《算法与数据结构》实验指导及报告书

**（2019版）**

**2019 - 2020学年 第 2学期**

**姓 名 周 吉 瑞**

**学 号： 20190521340**

**班 级：\_2019级 计算机科学与技术专业 3班\_**

**指导教师：\_\_ 贾 春 花\_\_\_\_\_\_\_**

**楚雄师范学院信息学院**

**2019**

# 实验1 C语言的函数、数组、指针、结构体知识

**实验日期**： 2020年 4月 7日 **报告日期**： 2020 年 4 月 10 日

### 一、实验目的

1、复习C语言中函数、数组、指针、结构体等的概念。

2、熟悉利用C语言进行程序设计的一般方法。

### 二、实验预习

复习以下C语言中的概念

1. 函数：函数是一个自我包含的完成一定相关功能的执行代码段。
2. 数组：所谓数组,是有序的元素序列。

3、指针：指针：地址。指针变量：存放其他类型变量地址的特殊变量。

4、结构体：由开发者自定义的包含多个数据类型的数据类型。

### 三、实验内容和要求

1、调试程序：输出100以内所有的素数（用函数实现）。

#include<stdio.h>

int isprime(int n){ /\*判断一个数是否为素数\*/

int m;

for(m=2;m\*m<=n;m++)

if( n%m ) return 0;

return 1;

}

int main(){ /\*输出100以内所有素数\*/

int i; printf("\n");

for(i=2;i<100;i++)

if(isprime(i)==1) printf("%4d",i);

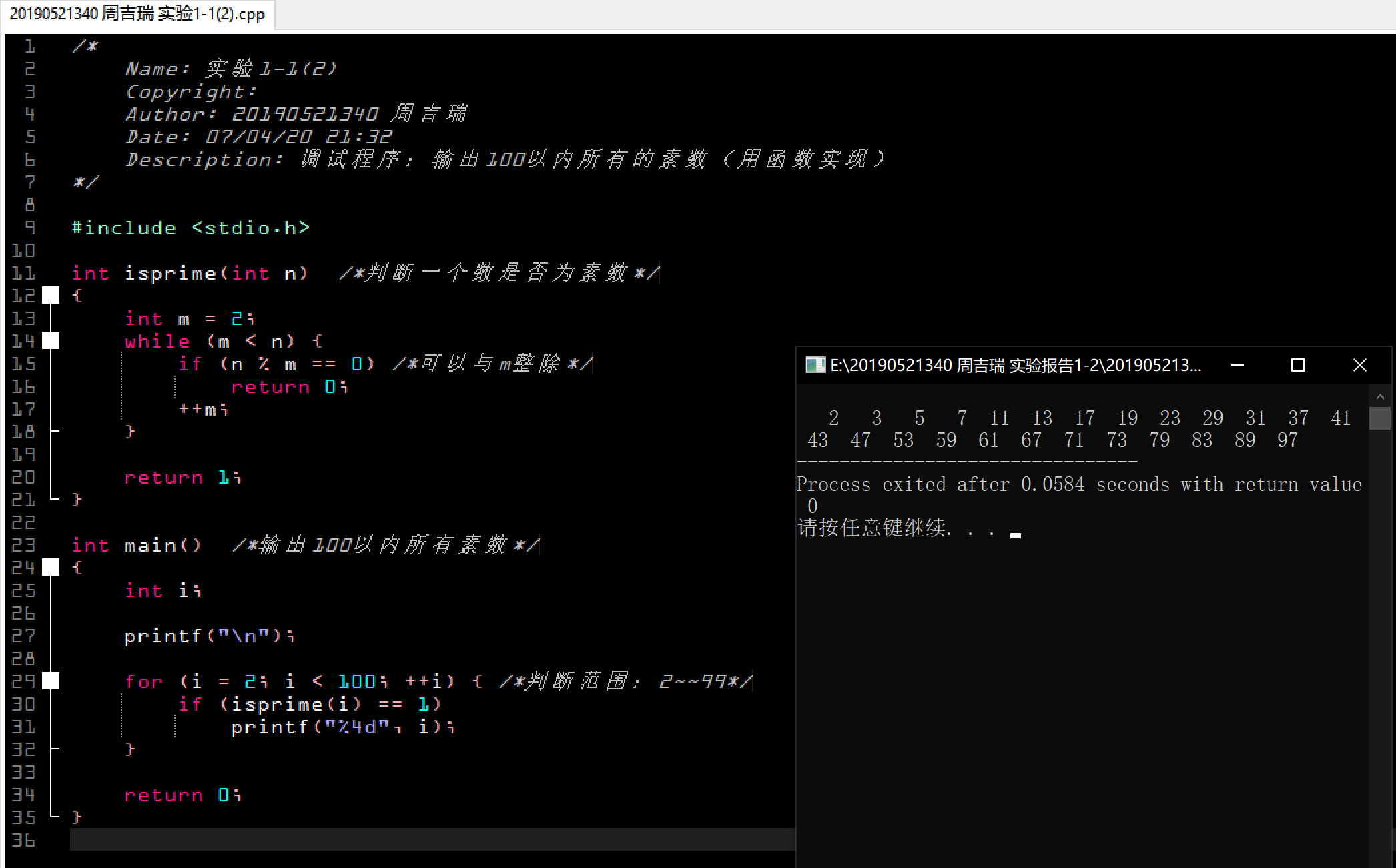
return 0;

}

**运行结果：**



方法2：改用while 语句：



2、 调试程序：对一维数组中的元素进行逆序排列。

#include<stdio.h>

#define N 10

int main(){

int a[N]={0,1,2,3,4,5,6,7,8,9},i,temp;

printf("\nthe original Array is:\n ");

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

printf("%4d",a[i]);

for(i=0;i< N ;i++){ /\*交换数组元素使之逆序\*/

temp=a[i];

a[i]=a[N-i-1];

a[N-i-1]=temp;

}

printf("\nthe changed Array is:\n");

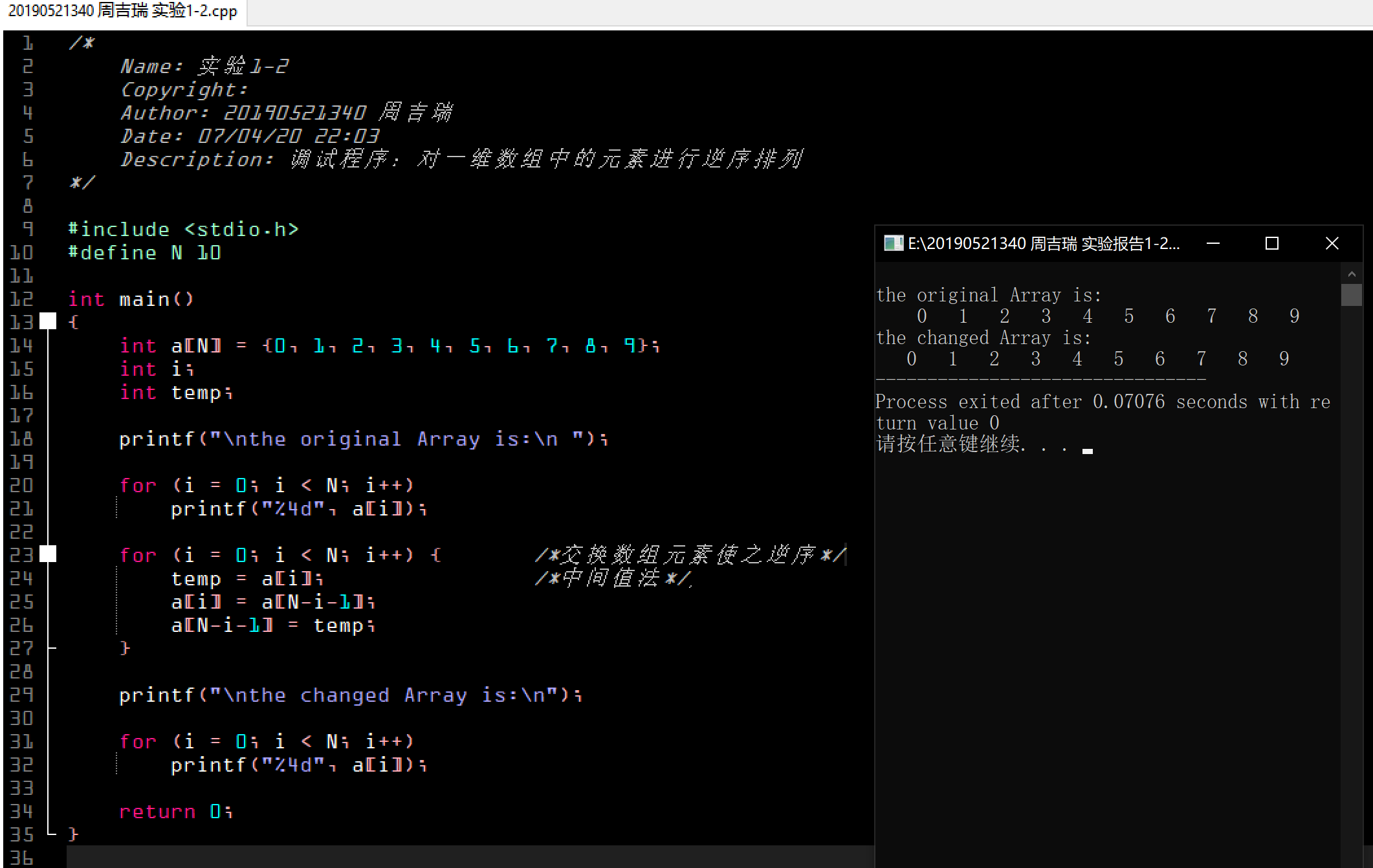
for(i=0;i<N;i++)

printf("%4d",a[i]);

return 0;

