数据结构实验报告

实验成绩：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 专业班级 |  |
| 指导教师 | 张先宜 | 实验时间 | 4月5日 | 实验地点 | 4号机房 |

**实验名称：** 单链表实验

1. **实验目标**
2. 熟练掌握线性表的链式存储结构。
3. 熟练掌握单链表的有关算法设计和实现。
4. 根据具体问题的需要，设计出合理的表示数据的链式存储结构，并设计相关算法。
5. **实验要求**

实验要求

1. 本次实验中的链表结构指带头结点的单链表；
2. 单链表结构和运算定义，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：linkedList.h
3. 程序运行、测试正确；
4. 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；
5. 程序有适当的注释，程序的书写要采用缩进格式。
6. **实验内容**

编写算法实现下列问题的求解。

1. 在第i个结点位置插入值为x的结点。

实验测试数据基本要求：

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

第二组数据：单链表长度n=0，x=100，i=5

1. 删除单链表中第i个元素结点。

实验测试数据基本要求：

第一组数据：单链表长度n≥10，i分别为5,n,1,n+1,0

第二组数据：单链表长度n=0， i=5

1. 在一个递增有序的单链表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：

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

x分别为25，85，110和8

1. 将单链表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数），分别放入新的单链表中，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：单链表元素为 （1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）

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

1. 求两个递增有序单链表L1和L2中的公共元素，放入新的单链表L3中。

实验测试数据基本要求：

第一组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）

第二组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

第二个单链表元素为 （2，4，5，7，8，9，12，22）

第三组

第一个单链表元素为 （）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）

1. 删除递增有序单链表中的重复元素，要求时间性能最好。

实验测试数据基本要求：

第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9）

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

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

1. 递增有序单链表L1、L2，不申请新结点，利用原表结点对两表进行合并，并使得合并后成为一个集合，合并后用L1的头结点作为头结点，删除多余的结点，删除L2的头结点。要求时间性能最好。实验测试数据基本要求：

第一组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）

第二组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

第二个单链表元素为 （2，4，5，7，8，9，12，22）

第三组

第一个单链表元素为 （）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）

1. **数据结构设计**

struct slNode

{

int data; //数据域

struct slNode \*next; //指针域，结构（结点）自身引用

};

typedef struct slNode node, \*linkList;

1. **算法设计（宋体、小四号、加粗）**

**5.1、5.2**均为书上的基本运算，故算法设计思想省略。

**5.3**

【算法思想】

（1）声明一个结点指针变量，使其指向链表的头结点L；

（2）再产生一个结点u，装入插入的元素x；

（3）遍历链表：判断结点p的next是否为空，若不为空，再判断结点p的元素值是否比x大，若比x大，将u的next指向p的next，p的next指向u，结束插入操作；若比x小，将p指向p的下一个结点，进行下一循环。

【算法描述】

void listInsert2(node\* L, int x)

{

node\* u;

node\* p = L;

while (p->next != NULL && p->next->data < x) //搜索插入位置p

{

p = p->next;

}

if (p->next == NULL || p->next->data > x) //p后移一个结点

{

u = new node;

u->data = x;

u->next = p->next;

p->next = u;

}

listOutput(L);

}

**5.4**

【算法思想】

（1）首先建立两个空表，用于存放分解开的偶数和奇数；

（2）声明一个结点指针u，使其指向头结点L的next；同时声明两个变量a和b,并将它们初始化为0，用于记录插入新表中的偶数或奇数的位置；

（3）判断u结点是否为空，若为空，结束；

（4）否则，判断u结点的元素是否为偶数，若为偶数，a自增，调用listInseart()函数，插入该偶数到L1表中，u指向u的下一个结点；若为奇数，b自增，调用listInseart()函数，插入该奇数到L2表中，u指向u的下一个结点。

（5）调用listOutput（）函数，将原表和两个新表都输出到屏幕上。

【算法描述】

void listSeperate(linkedList& L)

{

linkedList L1, L2;

initialList(L1); //奇数表

initialList(L2); //偶数表

node\* p; //原单链表的指针

node\* u; //奇偶新链表的结点指针

node\* q1, \*q2; //奇数表、偶数表的指针

p = L->next; //p开始指向首元素结点

q1 = L1;

q2 = L2;

while (p != NULL)

{

//原表数据尾插插入新表元素

if (p->data % 2 != 0) //奇数表

{

u = new node;

u->data = p->data;

u-> next = NULL;

q1->next = u;

q1 = q1->next;

//q1 = u;

}

else //偶数表

{

u = new node;

u->data = p->data;

u->next = NULL;

q2->next = u;

q2 = q2->next;

// q2 = u;

}

p = p->next;

}

cout << "原链表L为：";

listOutput(L);

cout << "奇数链表L1为：";

listOutput(L1);

cout << "偶数链表L2为：";

listOutput(L2);

}

**5.5**

【算法思想】

（1）创建一个新的空表L3；

（2）声明两个结点指针变量p1和p2，p1指向L1的头结点的next，p2指向L2的头结点的next；

（3）同时遍历L1和L2两表，从两表的头元素开始比较，如果两个元素相等，则利用尾插法将数据存放到新链表L3中，然后p1和p2后移；如果L1中的元素小于L2中的元素，则u指向u的下一个结点，否则p指向p的下一个结点；

（4）调用listOutput()函数，将新表L3中的元素输出到屏幕上；

（5）销毁链表L3。

【算法描述】

void listBoth(linkedList &L1, linkedList &L2)

{

linkedList L3;

initialList(L3);

node\* p1, \* p2, \* p3;

node\* q; //创立存放公共元素的新结点

p1 = L1->next;

p2 = L2->next;

p3 = L3;

while (p1 != NULL && p2 != NULL)

{

if (p1->data == p2->data)

{

q = new node;

q->data = p1->data;

q->next = NULL;

p3->next = q;

p3 = p3->next; //新表的指针后移一个

p1 = p1->next;

p2 = p2->next;

}

else if (p1->data < p2->data)

{

p1 = p1->next;

}

else if (p1->data > p2->data)

{

p2 = p2->next;

}

}

cout << "新表L3为：";

listOutput(L3);

delete L3;

}

**5.6**

【算法思想】

（1）声明两个结点指针变量p和q，使其分别指向头结点L->next和L->next->next；

（2）遍历链表，将p结点的元素和q结点的元素进行比较，若两者相等，q指针后移，p->next指针指向q；若不相等，将p和q指针继续后移比较，直至q指向NULL结束；

（4）遍历完后，调用listOutput()函数将删除后的新表元素输出到屏幕上。

【算法描述】

void listCancle(linkedList L)

{

node\* p, \* q;

p = L->next;

q = L->next->next;

while (q)

{

if (p->data == q->data)

{

q = q->next;

p->next = q;

}

else

{

p = p->next;

q = q->next;

}

}

cout << "删除后的链表L为：";

listOutput(L);

delete L;

}

【算法分析】

算法的时间复杂度为O（n）。

**5.7**

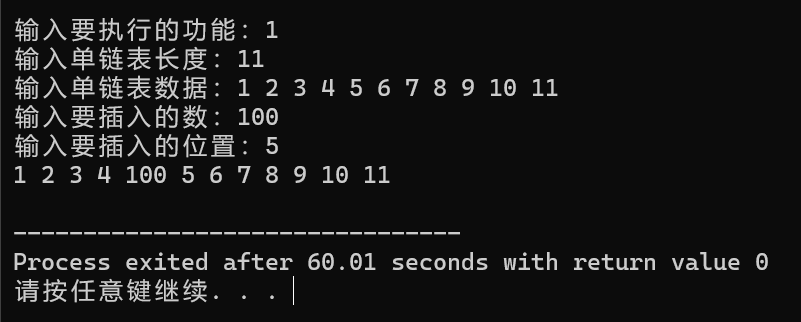
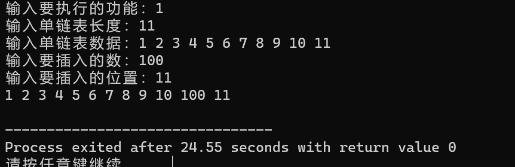
【算法思想】

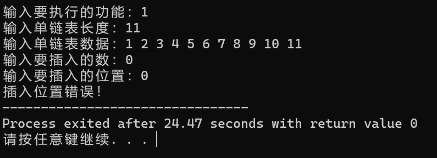
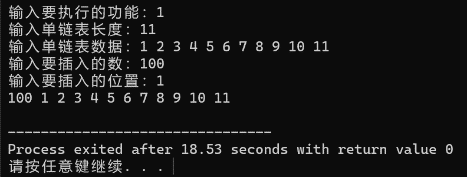
新建指针p，q，n，u；  
p = L1->next;  
q = L2->next;  
n = L1;  
  
比较p=->data与q->data的大小；如果p->data小，n->next=p,p=p->next;  
如果q->data小，n->next=q,q=q->next;(把小的那个值存入L1链表中)  
如果p->data与q->data一样；u=q;q=q->next;释放u；相当于删除一个重复元素。  
遍历链表，直到其中一个表结束。把未完的那个表后面的数直接用n->next接入L1；  
然后释放L2头结点。  
最后打印L1，

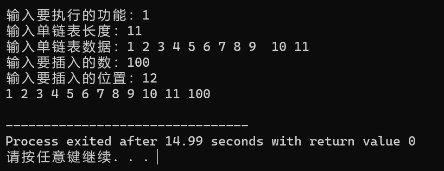
【算法描述】

void listUnite(linkList& L1, linkList& L2)  
{  
    node\* p, \* q,\*n;  
    p = L1->next;  
    q = L2->next;  
    n = L1;  
    while (q != NULL && p != NULL) {  
        if (p->data > q->data) {  
            n ->next= q;  
            n = q;  
            q = q->next;  
        }  
        else if (p->data == q->data) {  
            node\* u;  
            u = q;  
            q = q->next;  
            delete u;  
        }  
        else if (p->data < q->data) {  
            n->next = p;  
            n = p;  
            p = p->next;  
        }  
   }  
    if (p != NULL) {  
        n->next = p;  
    }  
    if (q != NULL) {  
        n->next = q;  
    }  
    delete L2;  
    show(L1);  
}

