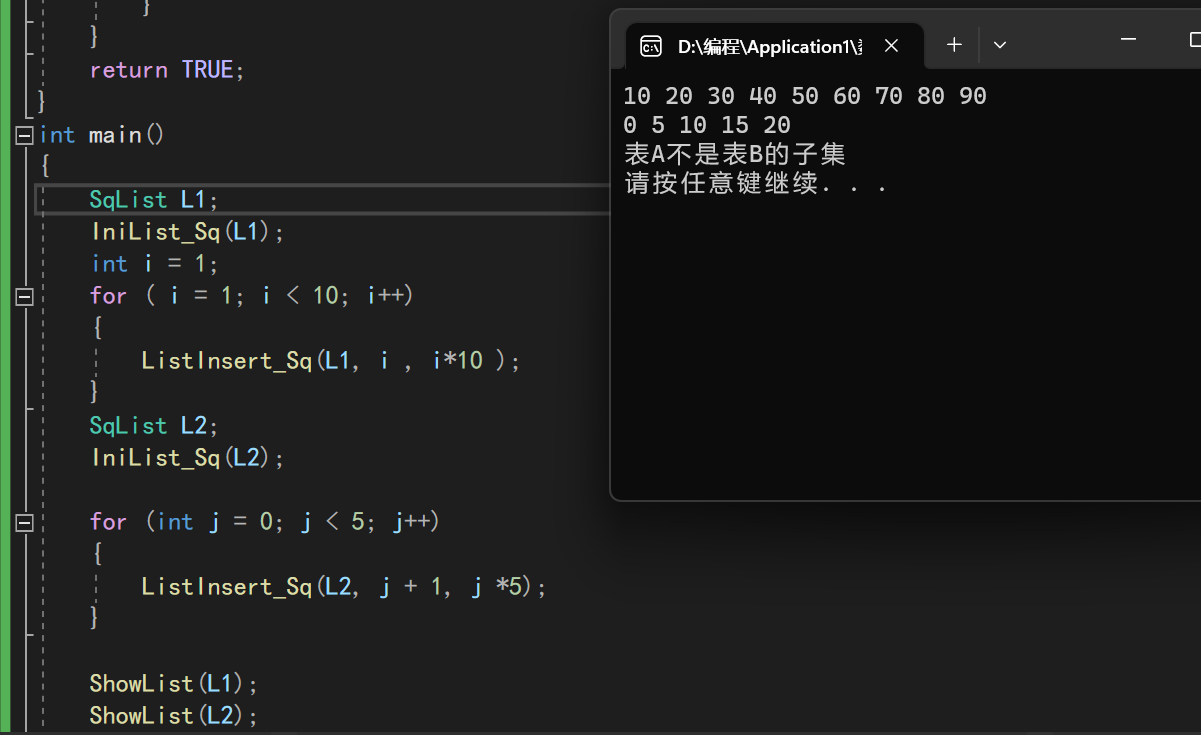
**图二：求两表的交集**

****

**图三：求两表的并集**

****

**图四：求A-B表**

****

**图五：判断表A是否为表B的子集**

**1.7 附源代码**

A.

void SeqList::FindArray(int i)

{

if (i<1 || i>listLen)

{

cout << "查找位置有误请重新输入" << endl;

}

else

{

cout << "该顺序表中第" << i << "个位置的元素为： ";

cout << data[i - 1] << endl;

}

}

B.

int SeqList::listInsert(int i, int x)

{

int j = 0;

if (listLen == MaxLen)//如果表满则返回0

{

cout << "插入位置有误" << endl;

return 0;

}

else if (i<1 || i>listLen + 1)//插入位置为0或者大于表长度，则返回1

{

cout << "插入位置有误" << endl;

return 1;

}

else

{

for (j = listLen - 1; j > i - 1; j--)//数组元素下标

{

data[j + 1] = data[j];//将插入位置以后的每个元素向后移一位

}

data[i - 1] = x;//将插入位置赋值为x

listLen++;//插入后表长度+1;

return 2;//成功以后则返回2;

}

}

C.

int SeqList::listDelete(int i)

{

int j = 0;

if (listLen <= 0)

{

return 0;//即为空表

}

else if (i<1 || i>listLen)//删除的元素位置不符合要求

{

return 1;