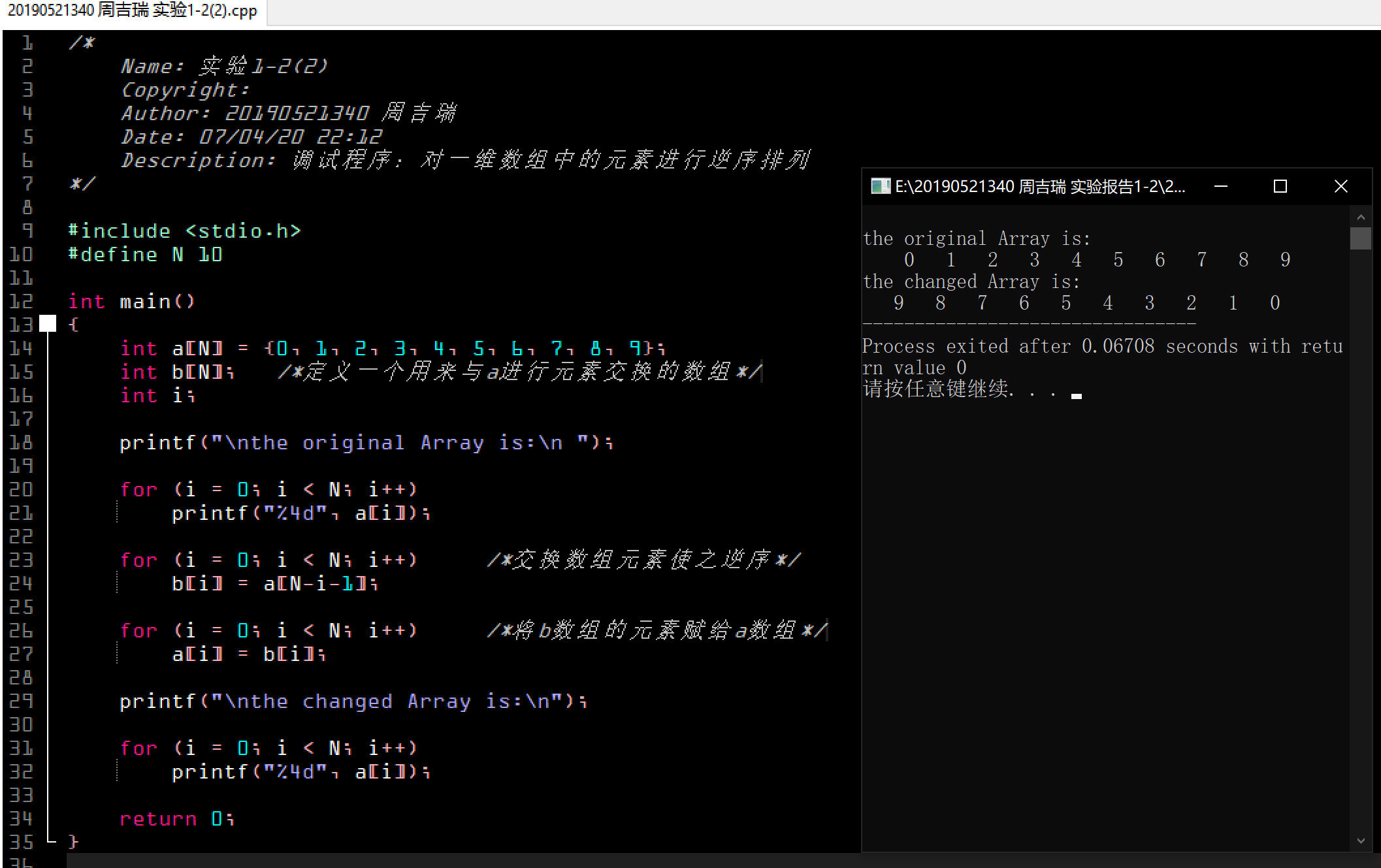
}

**运行结果：**



方法2：利用一个新数组进行交换

方法1比方法2更好：因为方法1只引入了一个变量作为中间变量，而方法2则是引入了一个数组作为中转站，从空间复杂的角度来看：方法1：O(1) 方法2：O(N)



3、 调试程序：在二维数组中，若某一位置上的元素在该行中最大，而在该列中最小，则该元素即为该二维数组的一个鞍点。要求从键盘上输入一个二维数组，当鞍点存在时，把鞍点找出来。

#include<stdio.h>

#define M 3

#define N 4

int main(){

int a[M][N],i,j,k;

printf("\n请输入二维数组的数据：\n");

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

for(j=0;j<N;j++)

scanf("%d",&a[i][j]);

for(i=0;i<M;i++){ /\*输出矩阵\*/

for(j=0;j<N;j++)

printf("%4d",a[i][j]);

printf("\n");

}

for(i=0;i<M;i++){

k=0;

for(j=1;j<N;j++) /\*找出第i行的最大值\*/

if(a[i][j]>a[i][k])

k=j;

for(j=0;j<M;j++) /\*判断第i行的最大值是否为该列的最小值\*/

if(a[j][k]<a[i][k])

break;

if(j== M ) /\*在第i行找到鞍点\*/

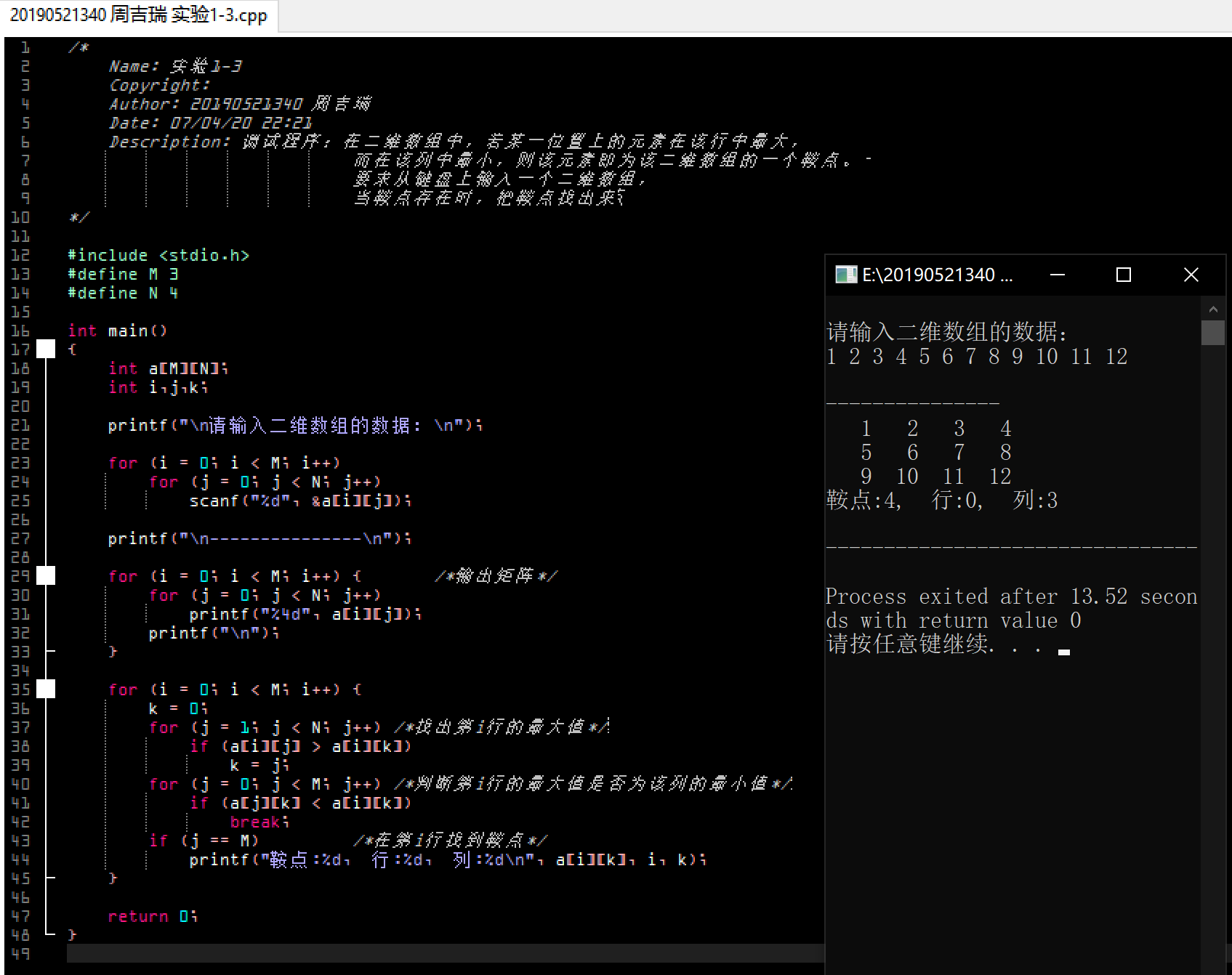
printf("%d,%d,%d\n",a[i][k],i,k);

}

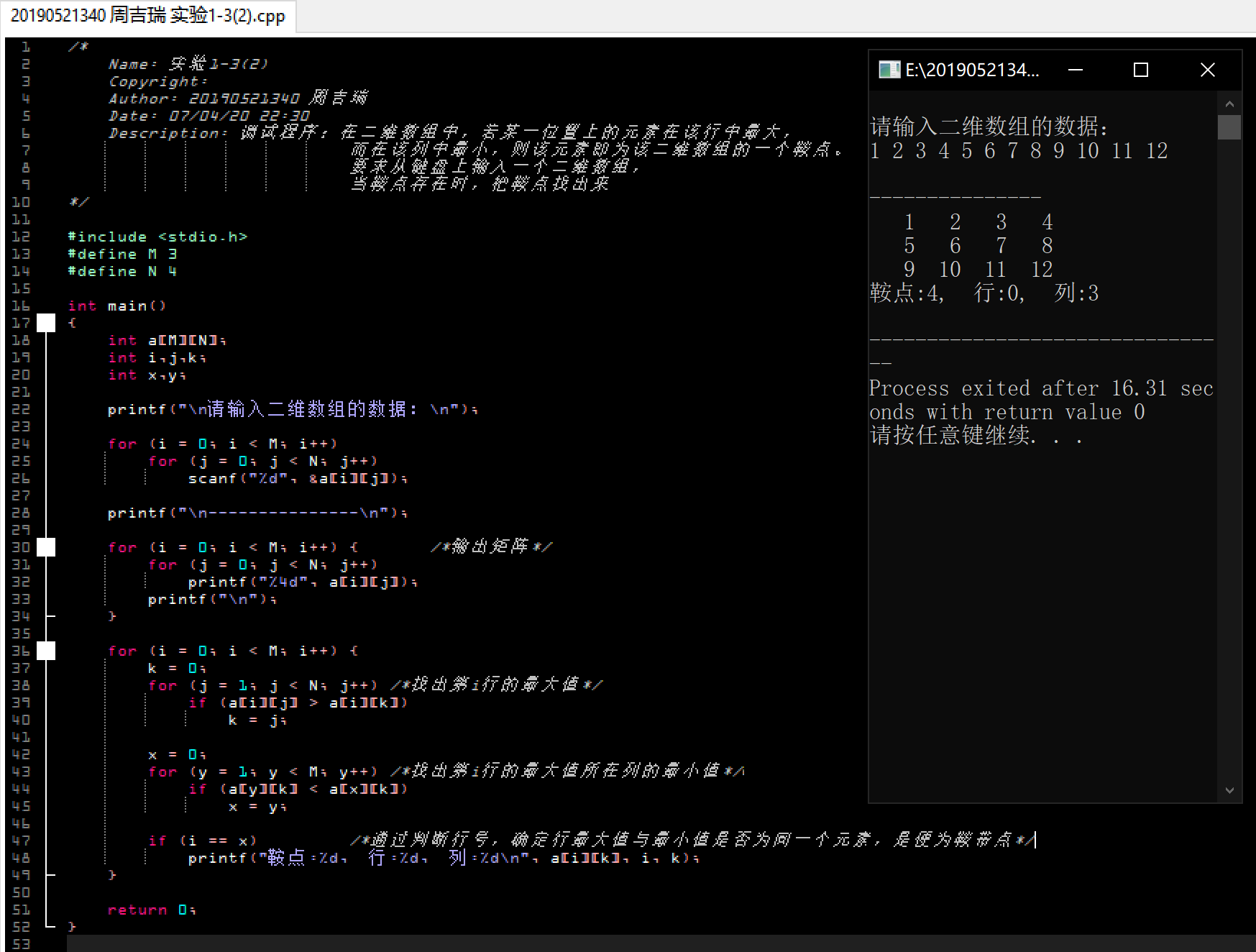
return 0;

}

**运行结果：**



方法2：对上面算法的改进，优化了算法的易读性



4、 调试程序：利用指针输出二维数组的元素。

#include<stdio.h>

int main(){

int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};

int \*p;

for(p=a[0];p<a[0]+12;p++){

if((p-a[0])%4==0) printf("\n");