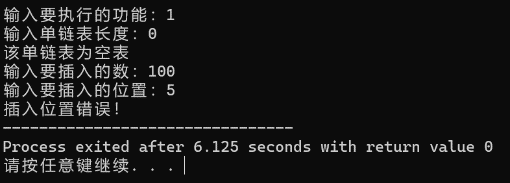
1. **运行和测试**
2. 第一组数据：单链表长度n≥10，x=100, i分别为5,n,n+1,0,1,n+2（以n=11为例）

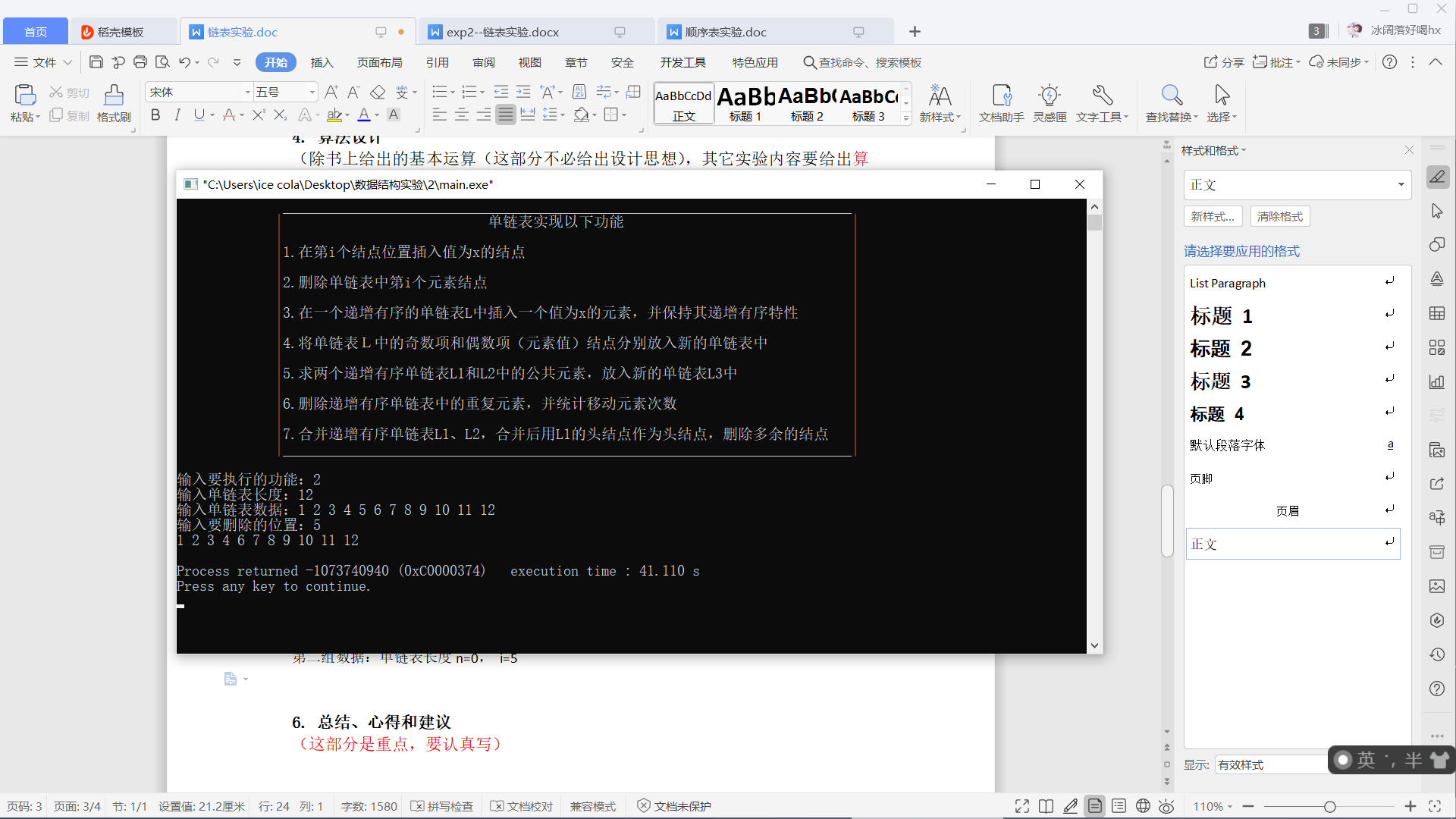
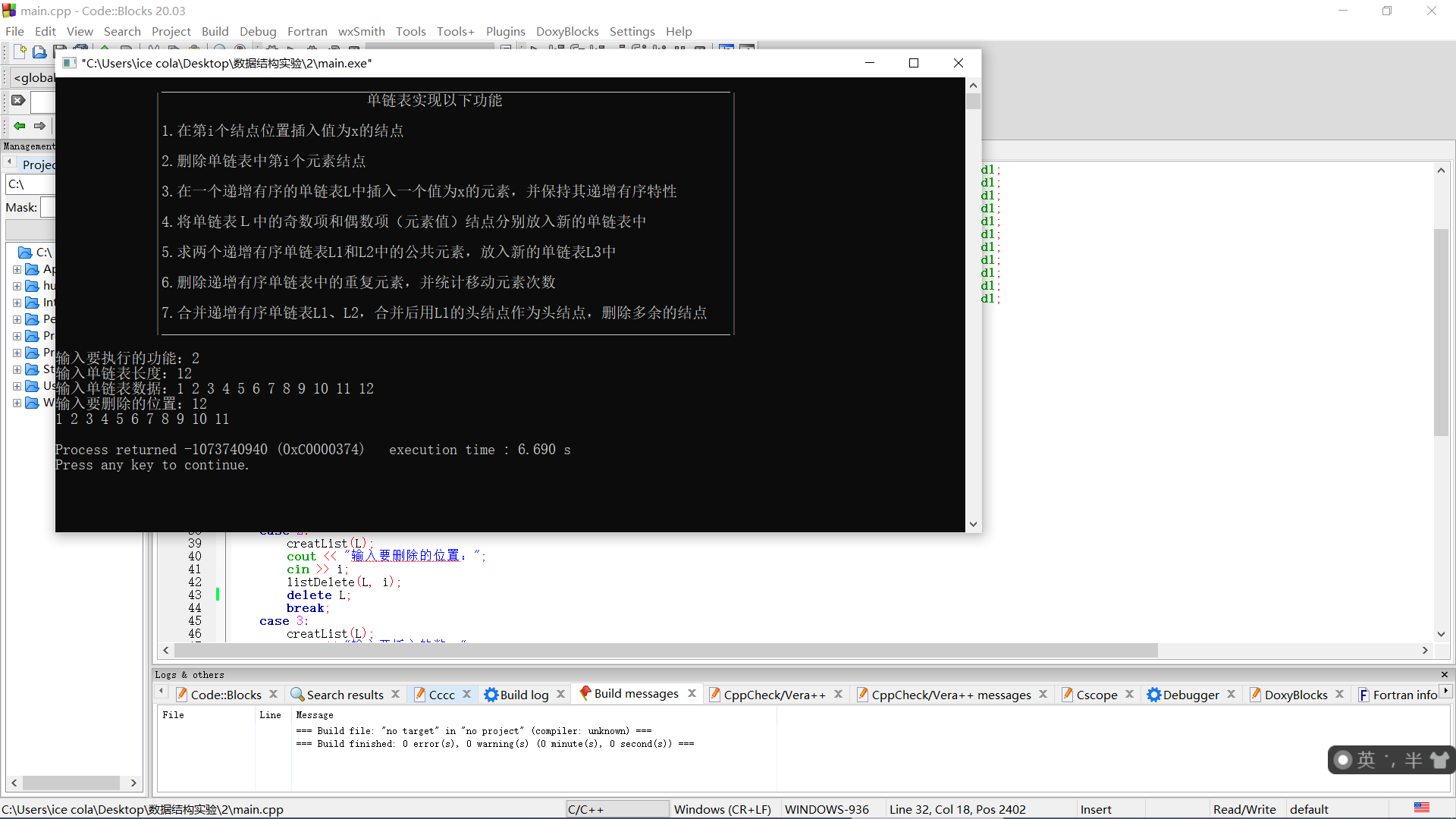
 