}

else

{

for (j = i; j < listLen; j++)

{

data[j - 1] = data[j];

}

listLen--;

return 2;//删除成功返回2

}

}

D.

int SeqList::incInsert(int x)//让一个递增有序的顺序表仍旧递增有序

{

int i = listLen - 1;

if (listLen == MaxLen)

{

cout << "插入位置有误：" << endl;

return -1;

}

else

{

while (i >= 0 && data[i] > x)

{

data[i + 1] = data[i];

i--;

}

}

data[i + 1] = x;

listLen++;

return i + 2;//返回元素的位置(元素编号)

}

E.

void SeparateArray(SeqList& A, SeqList& C,SeqList &D)

{

int ia = 1, ib = 1, ic = 1,id=1;

int y = 0,z=0;

while(ia <= A.length() )

{

A.getElement(ia, y);

ia++;

if (y % 2)

{

C.listInsert(ic, y);

ic++;

}

else

{

D.listInsert(id, y);

id++;

}

}

}

SeqList s1;

int x = 1;

int y = 20;

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

{

s1.listInsert(i, x);

x++;

}

for (int i = 11; i <= 15; i++)

{

s1.listInsert(i, y);

y += 20;

}

s1.showInfo();

SeqList s2, s3;

SeparateArray(s1, s2, s3);

s2.showInfo();

s3.showInfo();

system("pause");

return 0;

F．

void FindAllHave(SeqList A,SeqList B,SeqList &C)

{

int ia = 1, ib = 1, ic = 1;

int x = 0, y = 0;

while (ia <= A.length())

{

A.getElement(ia, x);

B.getElement(ib, y);

if (x < y)

{

ia++;

}

else if (x > y)

{

ib++;

}

else if (x == y)

{

C.listInsert(ic, x);

ic++;

ia++;

ib++;

}

}

}

SeqList s2, s3;

int x = 1;

int y = 2;

int z = 1;

for (int i = 1; i <= 5; i++)

{

s1.listInsert(i, x);

x += y;

y++;

}

x = 16;

for (int i = 6; i <= 10; i++)

{

s1.listInsert(i, x);

x++;

}

s1.showInfo();

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

{

s2.listInsert(i, z);

z++;

}

s2.listInsert(11, 18);

s2.listInsert(12, 20);

s2.listInsert(13, 30);

s2.showInfo();

FindAllHave(s1, s2, s3);

s3.showInfo();

system("pause");

return 0;

}

SeqList s2, s3;

int x = 1;

int y = 2;

int z = 1;

for (int i = 1; i <= 5; i++)

{

s1.listInsert(i, x);

x += y;

y++;

}

x = 16;

for (int i = 6; i <= 10; i++)

{

s1.listInsert(i, x);

x++;

}

s1.showInfo();

s2.listInsert(1, 2);

s2.listInsert(2, 4);

s2.listInsert(3, 5);

s2.listInsert(4, 7);

s2.listInsert(5, 8);

s2.listInsert(6, 9);

s2.listInsert(7, 12);

s2.listInsert(8, 22);

s2.showInfo();

FindAllHave(s1, s2, s3);

s3.showInfo();

system("pause");

return 0;

G.

int FindBoth(SeqList& A)

{

int ia = 1;

int x = 0,y = 0;

int sum = 0;

while (ia+1 <= A.length() )

{

A.getElement(ia, x);

A.getElement(ia + 1, y);

if (x == y)

{

sum+=A.listDelete(ia + 1);

ia--;

}

ia++;

}

return sum;

}

int sum = 0;

int x = 1;

/\*for (int i = 1; i <= 10; i++)

{

s1.listInsert(i, x);

x ++;

}\*/

s1.listInsert(1, 1);

s1.listInsert(2, 1);

s1.listInsert(3, 2);

s1.listInsert(4, 2);

s1.listInsert(5, 2);

s1.listInsert(6, 3);

s1.listInsert(7, 4);

s1.listInsert(8, 5);

s1.listInsert(9, 5);

s1.listInsert(10, 5);

s1.listInsert(11, 6);

s1.listInsert(12, 6);

s1.listInsert(13, 7);

s1.listInsert(14, 7);

s1.listInsert(15, 8);

s1.listInsert(16, 8);

s1.listInsert(17, 9);

s1.showInfo();

sum = FindBoth(s1);

cout << "移动次数为：" << sum << "次" << endl;

cout << "操作后的顺序表为：" << endl;

s1.showInfo();

system("pause");

return 0;