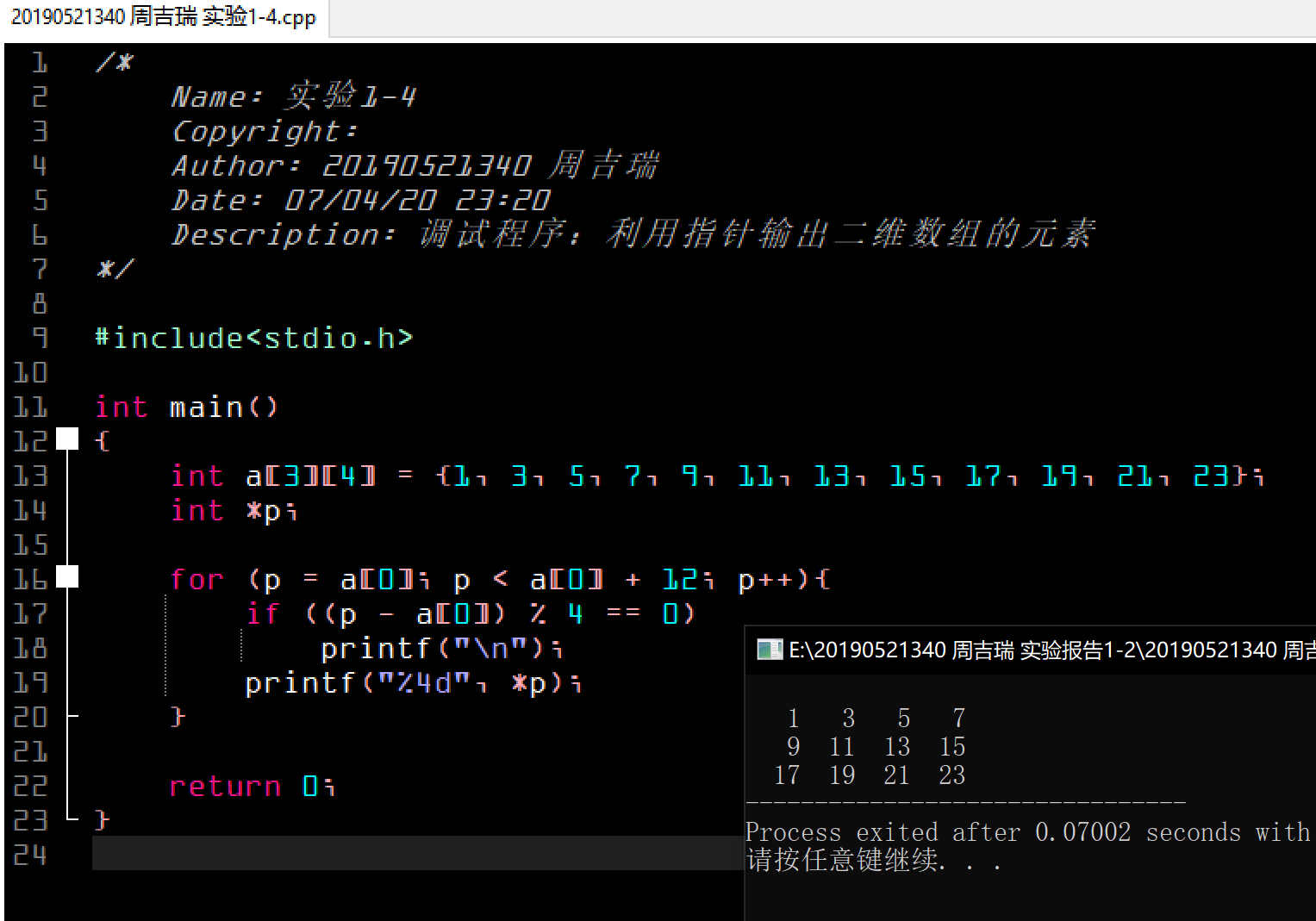
printf("%4d",\*p);

}

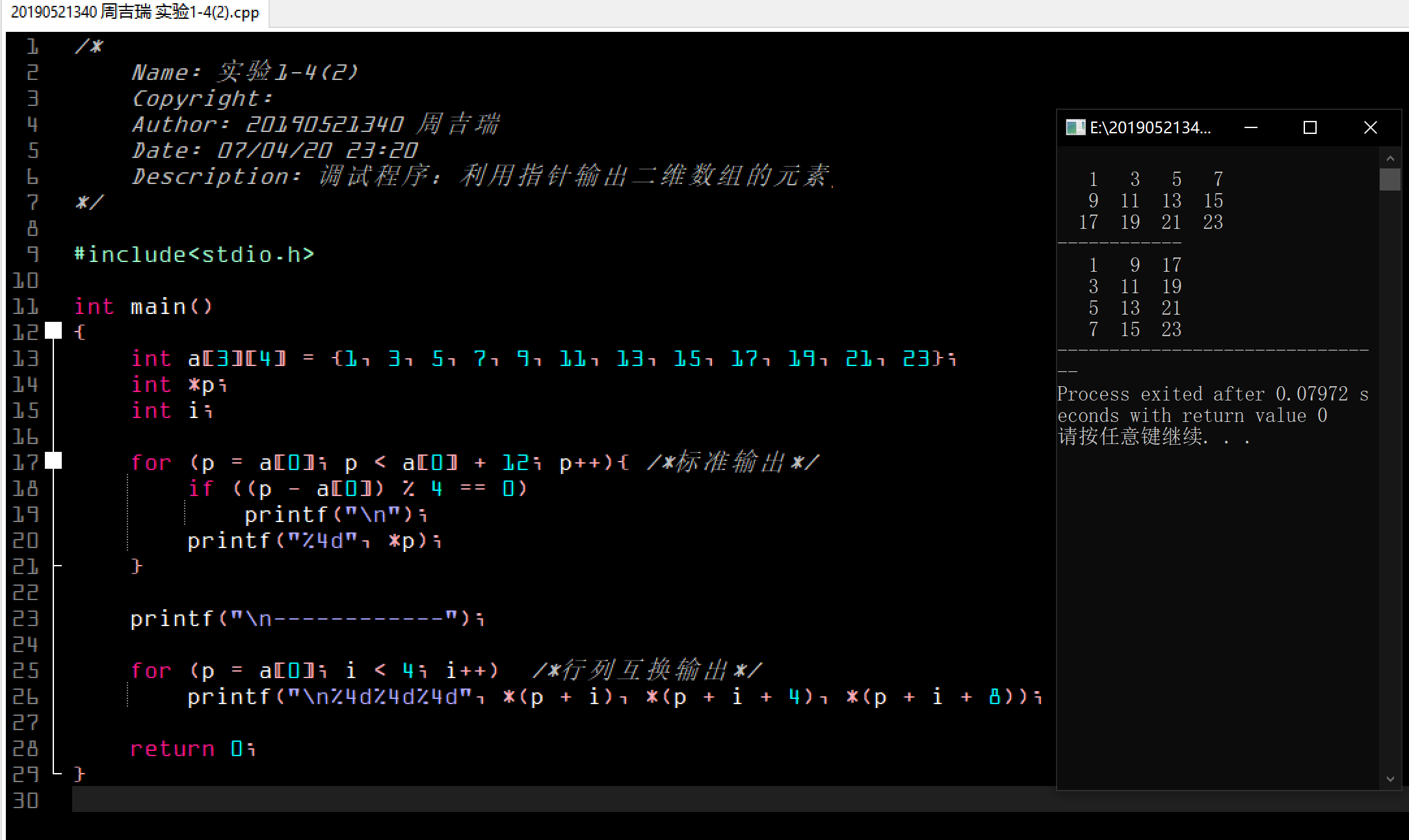
return 0;

}

**运行结果：**



补充：将二维数组行列互换输出



5、 调试程序：设有一个学生表格，有姓名、年龄、班级3项，编程输入人员的数据，再以表格输出。

#include <stdio.h>

#define N 10

**typedef** struct student{

char name[8]; /\*姓名\*/

int age; /\*年龄\*/

int class; /\*班级\*/

}PERSON;

int main(){

int i, n;

PERSON stu[N];

printf(“\n请输入人数(<10):\n”);

scanf(“%d”,&n);

for(i=0;i<n;i++){ /\*输入n个人员的信息\*/

printf("\n请输入第%d人员的信息：(name age class)\n",i+1);

scanf("%s%d%d",stu[i].name, &stu[i].age, &stu[i]. class);

}

printf(“name age class/office”);

for(i=0;i<n;i++){ /\*输出\*/

printf("%s is %3d years old. His class is %d\n",stu[i].name, stu[i].age, stu[i].class);

}

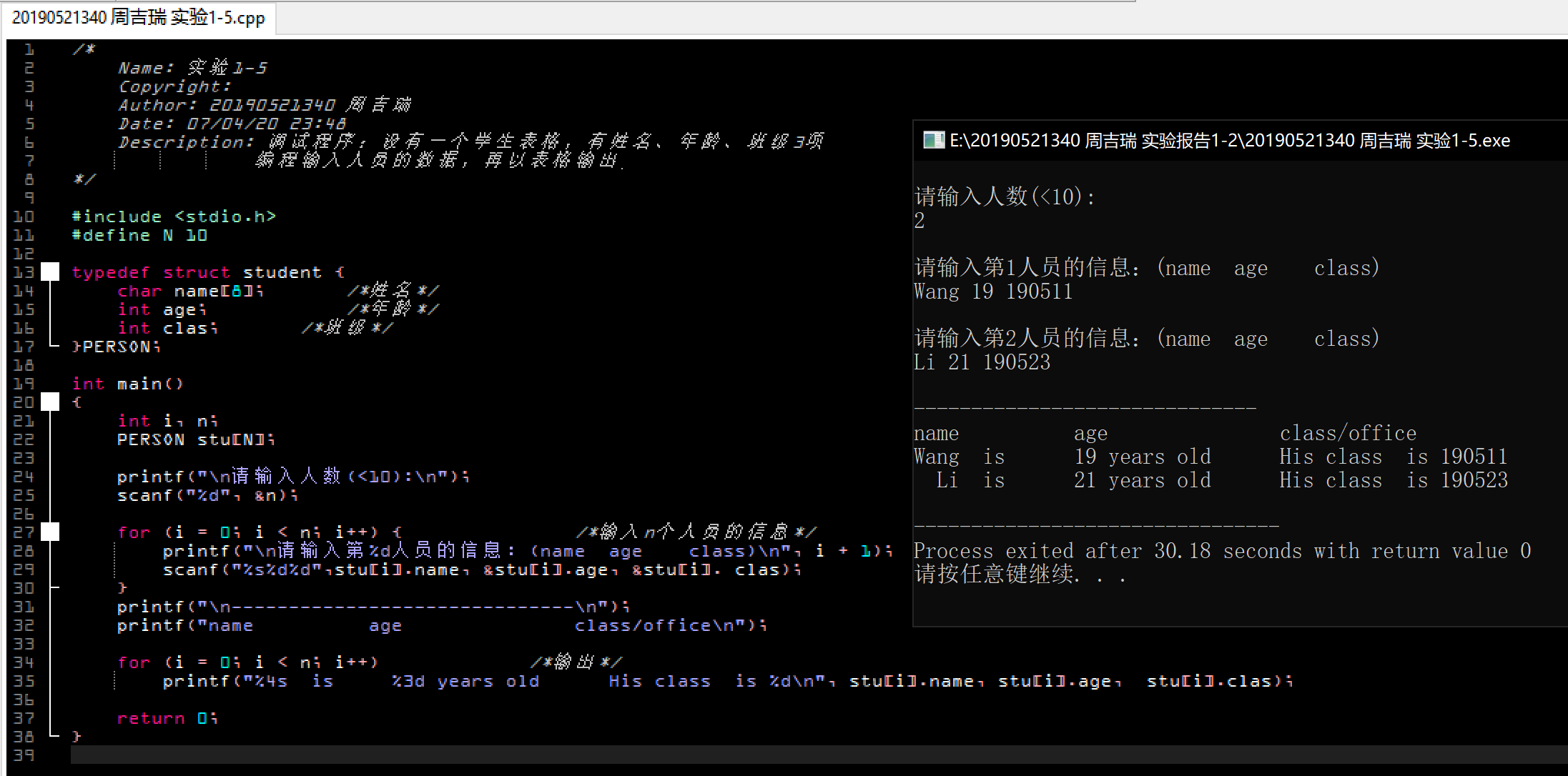
}

输入的数据：2

Wang 19 190511

Li 21 190523

**运行结果：**



对实验1-5的总结：

1. 在C++中：class是非法的变量名称，因为它是一个关键字或标识符（C语言中不存在影响）。

2.利用%4s比直接%s的其中一个好处在于可以利于对齐。

### 四、实验小结

C语言中的概念通俗理解：

1.函数：函数是一个自我包含的完成一定相关功能的执行代码段。

2.数组：所谓数组,是有序的元素序列。

3.指针：指针：地址。指针变量：存放其他类型变量地址的特殊变量。

4.结构体：由开发者自定义的包含多个数据类型的数据类型。

对于编程开发而言：

1. 应该注重算法的效率、易读性和健壮性。

2.应模块化（分函数分文件）进行功能的封装操作，这样方便于操作也利于管理及优化，甚至利于代码实现内容的封装。

3.编写程序，代码应该规范，自定义的名称要合法且有一定意义，要通俗易懂的写注释。

4.一个好的程序最终的实现不可能一次性就完成，应该按照一般的开发流程，一步一步推进，不断优化维护。

、

# 实验2 线性表及其应用

**实验日期**：2020年 4月 8日---- **报告日期**： 2020年 4月 10日

### 一、实验目的

1、掌握线性表中元素的前驱、后续的概念。

2、掌握顺序表与链表的建立、插入元素、删除表中某元素的算法。

3、对线性表相应算法的时间复杂度进行分析。

4、理解顺序表、链表数据结构的特点（优缺点）。

### 二、实验预习

复习以下概念

1、线性表：n个数据特性相同的元素构成的有限序列称为线性表。线性表中元素的个数n定义为线性表的长度,n等于零时，称为空表。除第一个元素之外，结构中的每个数据元素均只有一个前驱。除最后一个元素之外，结构中的每个数据元素均只有一个后继。