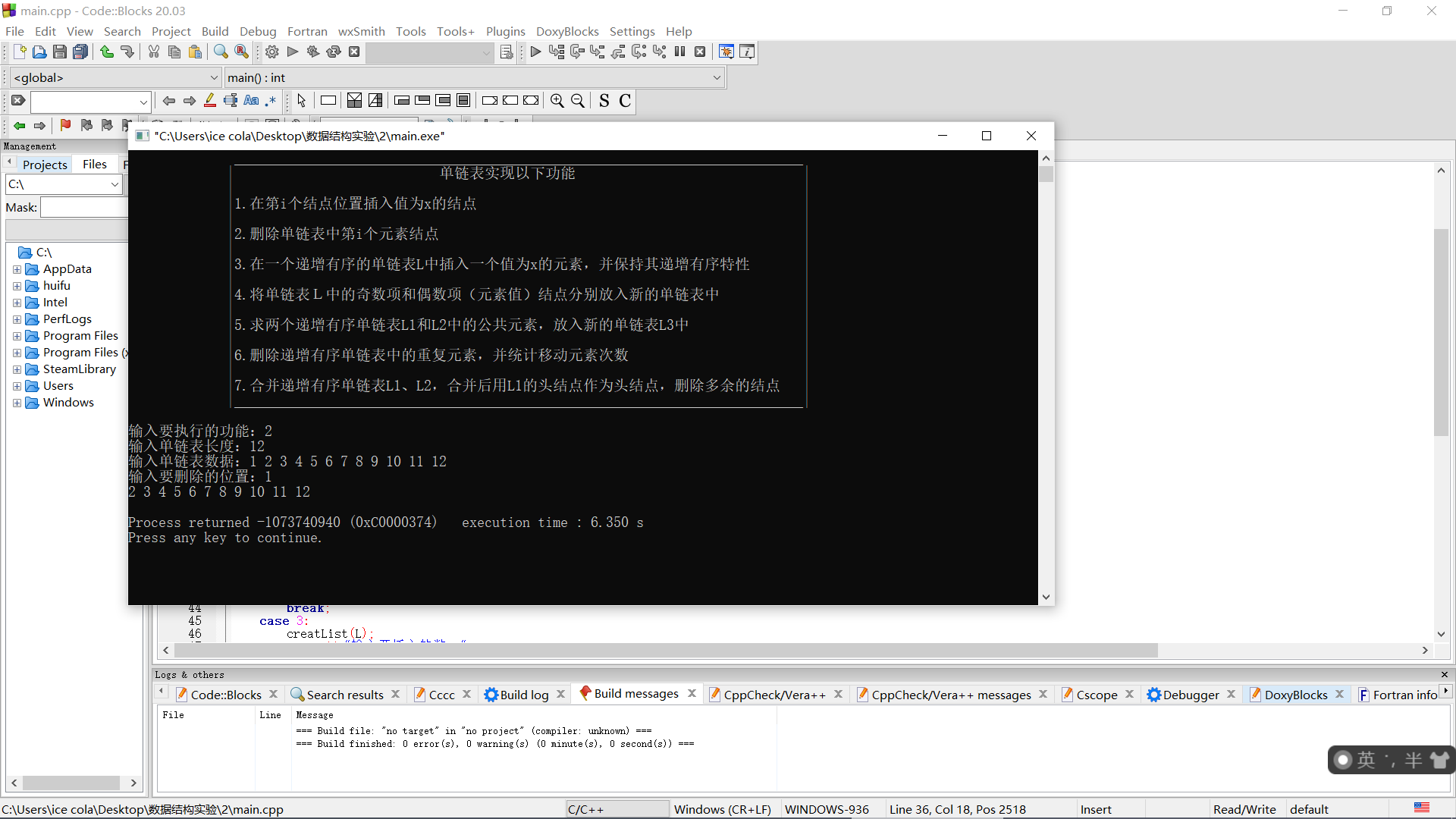
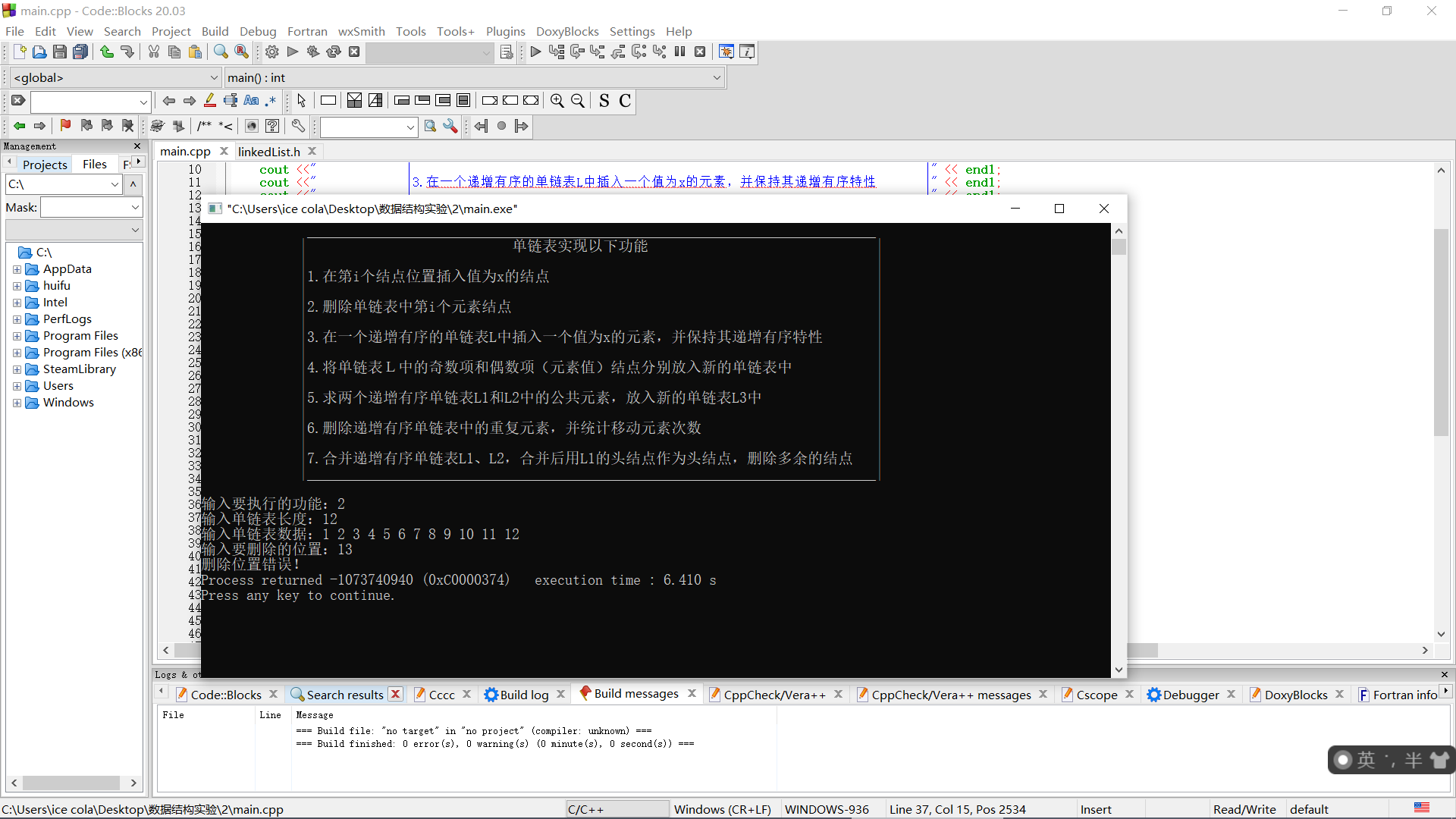