}

H.

void Voluation(SeqList& A, SeqList C)

{

for (int i = 0; i < C.length(); i++)

{

A.data[i] = C.data[i];

}

A.listLen = C.listLen;

}

int x = 0;

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

{

s1.listInsert(i, x);

x += 5;

}

SeqList s2;

int y = 5;

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

{

s2.listInsert(i, y);

y += 7;

}

SeqList s3;

mergeList(s1, s2, s3);

s1.showInfo();

s2.showInfo();

Voluation(s1, s3);

int z = 0, m = 0;

for (int i = 1; i < s1.length(); i++)

{

s1.getElement(i, z);

s1.getElement(i + 1, m);

if (z==m)

{

s1.listDelete(i);

}

}

s1.showInfo();

system("pause");

return 0;

}

int x = 0;

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

{

s1.listInsert(i, x);

x += 5;

}

SeqList s2;

int y = 5;

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

{

s2.listInsert(i, y);

y += 7;

}

SeqList s3;

mergeList(s1, s2, s3);

s1.showInfo();

s2.showInfo();

/\*mergeList(s1, s2, s3);\*/

/\*Voluation(s1, s3);\*/

int z = 0, m = 0;

for (int i = 1; i+1 <= s3.length(); i++)

{

s3.getElement(i, z);

s3.getElement(i + 1, m);

if (z==m)

{

s3.listDelete(i);

}

}

s3.showInfo();

system("pause");

return 0;

int x = 0;

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

{

s1.listInsert(i, x);

x += 5;

}

SeqList s2;

int y = 5;

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

{

s2.listInsert(i, y);

y += 7;

}

SeqList s3;

/\*mergeList(s1, s2, s3);\*/

s1.showInfo();

s2.showInfo();

Same(s1, s2, s3);

s3.showInfo();

system("pause");

return 0;

}

void Same(SeqList A, SeqList B, SeqList& C)

{

int ia = 1, ic = 1;

int x; int y;

while (ia <= A.length())

{

A.getElement(ia, x);

y = B.locate(x);

if (y != -1)

{

C.listInsert(ic, x);

ic++;

}

ia++;

}

}

int x = 0;

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

{

s1.listInsert(i, x);

x += 5;

}

SeqList s2;

int y = 5;

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

{

s2.listInsert(i, y);

y += 7;

}

SeqList s3;

/\*mergeList(s1, s2, s3);\*/

s1.showInfo();

s2.showInfo();

//FindAllHave(s1, s2,s3);

//FindBoth(s3);

//Same(s1, s2, s3);

mergeSet(s1, s2, s3);

s3.showInfo();

system("pause");

return 0;

}

bool subset(SeqList& A, SeqList& B)

{

int ia = 1, ib = 1;

int x = 0, y = 0;

while (ia <= A.length() && ib <= B.length())

{

A.getElement(ia, x);

B.getElement(ib, y);

if (x == y)

{

ia++;

ib++;

}

else if (x > y)

{

ib++;

}

else

{

return false;

}

if (ia >= A.length())

{

return true;

}

else

{

return false;//表A已经遍历结束但是表B并未遍历结束

}

}

}

中位数：

void FindMiddle(SeqList A, SeqList B,int &Ax,int &By)

{

int x = A.length();

int y = B.length();

int middle=0;

int A1, B1,A2,B2;

if (x % 2==1)

{

A.getElement((x + 1 / 2), Ax);

B.getElement((y + 1 / 2), By);

}

else

{

A.getElement((x / 2), A1);

A.getElement((x / 2)+1, A2);

Ax = (A2 + A1) / 2;

B.getElement((y / 2), B1);

B.getElement((y / 2)+1, B2);

By = (B2 + B1) / 2;

}

}

**1.8 调试过程中出现的bug及总结**

在这次实验的编写过程中，在删除表中相同元素的时候出现了无法将所有相同元素进行删除操作，现在发现是由于循环的次数导致无法全部删除。