2、顺序表线性表的顺序表示指的是用一组地址连续的存储单元依次存储线性表的数据元素。这种表示也称作线性表的顺序存储结构或顺序映像，通常这种存储结构的线性表为顺序表。其特点是逻辑上相邻的数据元素，其物理次序也是相邻的。：

3、链表：用一组任意的存储单元。存储线性表的数据元素。这组存储单元可以是连续的，也可以是不连续的。因此为了表示每个数据元素与其直接后继数据元素之间的逻辑关系。对数据元素来说，除了存储其本身信息之外，还需存储一个指示其直接后继的信息。即：直接后继的存储位置，这两部分信息组成数据元素的存储映像称为结点。它包括两个域，其中存储数据元素的域成为称为数据域，存储直接后继存储位置的域称为指针域，支撑一种存储的信息称为指针或链。n个节点链结成一个链表,即：线性表。

### 三、实验内容和要求

1、阅读下面程序，在横线处填写欠缺内容，并**补充函数的基本功能注**释。运行程序，写出结果（运行截图）。

#include<stdio.h>

#include<malloc.h>

#define ERROR 0

#define OVERFLOW -2

#define OK 1

#define MAXSIZE 100 /\*初始分配的顺序表长度\*/

typedef int ElemType; /\*定义表元素的类型\*/

typedef  **int**  Status;

typedef struct {

ElemType \*elem; //存储空间的基地址

int length; //当前长度

} Sqlist;

int InitList\_sq(Sqlist &L); /\*顺序表的初始化(构造一个空的顺序表L)\*/

int CreateList\_sq(Sqlist &L,int n); /\* 创建顺序表(顺序输入数据元素) \*/

int ListInsert\_sq(Sqlist &L,int i,ElemType e);/\* 顺序表的第i个位置之前插入元素e \*/

int PrintList\_sq(Sqlist &L); /\*输出顺序表的元素\*/

int ListDelete\_sq(Sqlist &L,int i); /\*删除第i个元素\*/

int ListLocate(Sqlist &L,ElemType e); /\*查找值为e的元素\*/

Status InitList\_sq(SqList &L) //算法2.1 顺序表的初始化(构造一个空的顺序表L)

{

L.elem=(ElemType\*)malloc(MAXSIZE\*sizeof(ElemType));

//为顺序表分配一个大小为MAXSIZE的数组空间

if (!L.elem)

exit(OVERFLOW); //存储分配失败退出

L.length = 0 ; //空表长度为0

return OK;

}/\*InitList\*/

int CreateList\_sq(Sqlist &L,int n){

ElemType e;

int i;

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

printf("input data %d: ",i+1);

scanf("%d",&e);

if(!ListInsert\_sq(L, **i+1** ,e))

return ERROR;

}

return OK;

}/\*CreateList\*/

/\*输出顺序表中的元素\*/

int PrintList\_sq(Sqlist L){

int i;

for(i=1;i<=L.length;i++)

printf("%5d", L.elem[i - 1] );

return OK;

}/\*PrintList\*/

int ListInsert\_sq(Sqlist &L,int i,ElemType e){

int k;

if(i<1||i>L.length+1)

return ERROR;

if(L.length>=MAXSIZE){

return ERROR;

}

for(k=L.length-1;k>=i-1;k--){

L.elem[ **k + 1** ]= L.elem[ **k** ];

}

L.elem[i-1]=e;

++L.length ;

return OK;

}/\*ListInsert\*/

/\*在顺序表中删除第i个元素,并用e返回其值，i值的合法范围是1<=i<=L.length \*/

Status ListDelete\_Sq(Sqlist &L, int i,ElemType &e) {

if ((i < 1) || (i > L.length))

return ERROR;

e = L.elem[i - 1];

for (int j = i; j <= L.length; j++)

L.elem[j - 1] = L.elem[j];

--L.length;

return OK;

}

/\*在顺序表中查找指定值元素，返回其序号\*/

int ListLocate(Sqlist &L,ElemType e){

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

if (L.elem[i] == e)

return i + 1;

return 0;

}

int main(){

SqList sl;

int n,m,k;

printf("please input n:"); /\*输入顺序表的元素个数\*/

scanf("%d",&n);

if(n>0){

printf("\n1-Create SqList:\n");

InitList\_sq(sl);

CreateList\_sq(sl,n);

printf("\n2-Print SqList:\n");

PrintList\_sq(sl);

printf("\nplease input insert location and data:(location,data)\n");

scanf("%d,%d",&m,&k);

ListInsert\_sq(sl,m,k);

printf("\n3-Print SqList:\n");

PrintList\_sq(sl);

printf("\n");

}

else

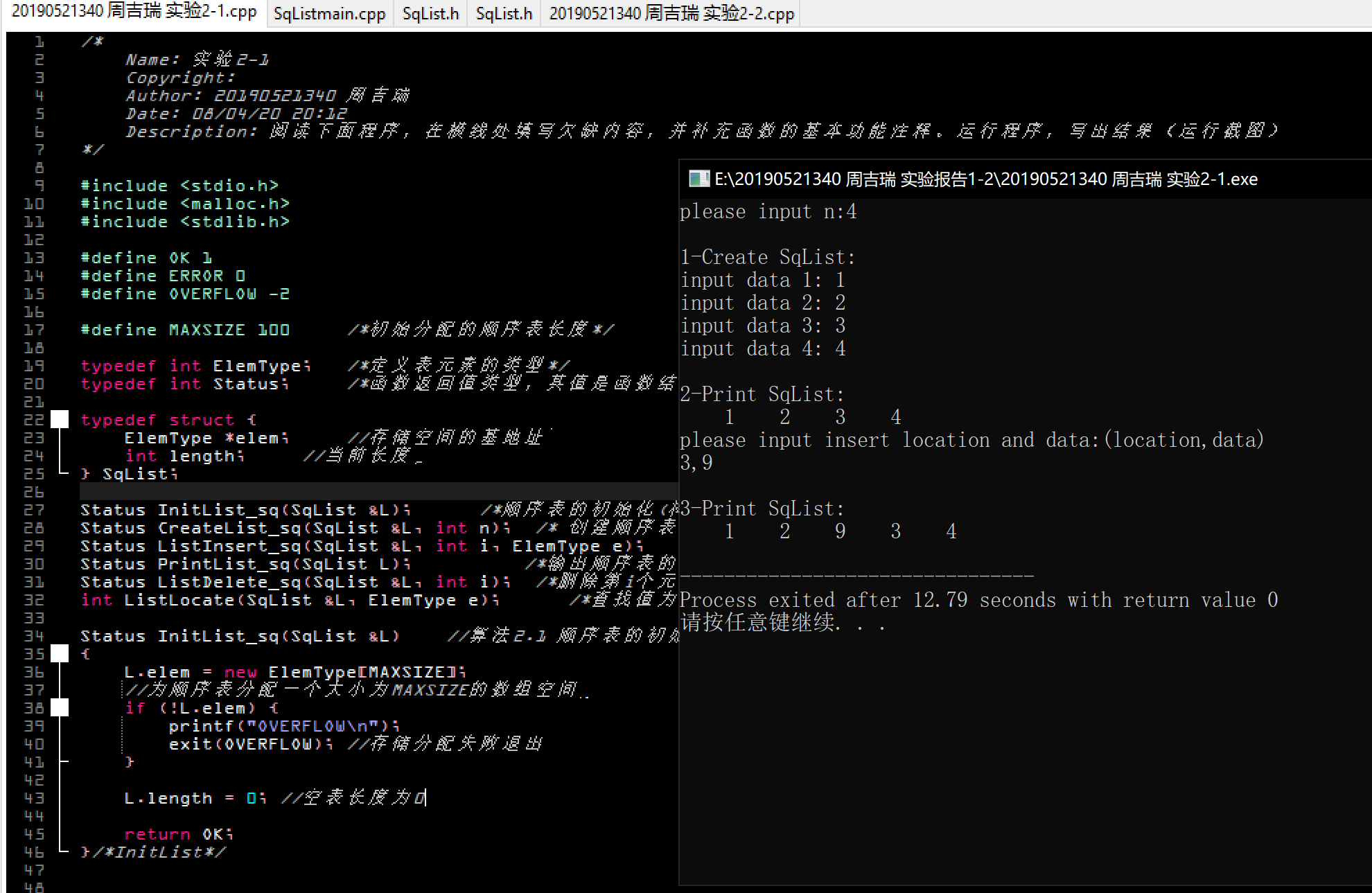
printf("ERROR");

return 0;