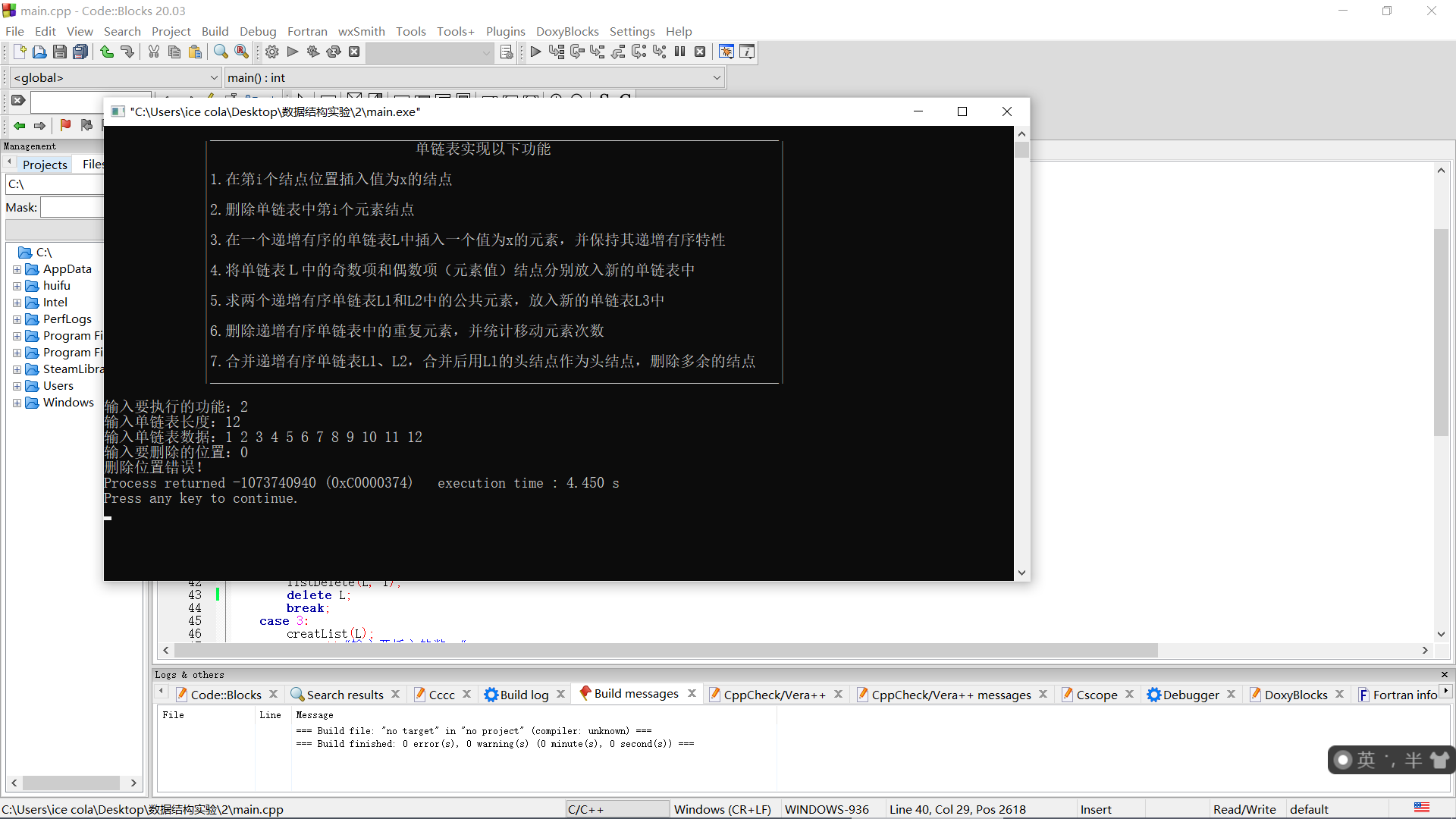
第二组数据：单链表长度n=0，x=100，i=5



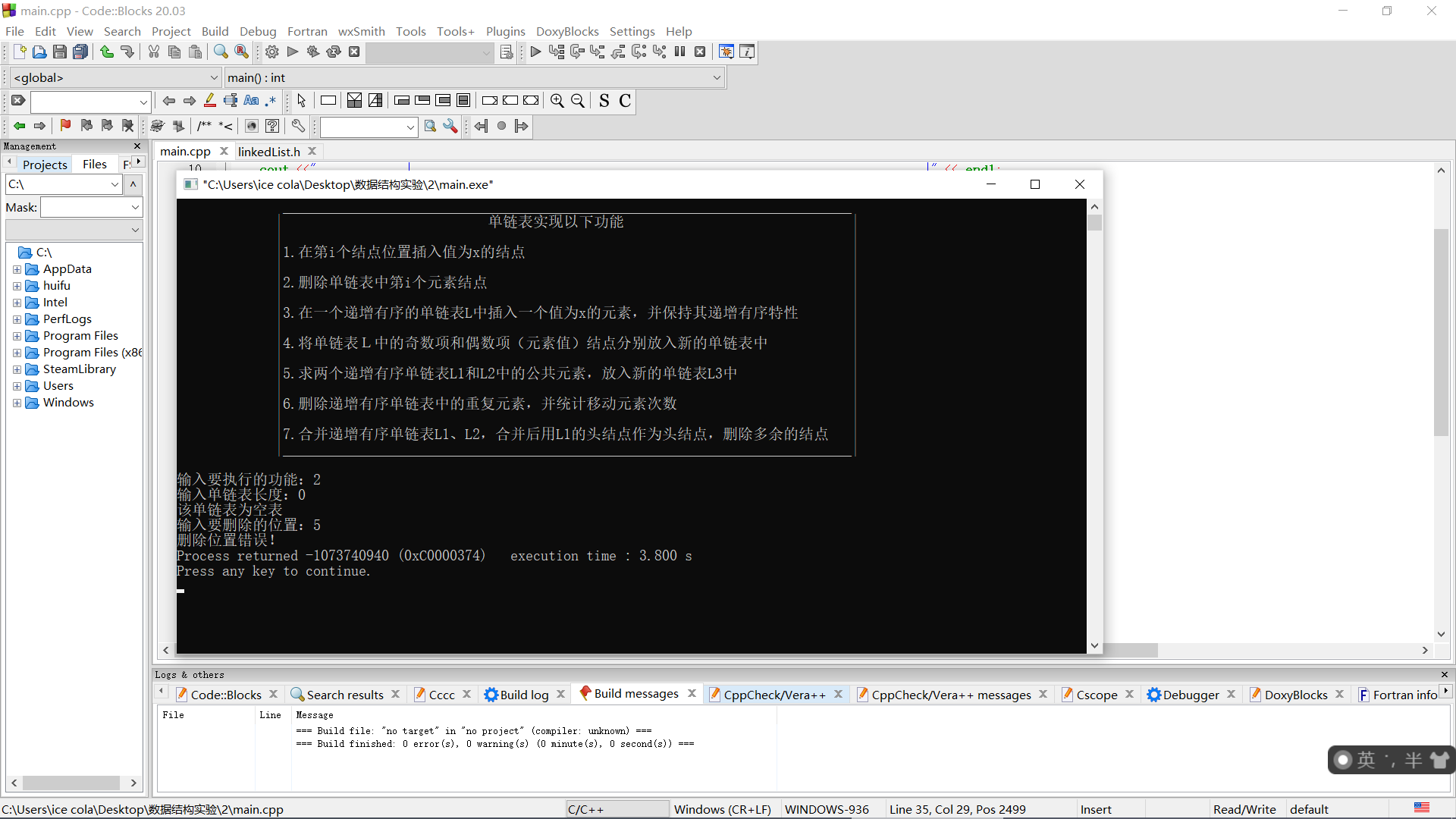
②第一组数据：单链表长度n≥10，i分别为5,n,1,n+1,0 （以n=12为例）

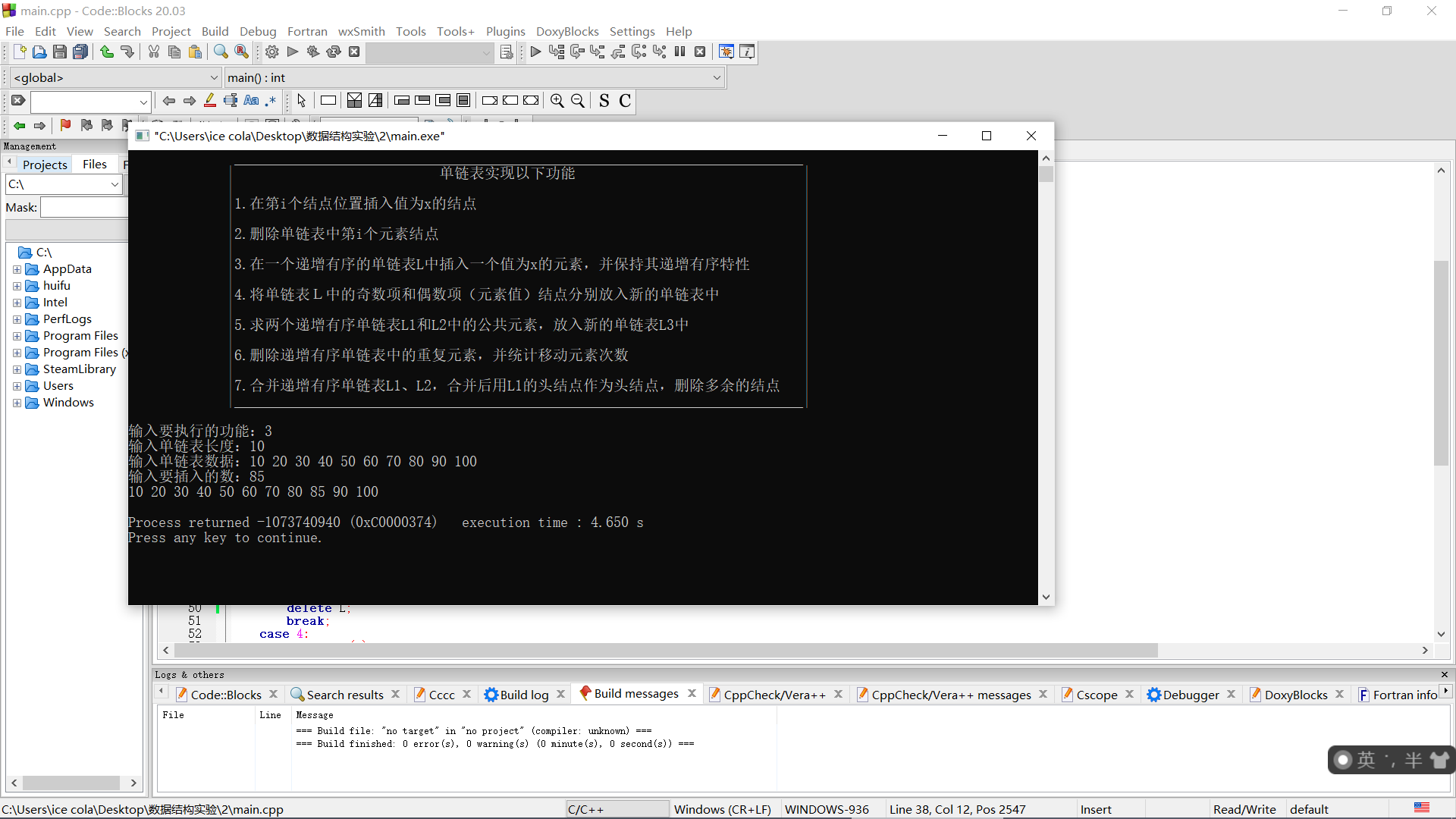
 

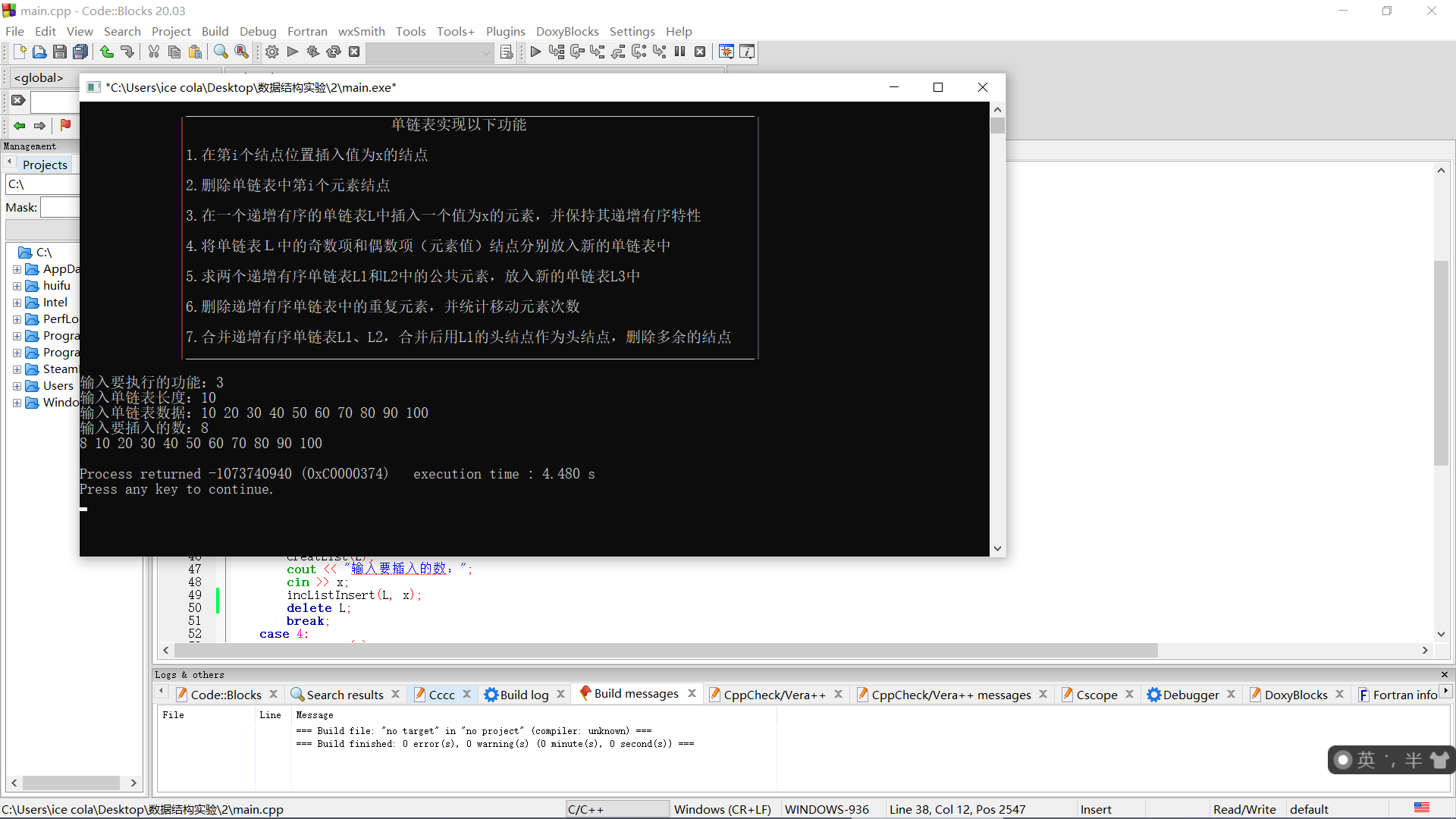
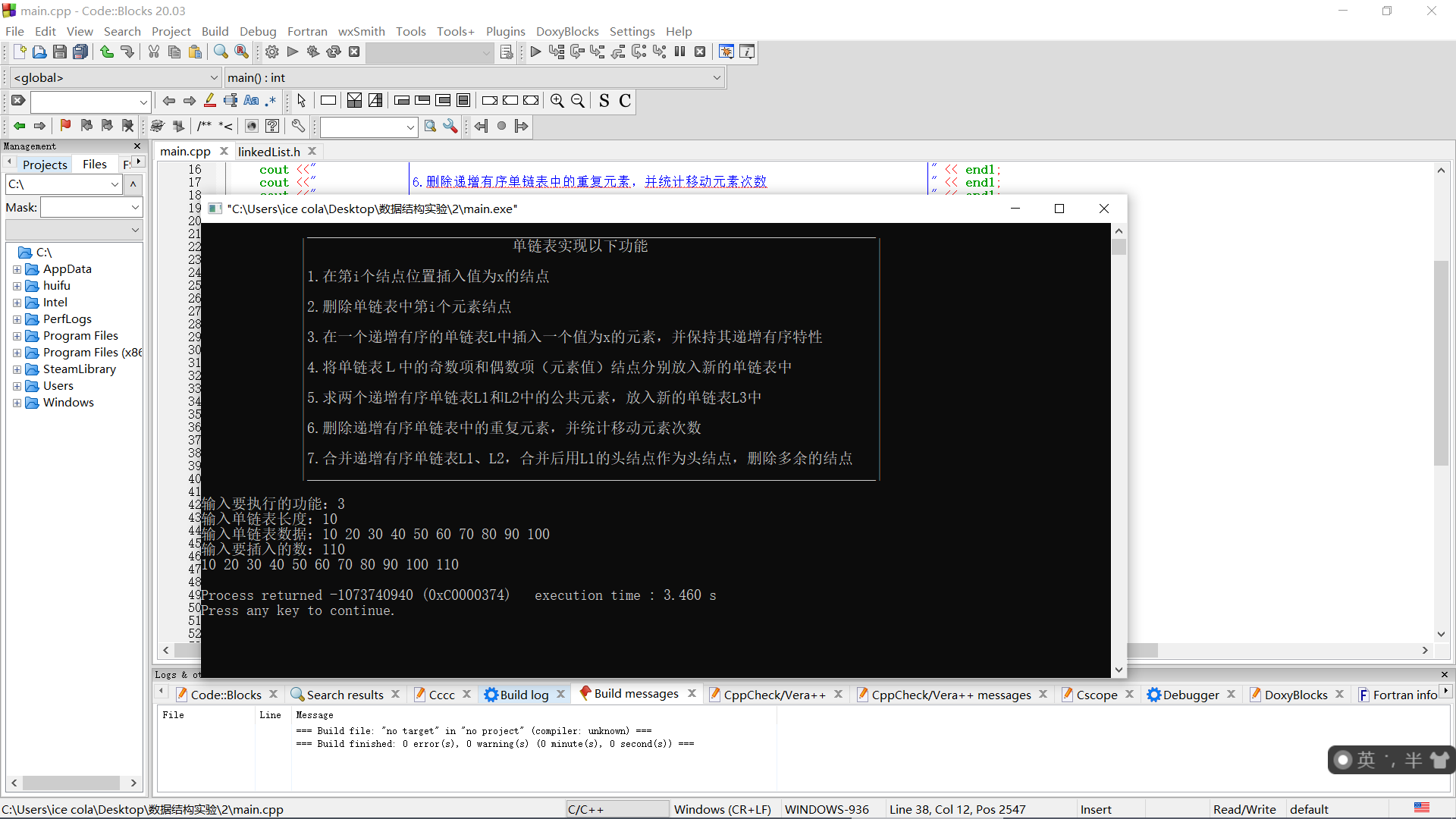