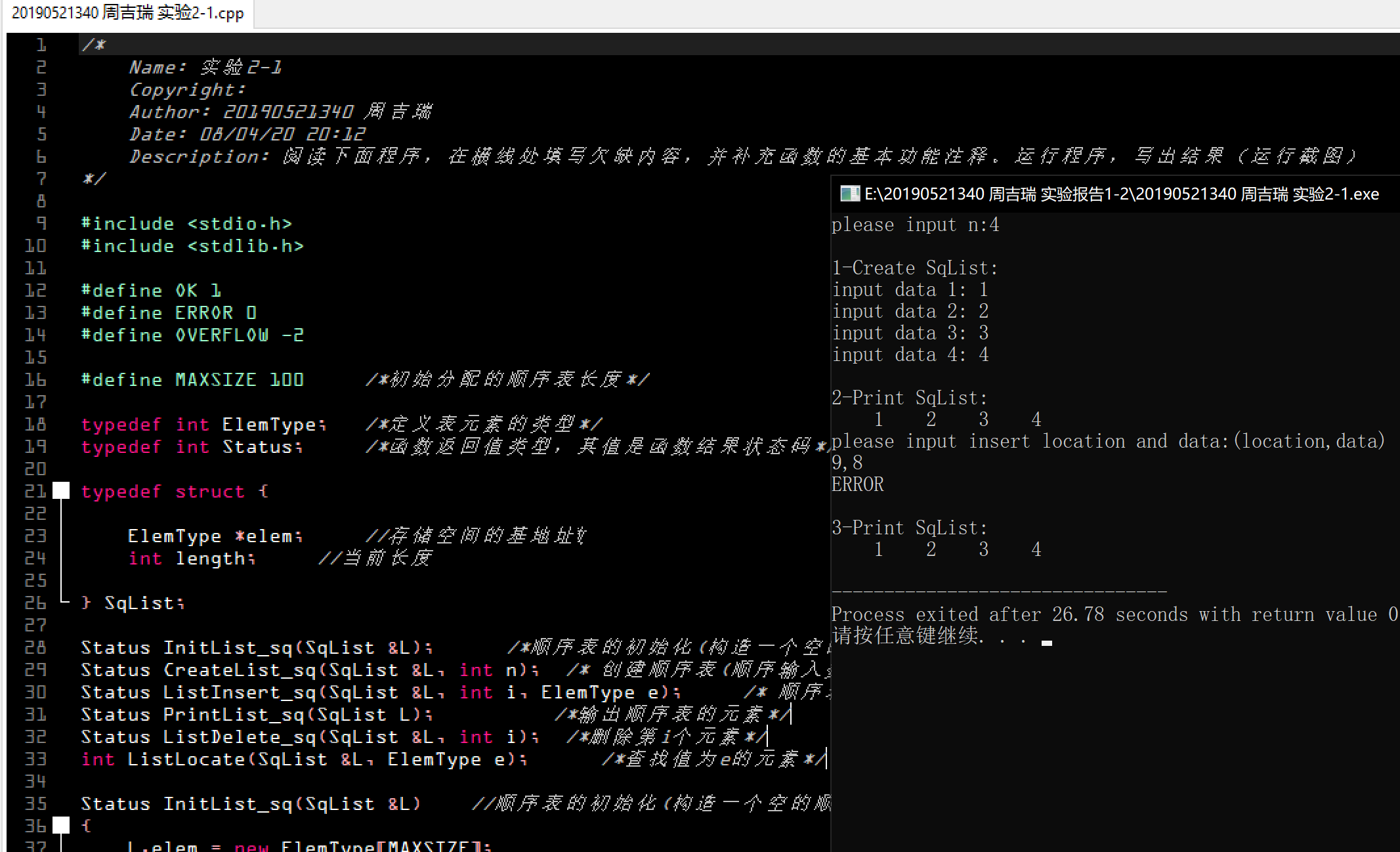
}

* 运行结果

1.正确输入时：



2.非法输入时：



* 算法分析

取值算法：

根据顺序表随机存取的特点

故：删除算法的时间复杂度 T(n) = O(n)

插入算法：

在顺序表中某个位置上插入一个数据源说是。其时间主要消耗在移动源说上，而移动元素的个数取决于插入元素的位置。

执行范围：1~n 平均执行：n / 2

根据“抓大放小”原则，取：f(n) = n

故：插入算法的时间复杂度 T(n) = O(n)

2、为第1题补充删除和查找功能函数，并在主函数中补充代码，验证这两个函数的正确性。

删除算法代码：

/\*在顺序表中删除第i个元素,并用e返回其值，i值的合法范围是1<=i<=L.length \*/

Status ListDelete\_Sq(Sqlist &L, int i,ElemType &e) {

if ((i < 1) || (i > L.length)) {

printf("ERROR\n");

return ERROR; //i值不合法

}

e = L.elem[i - 1];

for (int j = i; j <= L.length; j++)

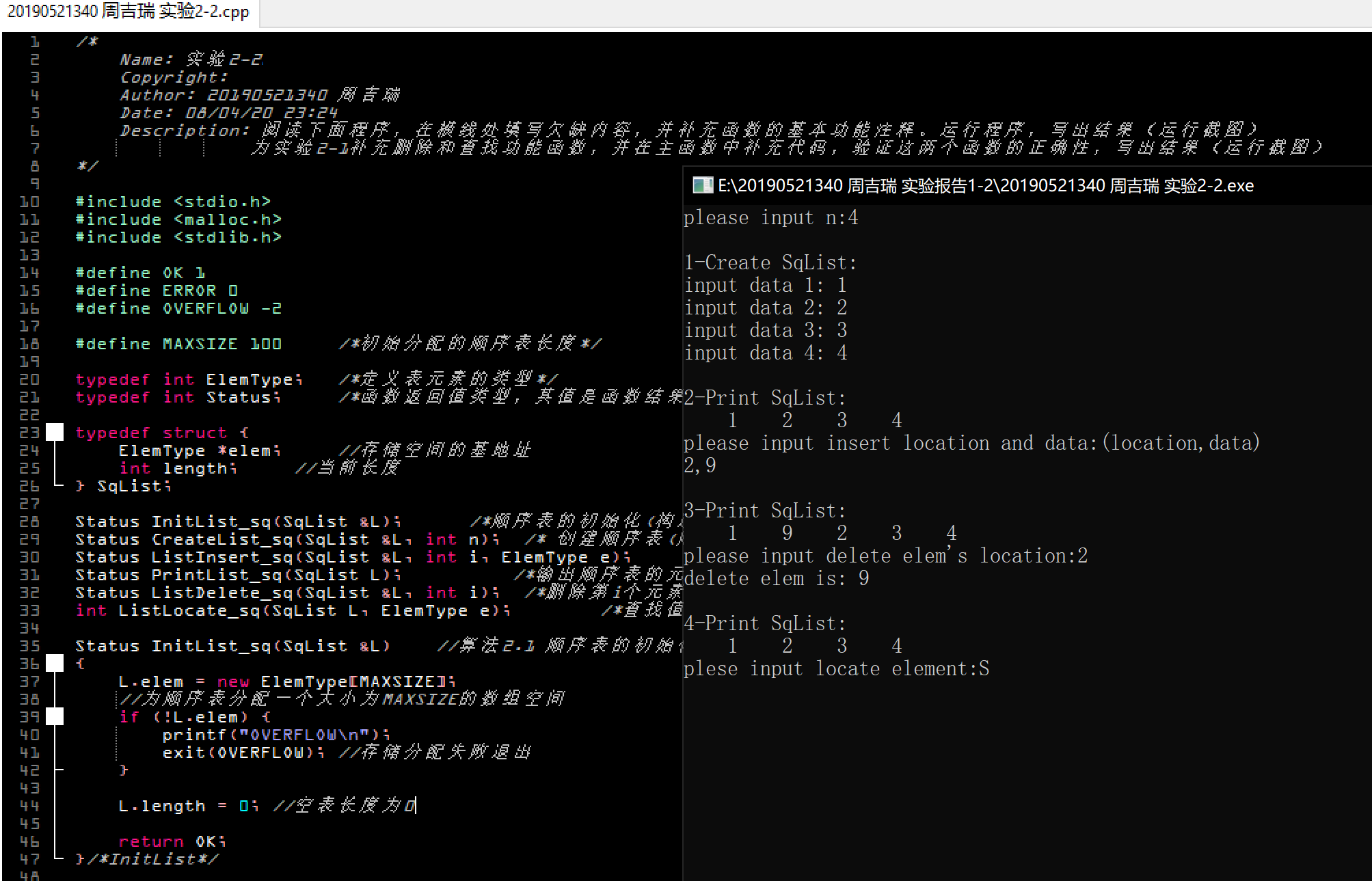
L.elem[j-1] = L.elem[j]; //被删除元素之后的元素前移

--L.length; //表长减1

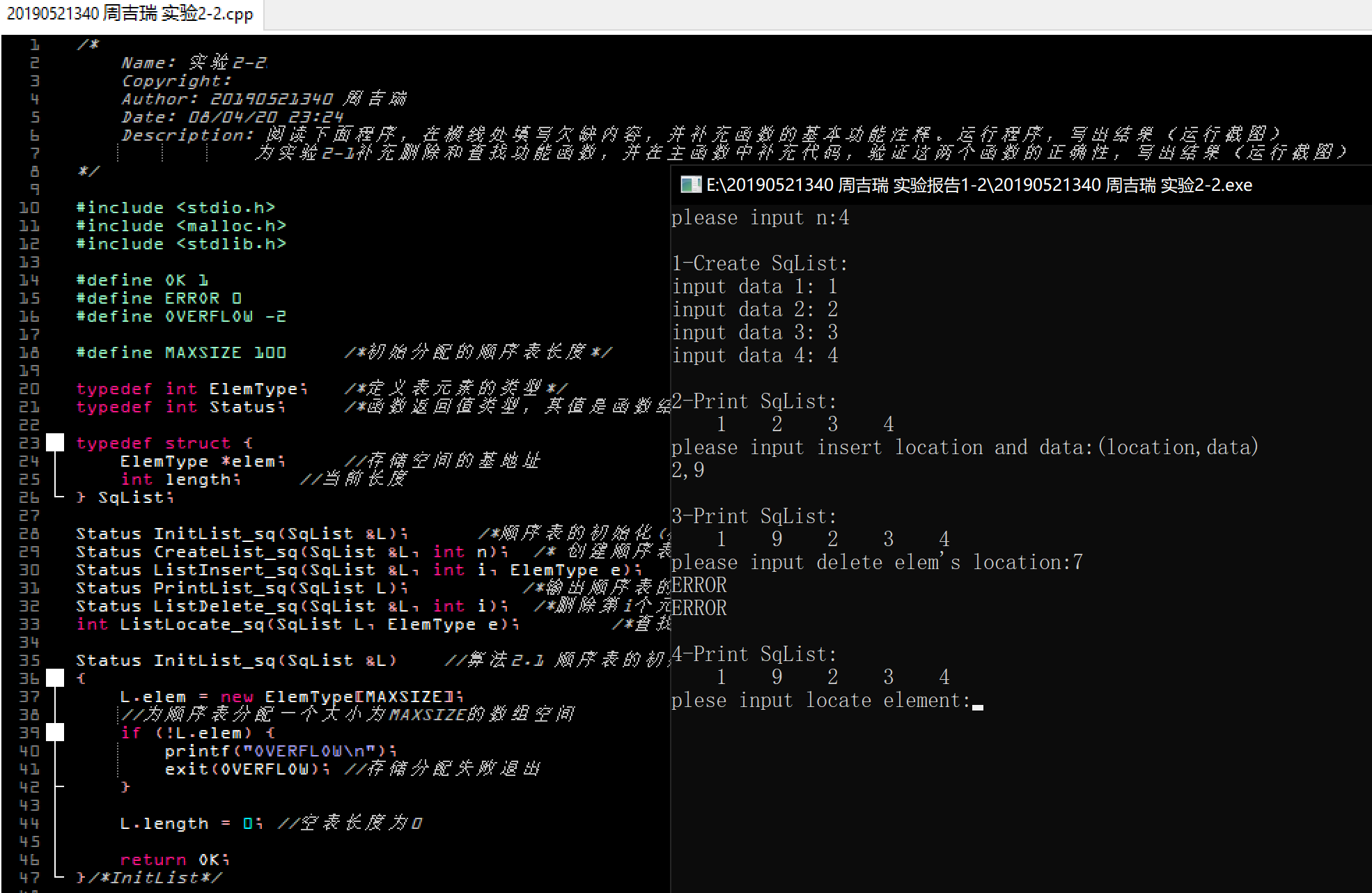
return OK; }

* 运行结果

1.正确输入时：



2.非法输入时：



* 算法分析

当在顺序表中某个位置上删除一个数据元素时，时间主要消耗在移动元素上，把移动元素的个数取决于删除元素的位置。

执行范围：1~n 平均执行：n / 2

f(n) = n / 2

根据“抓大放小”原则，取：f(n) = n

故：删除算法的时间复杂度 T(n) = O(n)

查找算法代码：

/\*在顺序表中查找指定值元素，返回其序号\*/

int ListLocate(Sqlist &L,ElemType e){

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

if (L.elem[i] == e)

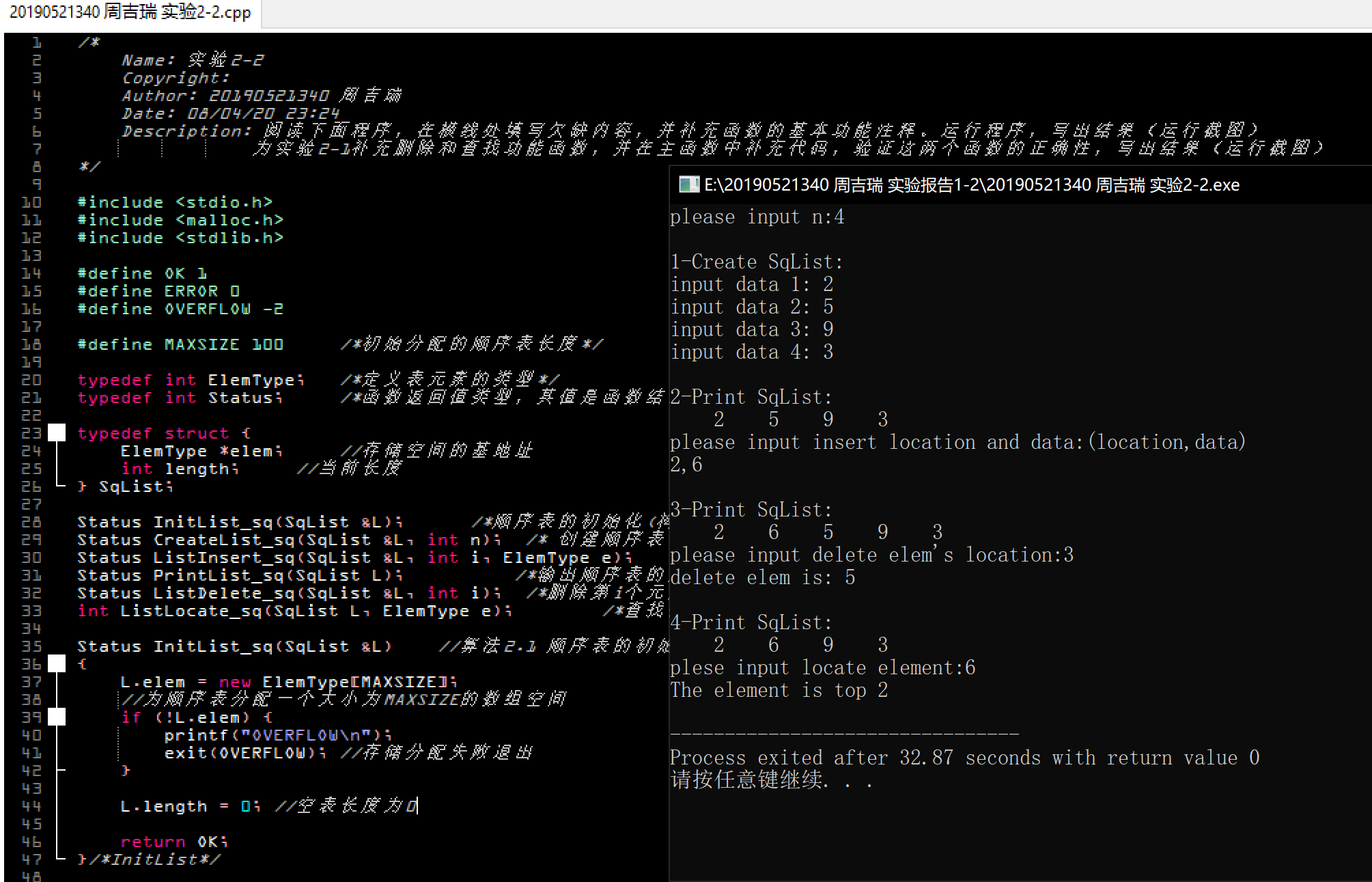
return i + 1; //查找成功，返回序号i+1

printf("There is no the element.\n") ;

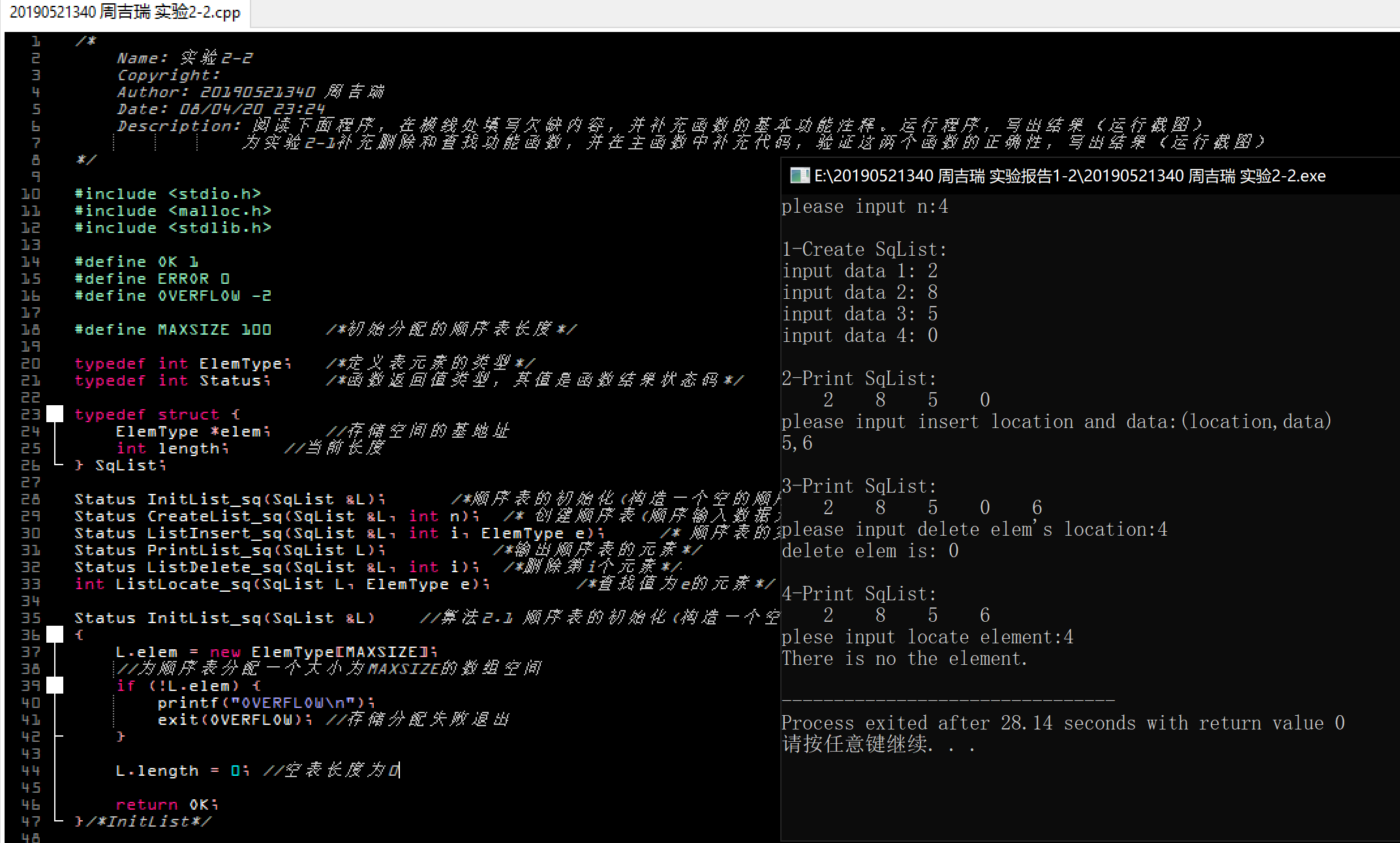
return 0; //查找失败，返回0 }

* 运行结果

1.正确输入时：



2.非法输入时：



* 算法分析

当在顺序表中查找一个数据元素，实际时间主要消耗在数据的比较上。而比较的次数取决于被查元素在线性表中的位置。

执行范围：1~n次，平均执行：n / 2次

f(n) = n / 2

根据“抓大放小”原则，取：f(n) = n

故：查找算法的时间复杂度 T(n) = O(n)

3、阅读下面程序，在横线处填写欠缺内容，并补充函数的基本功能注释。运行程序，写出结果（运行截图）。

#include<stdio.h>

#include<malloc.h>

#define ERROR 0

#define OK 1

typedef int ElemType; /\*定义表元素的类型\*/

typedef int Status ;

typedef struct LNode{ /\*线性表的循环单链表存储\*/

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

LinkList CreateList(int n); /\*建立带头结点的循环单链表\*/

void PrintList(LinkList L); /\* 顺序输出单链表 \*/

int GetElem(LinkList L,int i,ElemType &e); /\* 取出单链表第i个值并用e返回其值\*/

LinkList CreateList(int n){

LNode \*p,\*q,\*head;

int i;

head=(LinkList)malloc(sizeof(LNode));

head->next=head;

p= head ;

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

q=(LinkList)malloc(sizeof(LNode));

printf("input data %i:",i+1);

scanf("%d", &(q->date) ); /\*输入元素值\*/

p->next=q; /\*新结点连在表末尾\*/

p=q;

}

q->next= head ;

return head;

}/\*CreateList\*/

void PrintList(LinkList L){

LNode \*p;

p=L->next; /\*p指向单链表的第1个元素\*/

while(p!= L ){

printf("%5d %d %d\n",p,p->data,p->next);

p = p->next ;

}

}/\*PrintList\*/

int GetElem(LinkList L,int i,ElemType &e){

LNode \*p;int j=1;

p=L->next;

while(p!=L &&j<i){

p=p->next;j++;

}

if( p == L || j > i )

return ERROR;

e= p->date ;

return OK;

}/\*GetElem\*/

int main(){

int n,i;ElemType e;

LinkList L=NULL; /\*定义指向单链表的指针\*/

printf("please input n:"); /\*输入单链表的元素个数\*/

scanf("%d",&n);

if(n>0){

printf("\n1-Create LinkList:\n");

L=CreateList(n);

printf("\n2-Print LinkList:\n");

PrintList(L);

printf("\n3-GetElem from LinkList:\n");

printf("input i=");

scanf("%d",&i);

if(GetElem( L, i, e ))

printf("No%i is %d",i,e);

else

printf("not exists");