第二组数据：单链表长度n=0， i=5

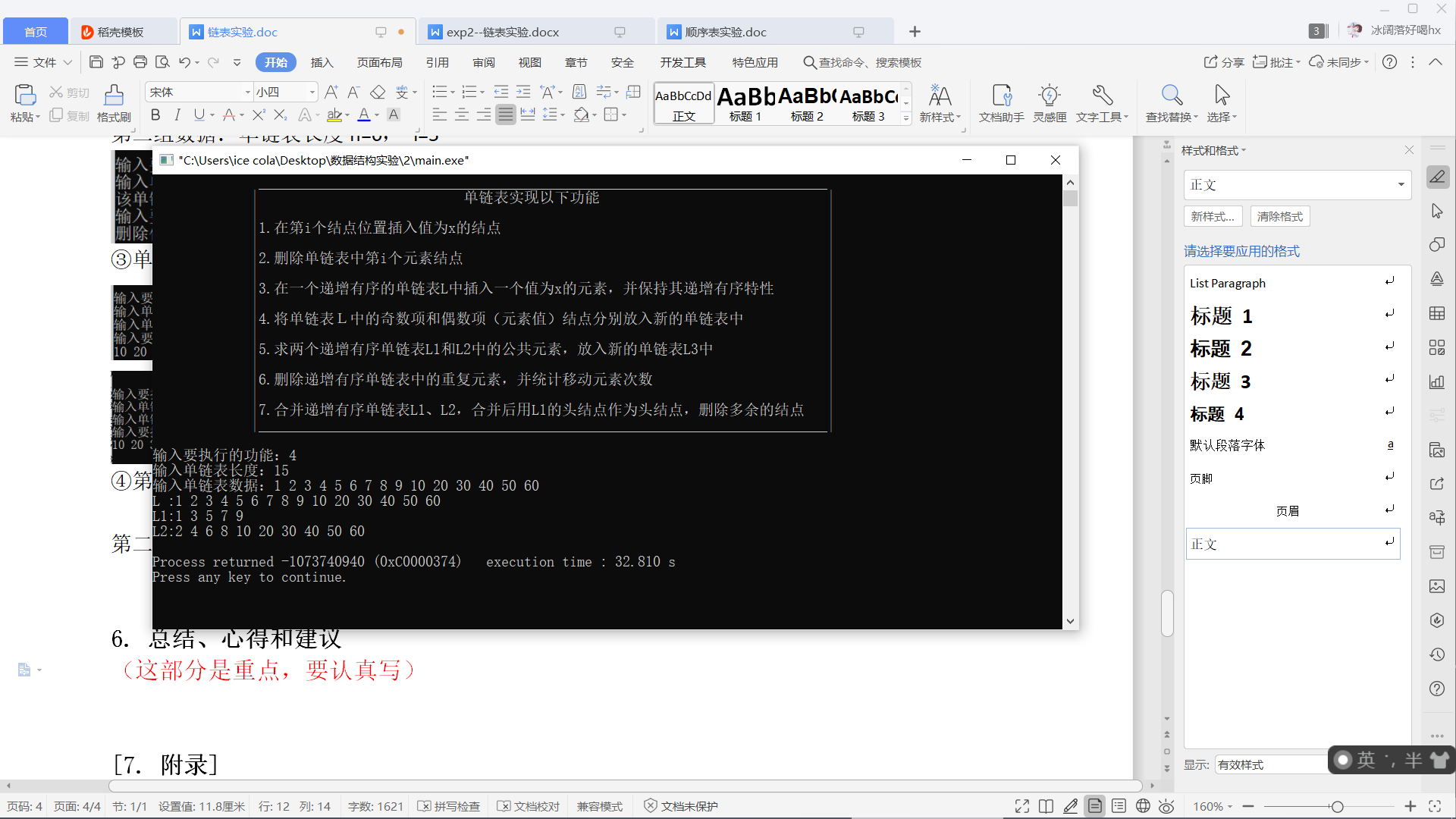


③单链表元素为 （10,20,30,40,50,60,70,80,90,100），x分别为25，85，110和8

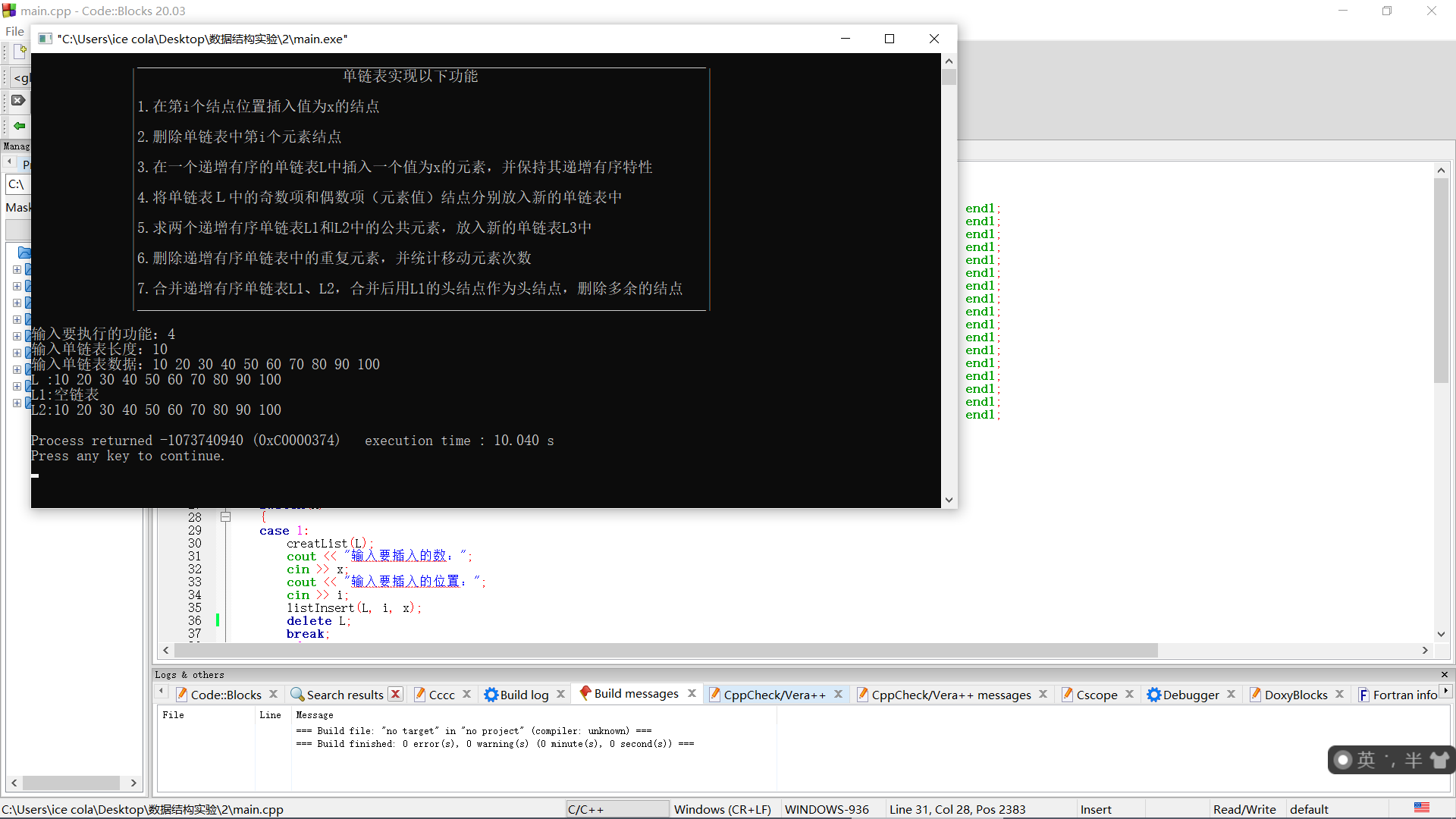




④第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）



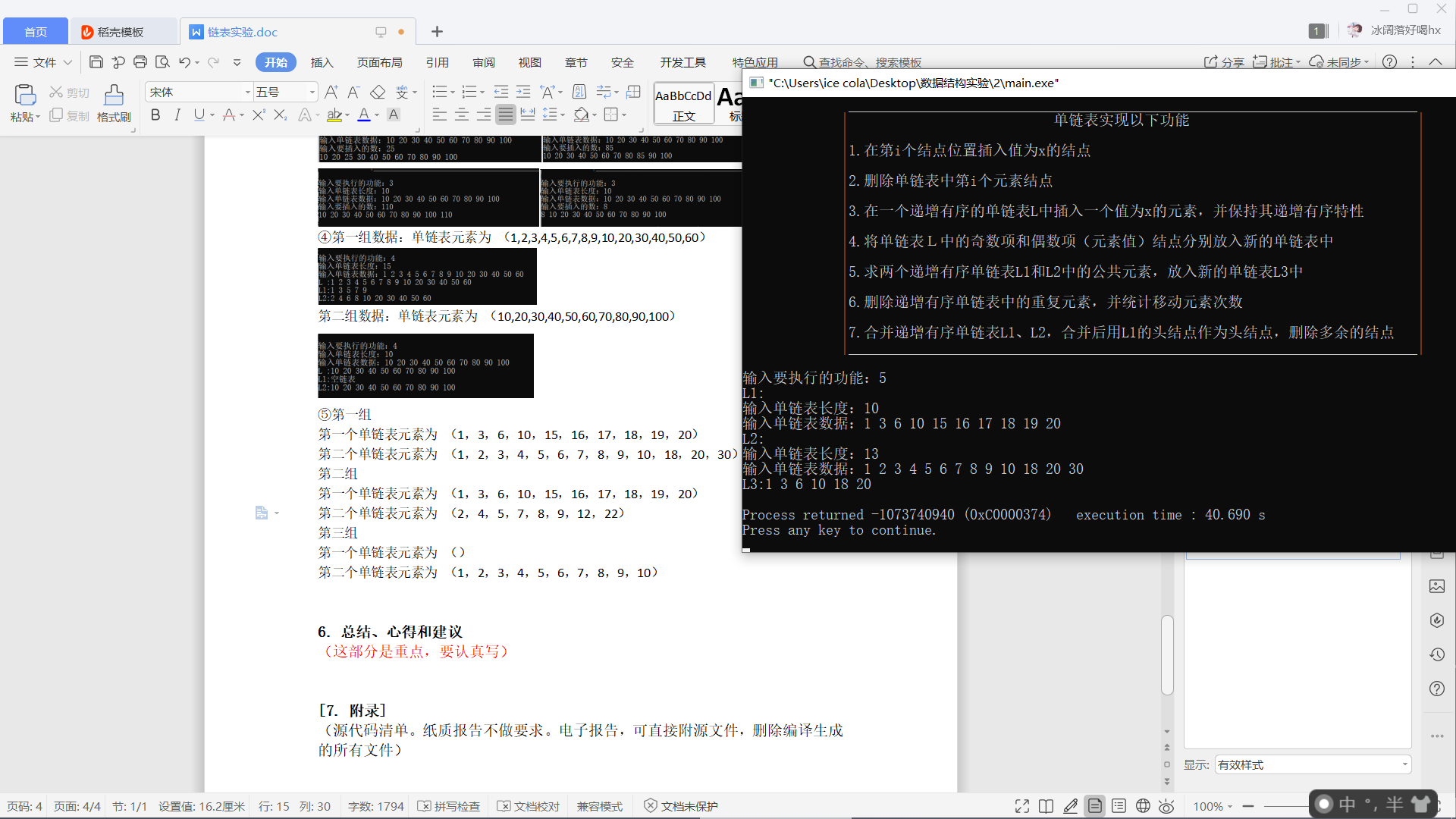
第二组数据：单链表元素为 （10,20,30,40,50,60,70,80,90,100）