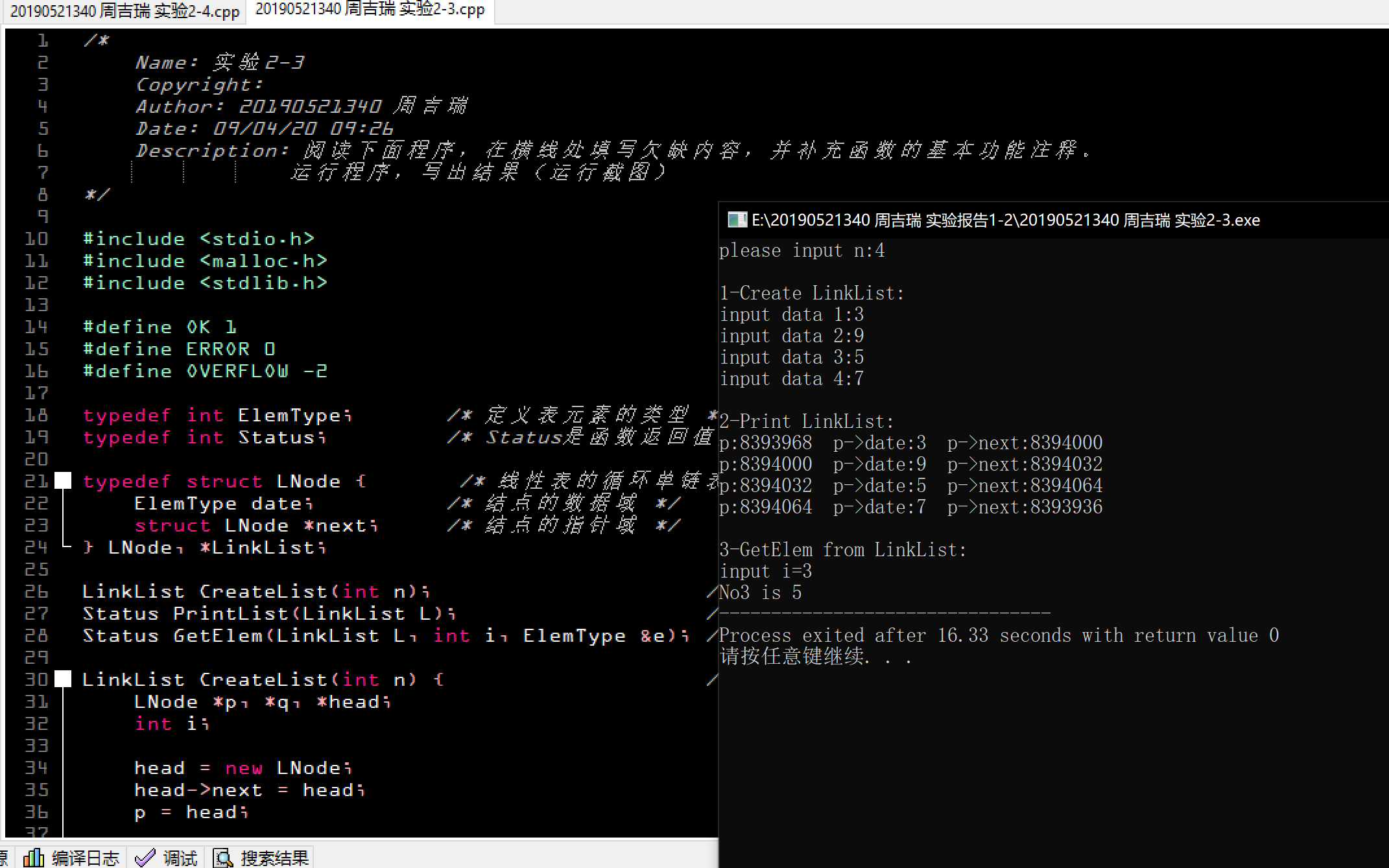
}else

printf("ERROR");

return 0;

}

* 运行结果



* 算法分析

创建算法：

算法的执行次数取决于须要输入的元素个数n

故：时间复杂度：T(n) = O(n)

取值算法：

算法的执行次数与所需要取位置i有关。

执行范围：1~n 平均执行：n / 2

f(n) = n / 2

根据“抓大放小”原则，取：f(n) = n

故：取值算法的时间复杂度 T(n) = O(n)

查找算法：

该算法的执行时间与待查找的值e相关。其平均时间复杂度分析类似于取值算法

故：查找算法的时间复杂度 T(n) = O(n)

4、为第3题补充插入功能函数和删除功能函数。并在主函数中补充代码,验证这两个函数的正确性。

插入算法代码：

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

//单链表的插入:在带头结点的单链表L中第i个位置插入值为e的新结点

{

int j;

LNode \*p, \*s;

p = L;

j = 0;

while (j < i - 1) {

p = p->next; /\* 查找第i-1个结点，p指向该结点 \*/

++j;

}

if (p == L || j > i - 1) {

printf("ERROR\n");

return ERROR;

}

s = new LNode; /\* 生成新结点\*s \*/

s->data = e; /\* 将结点\*s的数据域置位e \*/

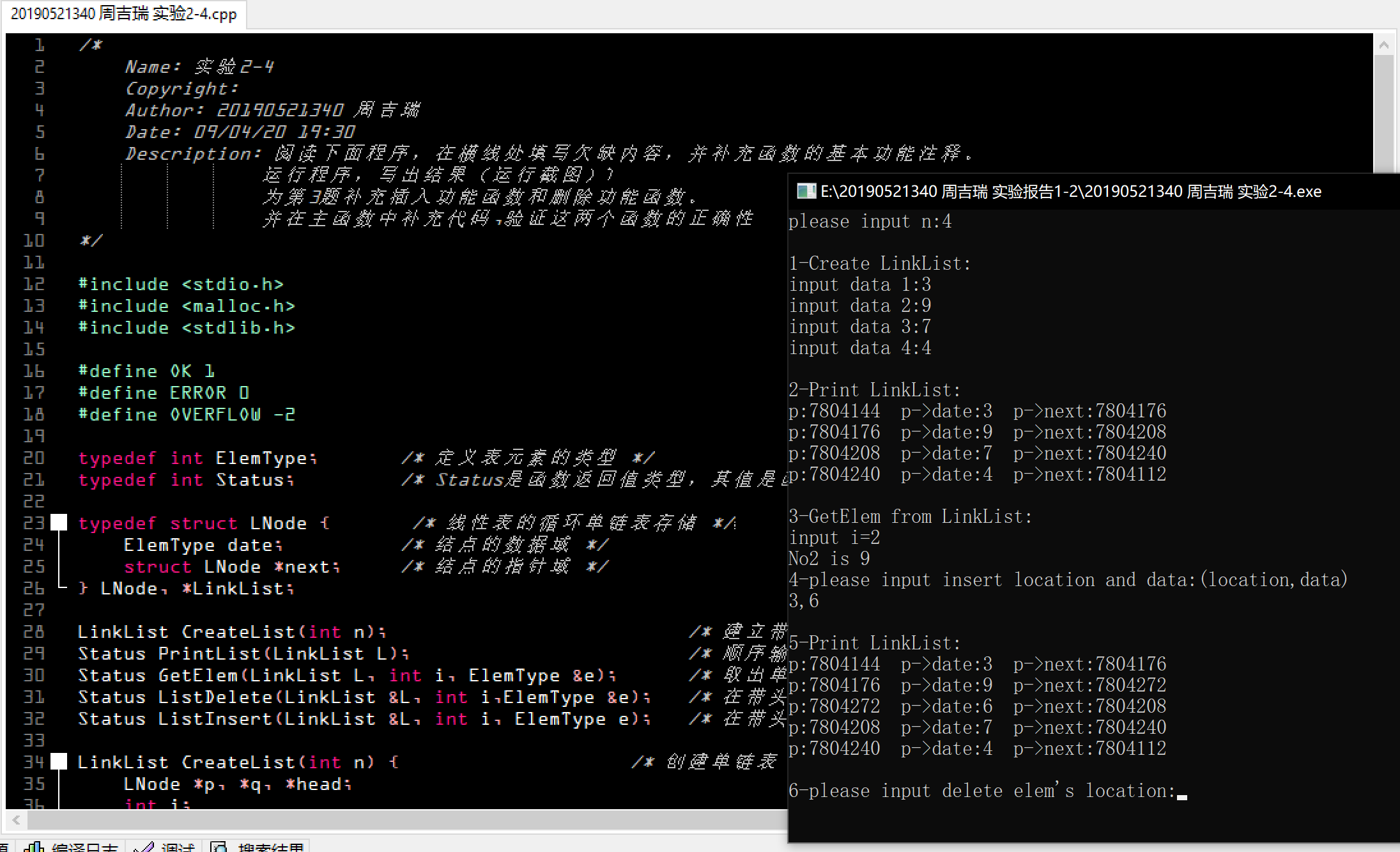
s->next = p->next; /\* 将结点\*s的指针域指向结点ai \*/

p->next = s; /\* 将结点\*p的指针域指向结点\*s \*/

return OK;

}

* 运行结果



* 算法分析

虽然单链表的插入操作不需要像顺序表的插入操作那样需要移动元素。但插入的前提还是需要对单链表进行查找。所以本质上单链表的插入算法与单链表的查找算法时间复杂度相同。

故：插入算法的时间复杂度 T(n) = O(n)

删除算法代码：

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

// 单链表的删除:在带头结点的单链表L中，删除第i个位置,并将删除元素的数据存入e

{

LNode \*p, \*q;

int j;

p = L;

j = 0;

while (j < i - 1) { /\* 查找第i-1个结点，p指向该结点 \*/

p = p->next;

++j;

}

if (p == L || (j > i - 1)) { /\* i值不合法 \*/

printf("ERROR\n");

return ERROR;

}

q = p->next; /\* 临时保存被删结点的地址以备释放 \*/

p->next = q->next; /\* 改变删除结点前驱结点的指针域 \*/

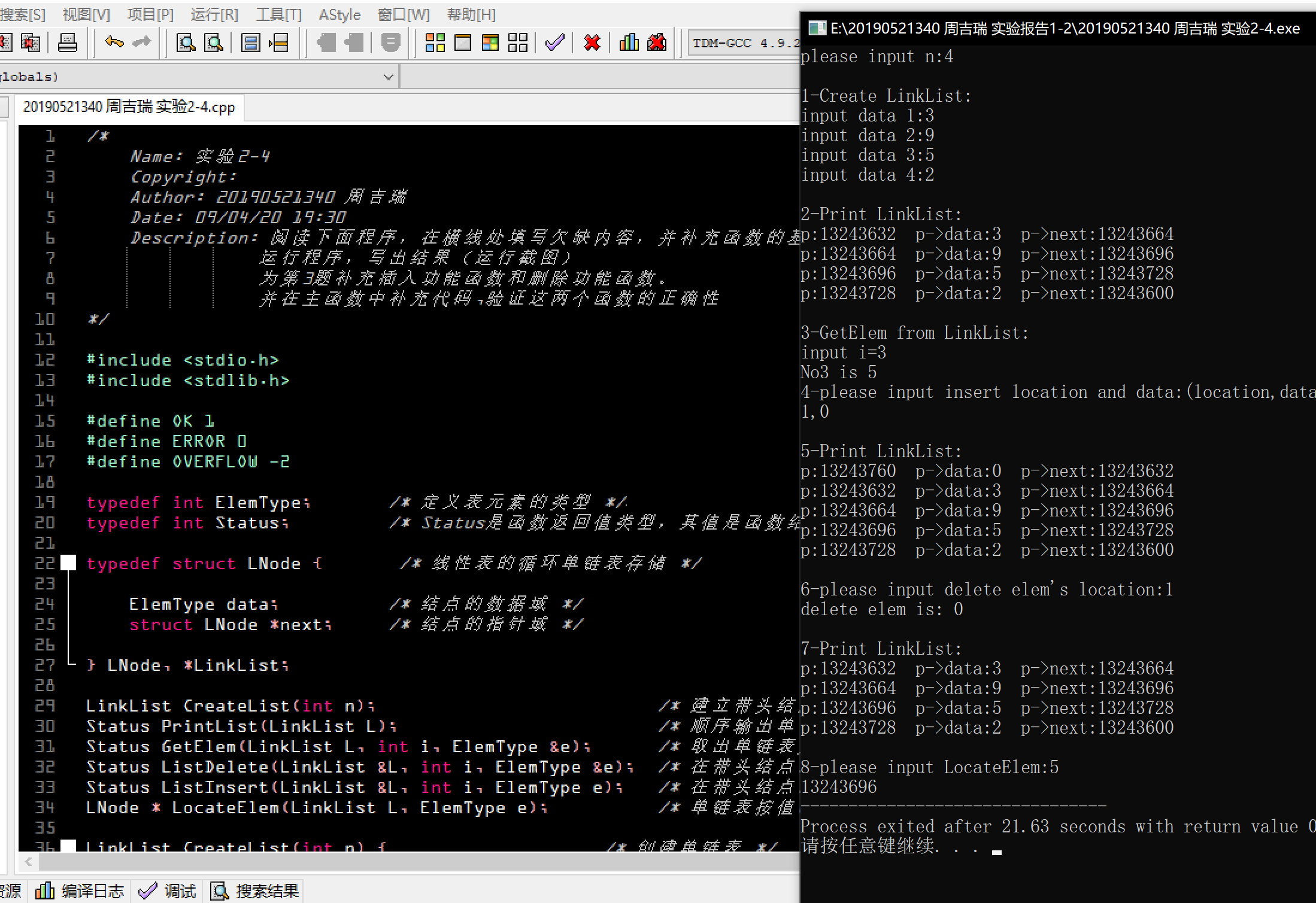
e = q->data; /\* 用e保存被删结点数据域 \*/

delete q; /\* 释放删除结点的空间 \*/

return OK;

}

* 运行结果



注：程序最后补充了，按值查找算法

* 算法分析

单链表的删除算法类似于插入算法，

故：删除算法时间复杂度T(n) = O(n)

以下为**选做实验**：

**\*5.参看教师的演示资料，任选一种存储结构，实现集合的并、交、差运算。（可只完成其中一种）**

* **算法代码**

/\*

Name: 实验2-5

Copyright:

Author: 20190521340 周吉瑞

Date: 10/04/20 14:01

Description: 利用顺序表实现两个集合AB的并集（A = A U B)

\*/

#include <stdio.h>

#include <stdlib.h>

#define OK 1

#define ERROR 0

#define OVERFLOW -2

#define MAXSIZE 100 /\*初始分配的顺序表长度\*/

typedef int ElemType; /\*定义表元素的类型\*/

typedef int Status; /\*函数返回值类型，其值是函数结果状态码\*/

typedef struct {

ElemType \*elem; //存储空间的基地址

int length; //当前长度

} SqList;

Status InitList\_sq(SqList &L); /\*顺序表的初始化(构造一个空的顺序表L)\*/

Status CreateList\_sq(SqList &L, int n); /\* 创建顺序表(顺序输入数据元素) \*/

Status PrintList\_sq(SqList L); /\*输出顺序表的元素\*/

Status MergeList\_sq(SqList &LA, SqList LB, int n, int m); /\* 线性表LA LB的合并 \*/

Status ListInsert\_sq(SqList &L, int i, ElemType e); /\* 顺序表的第i个位置之前插入元素e \*/

Status GetElem\_sq(SqList L, int i, ElemType &e); /\* 顺序表的取值 \*/

int ListLocate\_sq(SqList L, ElemType e); /\*查找值为e的元素\*/

Status InitList\_sq(SqList &L) // 顺序表的初始化(构造一个空的顺序表L)

{

L.elem = new ElemType[MAXSIZE];

//为顺序表分配一个大小为MAXSIZE的数组空间

if (!L.elem) {

printf("OVERFLOW\n");

exit(OVERFLOW); //存储分配失败退出

}

L.length = 0; //空表长度为0

return OK;

}/\*InitList\*/

Status CreateList\_sq(SqList &L, int n)

{

ElemType e;

int i;

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

printf("input data %d: ", i + 1);

scanf("%d", &e);

if (!ListInsert\_sq(L, (i + 1), e)) {

printf("ERROR\n");

return ERROR;

}

}

return OK;

}/\*CreateList\*/

/\* 输出顺序表中的元素 \*/

Status PrintList\_sq(SqList L)

{

int i;

for (i = 1; i <= L.length; i++)

printf("%5d", L.elem[i-1]);

return OK;

}/\*PrintList\*/

Status MergeList\_sq(SqList &LA, SqList LB, int n, int m) /\* 线性表LA LB的合并 \*/

{

/\* 将所有在线性表LB中但不在LA中的数据元素插入到LA中 \*/

int e;

for (int i = 1; i <= m; i++) {

GetElem\_sq(LB, i, e);

if (!ListLocate\_sq(LA, e))

ListInsert\_sq(LA, ++n, e);

}

return OK;

}/\*MergeList\_sq\*/

Status ListInsert\_sq(SqList &L, int i, ElemType e)

{

int k;

if (i < 1 || i > (L.length + 1)) {

printf("ERROR\n");

return ERROR;

}

if (L.length >= MAXSIZE) {

printf("ERROR\n");

return ERROR;

}

for (k = (L.length - 1); k >= (i - 1); k--)

L.elem[k+1] = L.elem[k];

L.elem[i-1] = e;

++L.length;

return OK;

}/\*ListInsert\*/

/\*在顺序表中查找指定值元素，返回其序号\*/

int ListLocate\_sq(SqList L, ElemType e)

{

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

if (L.elem[i] == e)

return i + 1; //查找成功，返回序号i+1

return 0; //查找失败，返回0

}/\*ListLocate\*/

Status GetElem\_sq(SqList L, int i, ElemType &e)

{

if (i < 1 || i > L.length)

return ERROR;

e = L.elem[i - 1];

return OK;

}/\*GetElem\*/

int main()

{

SqList LA, LB;

int n, m;

printf("please input LA(n):"); /\* 输入顺序表LA的元素个数 \*/

scanf("%d", &n);

printf("please input LB(m):"); /\* 输入顺序表的LB元素个数 \*/

scanf("%d", &m);

if ((n && m) > 0 && (n && m) <= 100) {

printf("\nCreate SqList(LA)):\n");

InitList\_sq(LA); /\*顺序表的初始化(构造一个空的顺序表LA)\*/

CreateList\_sq(LA, n); /\* 创建顺序表LA(顺序输入数据元素) \*/

printf("\nPrint SqList(LA):\n");

PrintList\_sq(LA); /\*输出顺序表LA\*/

printf("\nCreate SqList(LB):\n");

InitList\_sq(LB); /\*顺序表的初始化(构造一个空的顺序表LB)\*/

CreateList\_sq(LB, m); /\* 创建顺序表LB(顺序输入数据元素) \*/

printf("\nPrint SqList(LB):\n");

PrintList\_sq(LB); /\*输出顺序表LB\*/

MergeList\_sq(LA, LB, n, m); /\* 线性表LA LB的合并 \*/

printf("\nPrint SqList(LA U LB):\n");

PrintList\_sq(LA); /\*输出顺序表LA = LA U LB\*/

}

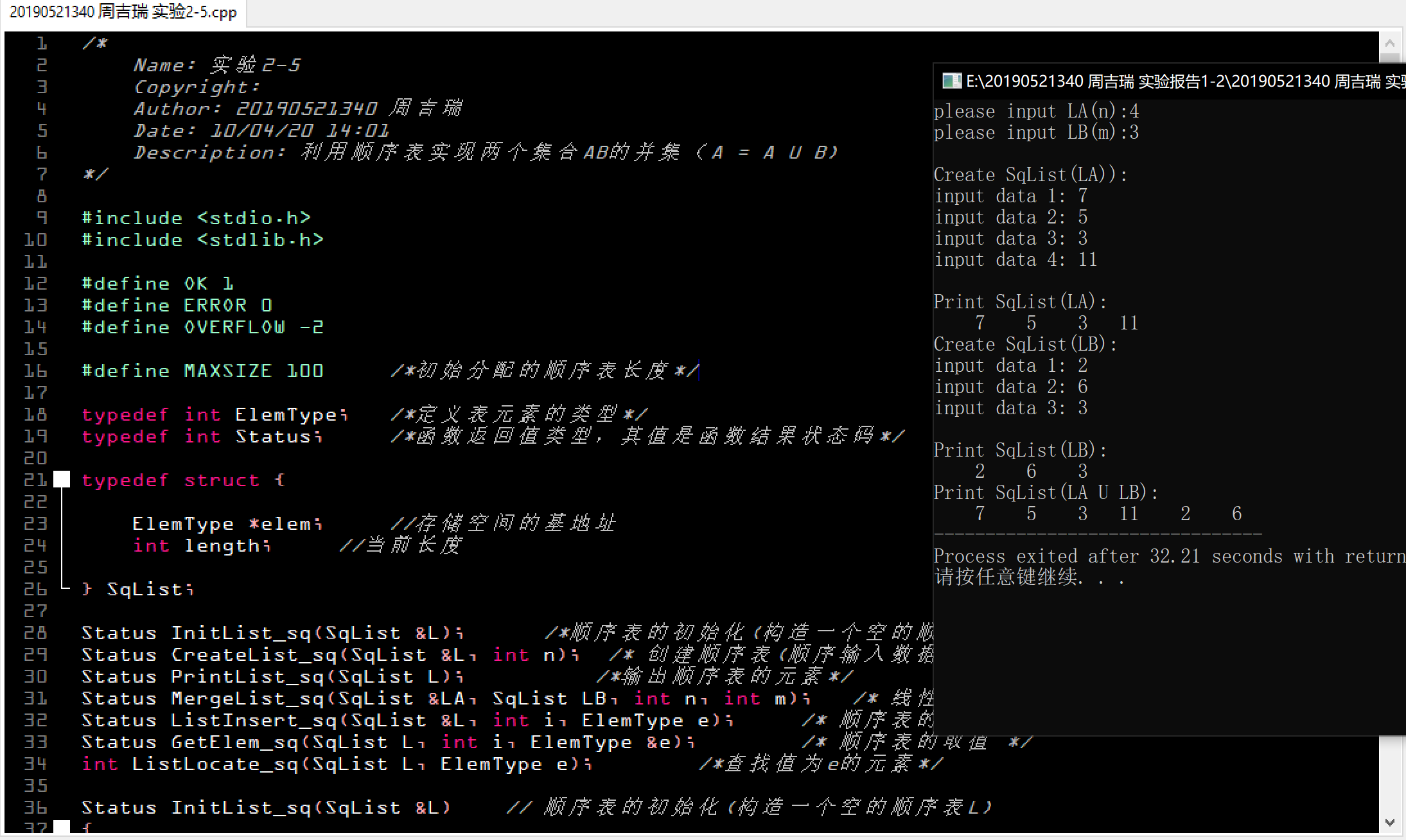
else

printf("ERROR\n");

return 0;

}

* 运行结果



### 四、实验小结

通过做实验2我深刻感悟到一下重要的几点：

1. 学习数据结构与算法，关键在于理解，不要能一味的将教材中的代码搬上来。我刚开始做实验2时，一味地将老师给的程序与教材中的程序一一对应，但是发现同样功能的程序好多地方都不一样，由于没有深刻理解数据结构与算法的内涵，导致程序好多地方出错并且也发现不了错误的地方，甚至好的地方都没有理解到位，后来我再次把教材一页一页的翻开看，一题一题，一个算法一个算法的分析理解，最后再来将实验所给的程序一句一句地检查，这时许多问题便迎刃而解了，我认为这是本次实验带给我最大的帮助，也是关于如何才能学好数据结构与算法的最大的体会！
2. 在进行实验2时，出现了许多问题，下面进行总结归纳：

(1).最常出现的问题就是“名称”不对应，例如：变量名称不对应，函数名称不对应，函数参数不对应……并且在出错时，往往较难找出，这要求写代码时，思路要清晰

(2).对结构体指针的理解不牢固，结构体指针本质上还是一个普通的指针，还是一个地址，只不过这个地址是一个结构体的地址，而结构体指针变量不过是一个保存结构体地址的普通变量，所以给结构体指针变量赋值依旧只能赋地址值。

例如：LNode \*p, \*n;

p = new LNode;

n = p;

以上代码表示：将p所指 赋给 n所指 因为new函数返回的是动态内存的初地址，即：把动态内存的初地址 赋值给 n

LNode \*n;

n = L->next;

以上代码可改为：

n = L;

n = n->next;

原因：n->next是一个地址，表示把n所指结构体中next保持的下一个结点的地址赋给n，就表示n指向下一个结点，n = L;表示n指向头结点，

n = n->next;表示n指向头结点的下一个结点首元结点；而

n = L->next;表示n直接指向首元结点

1. 在算法的分析中，不要只用眼睛看，一个用笔一步一步的算，这样才能有效分析出算法到底正确与否，也更好理清算法的执行思路。
2. 一定要区分 “=”和“==”，尤其在条件判断中，判断相等一定要“==”！！！