⑤第一组

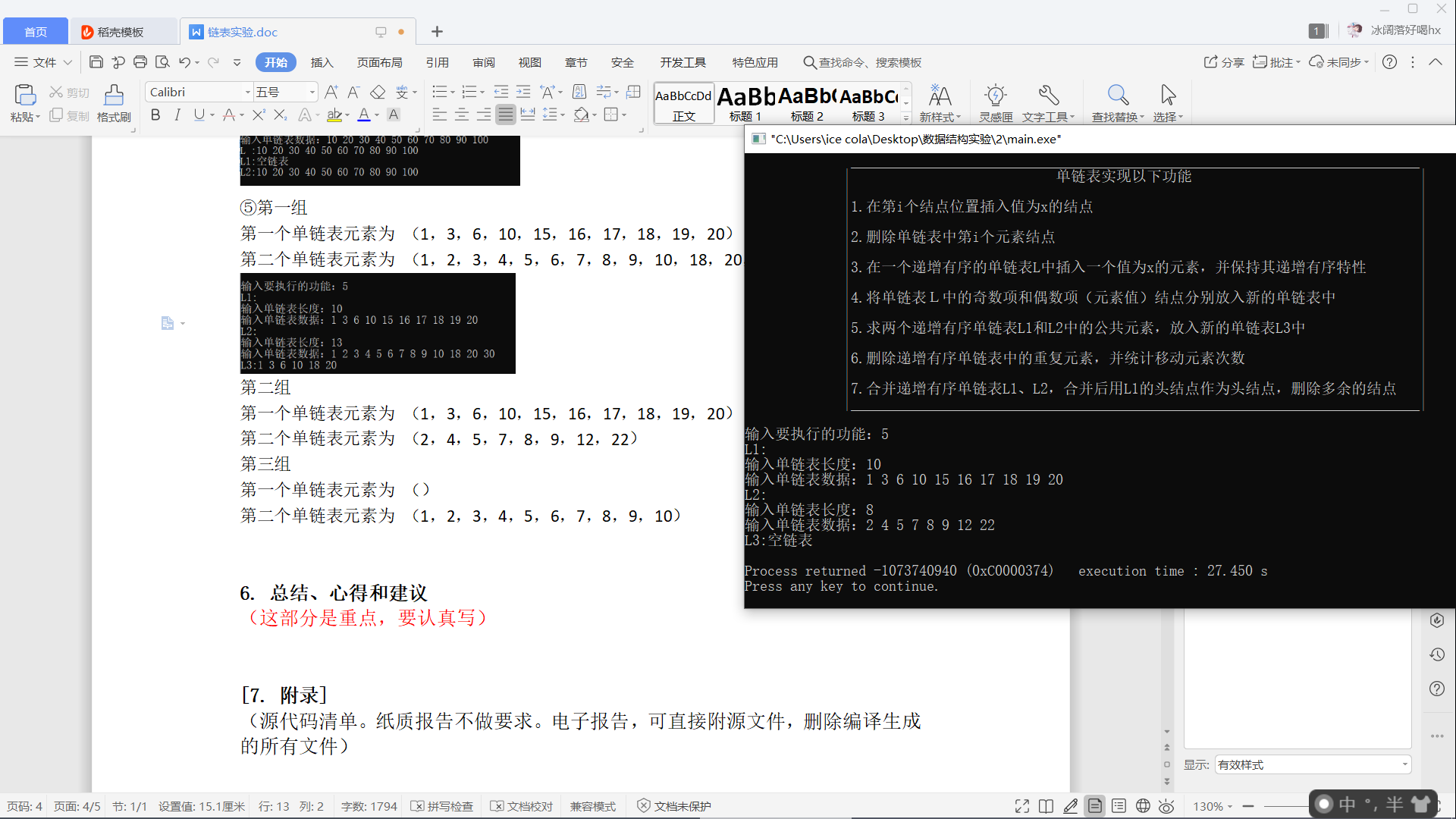
第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）

第二组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

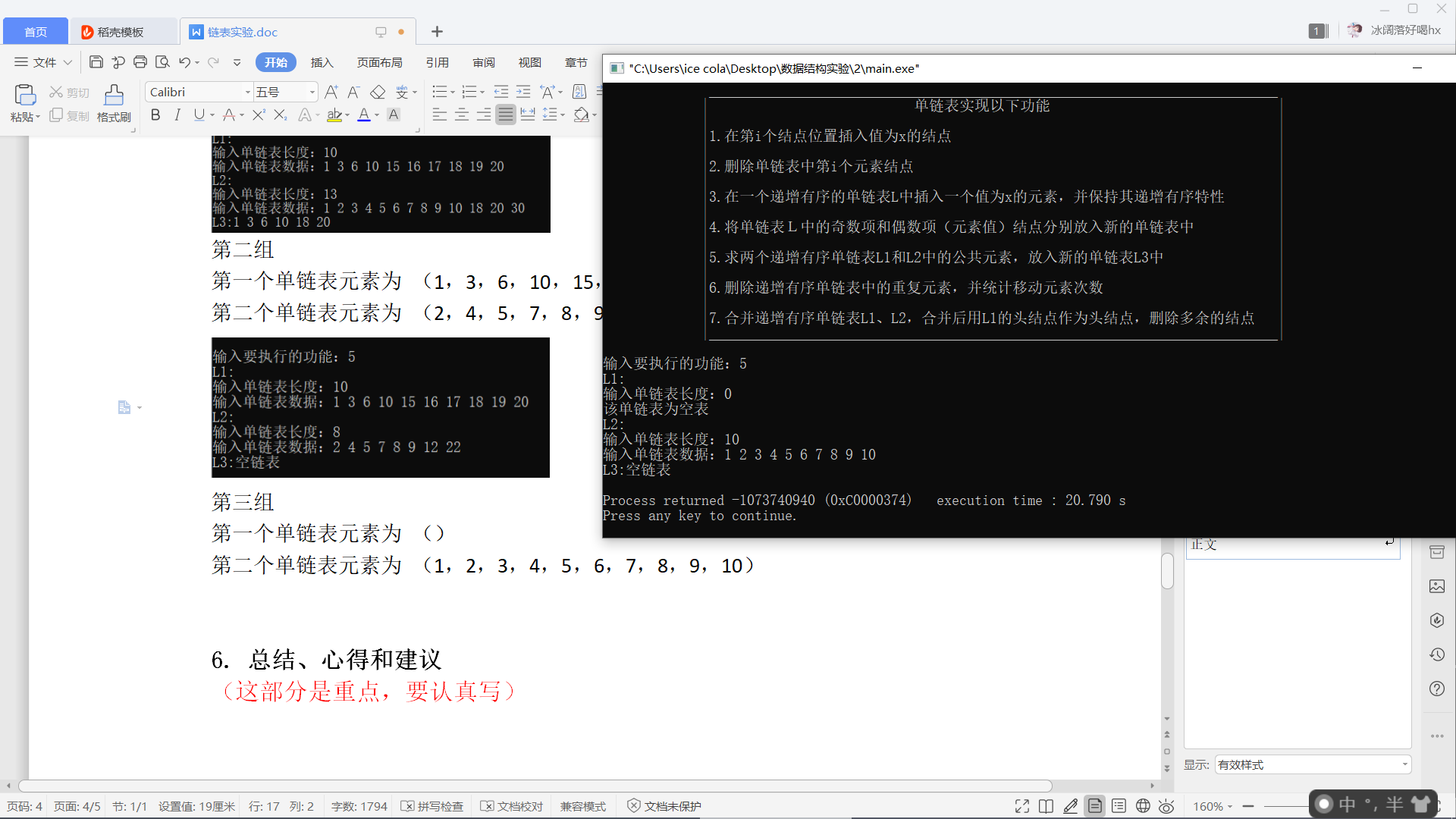
第二个单链表元素为 （2，4，5，7，8，9，12，22）



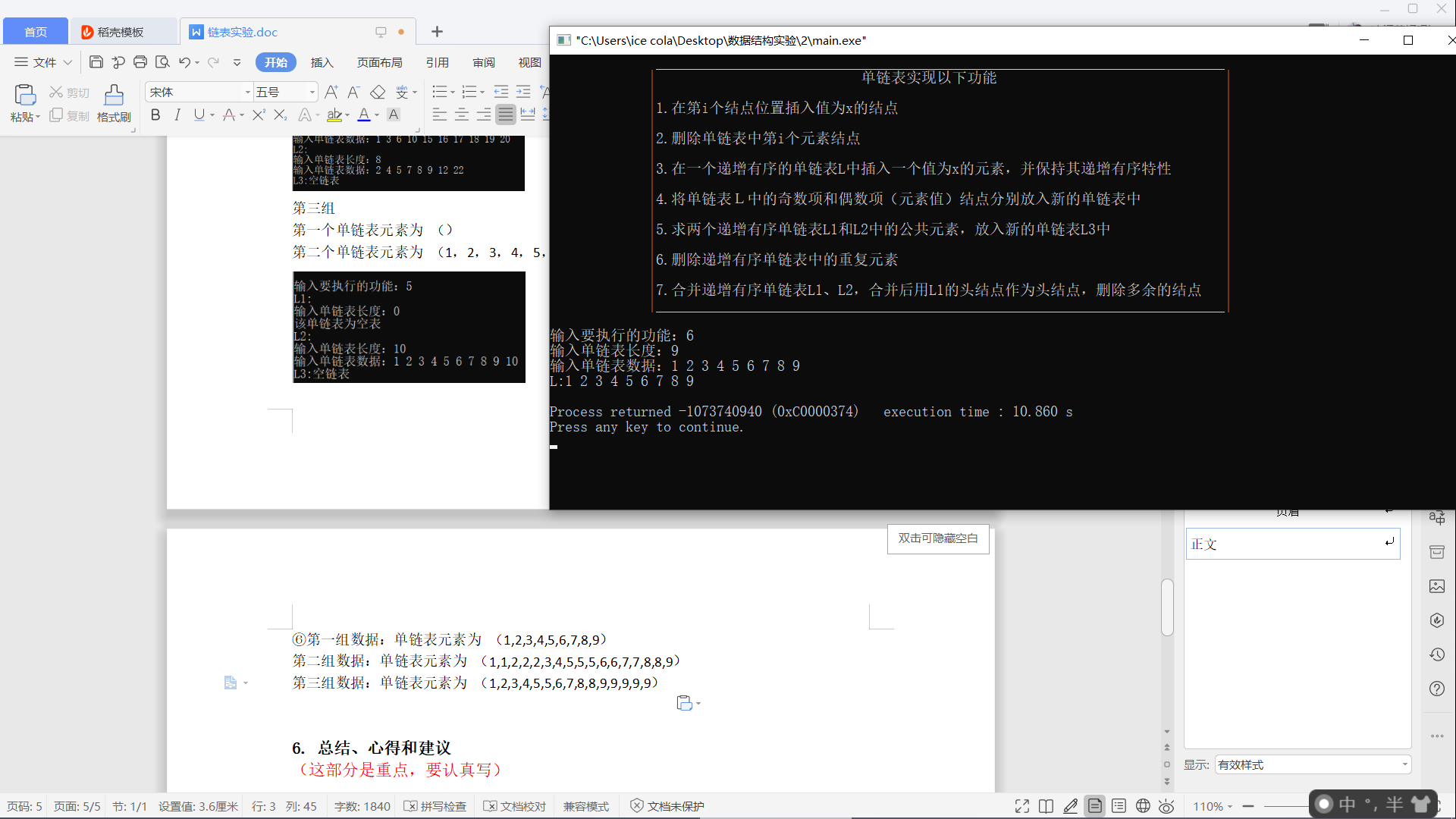
第三组

第一个单链表元素为 （）

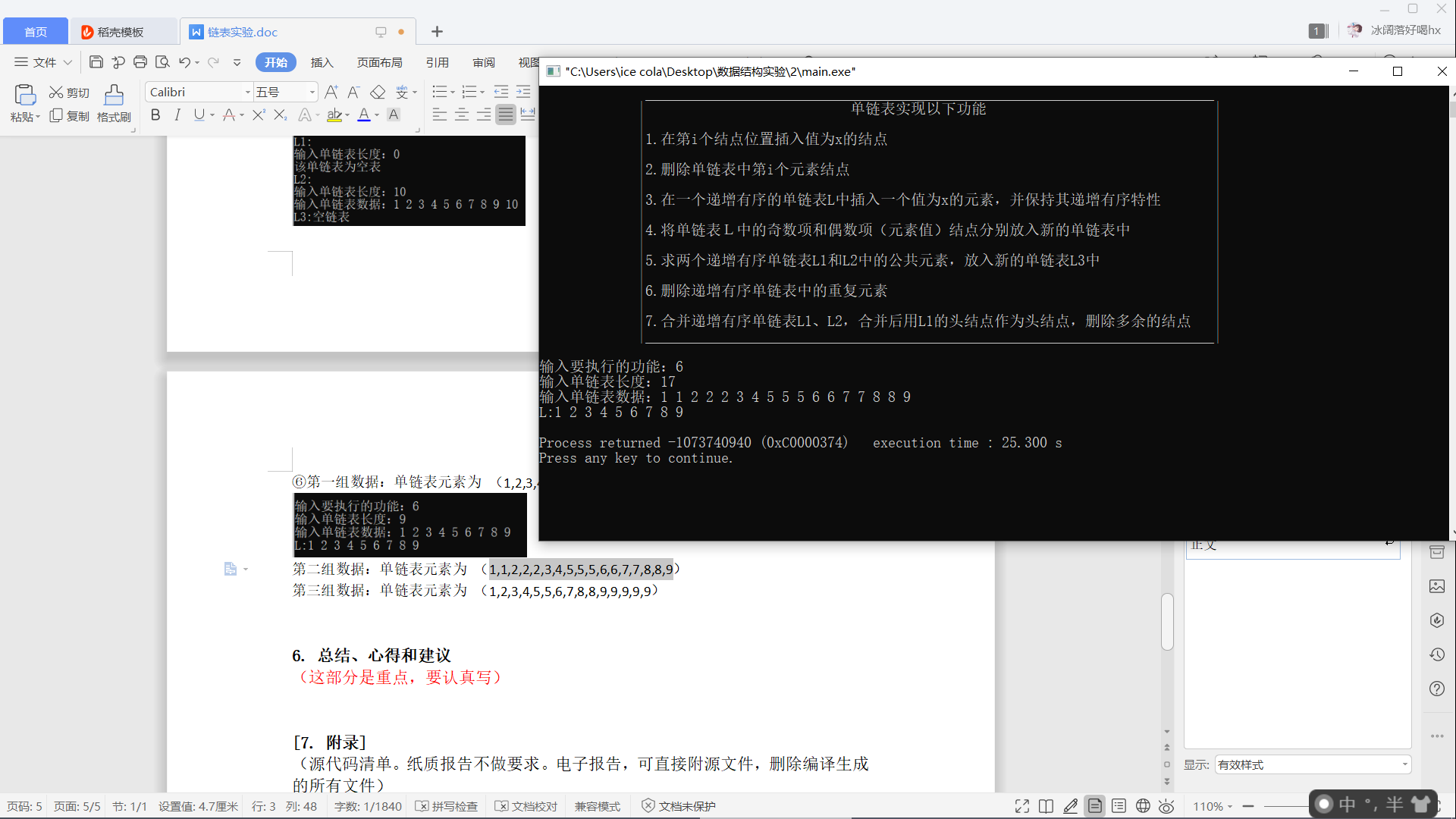
第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）



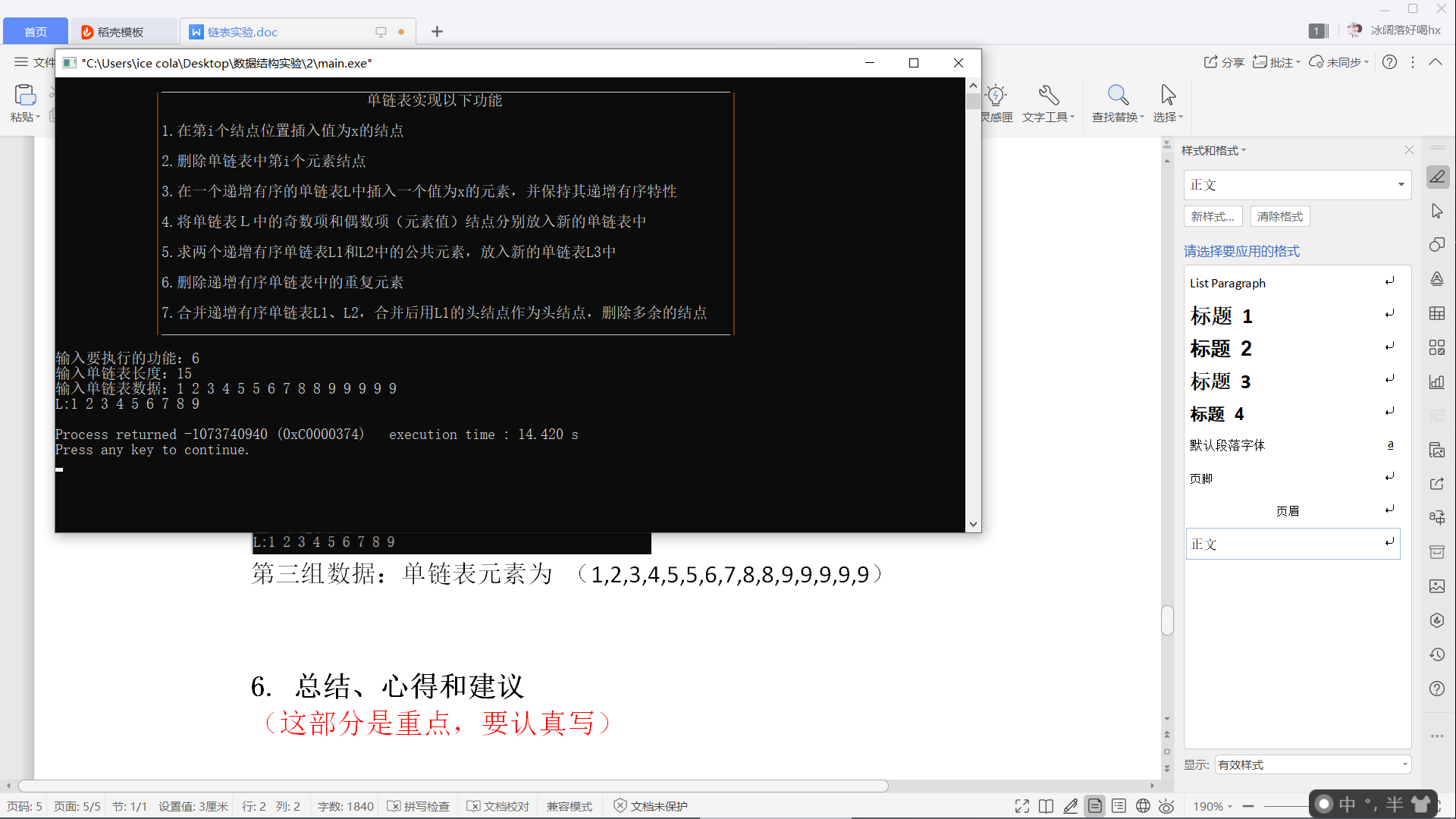
⑥第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9）



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



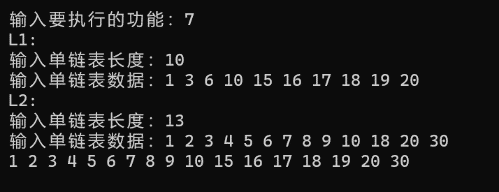
第三组数据：单链表元素为 （1,2,3,4,5,5,6,7,8,8,9,9,9,9,9）



⑦第一组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

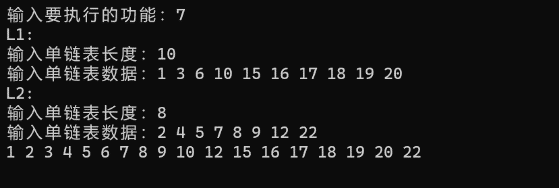
第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）



第二组

第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）

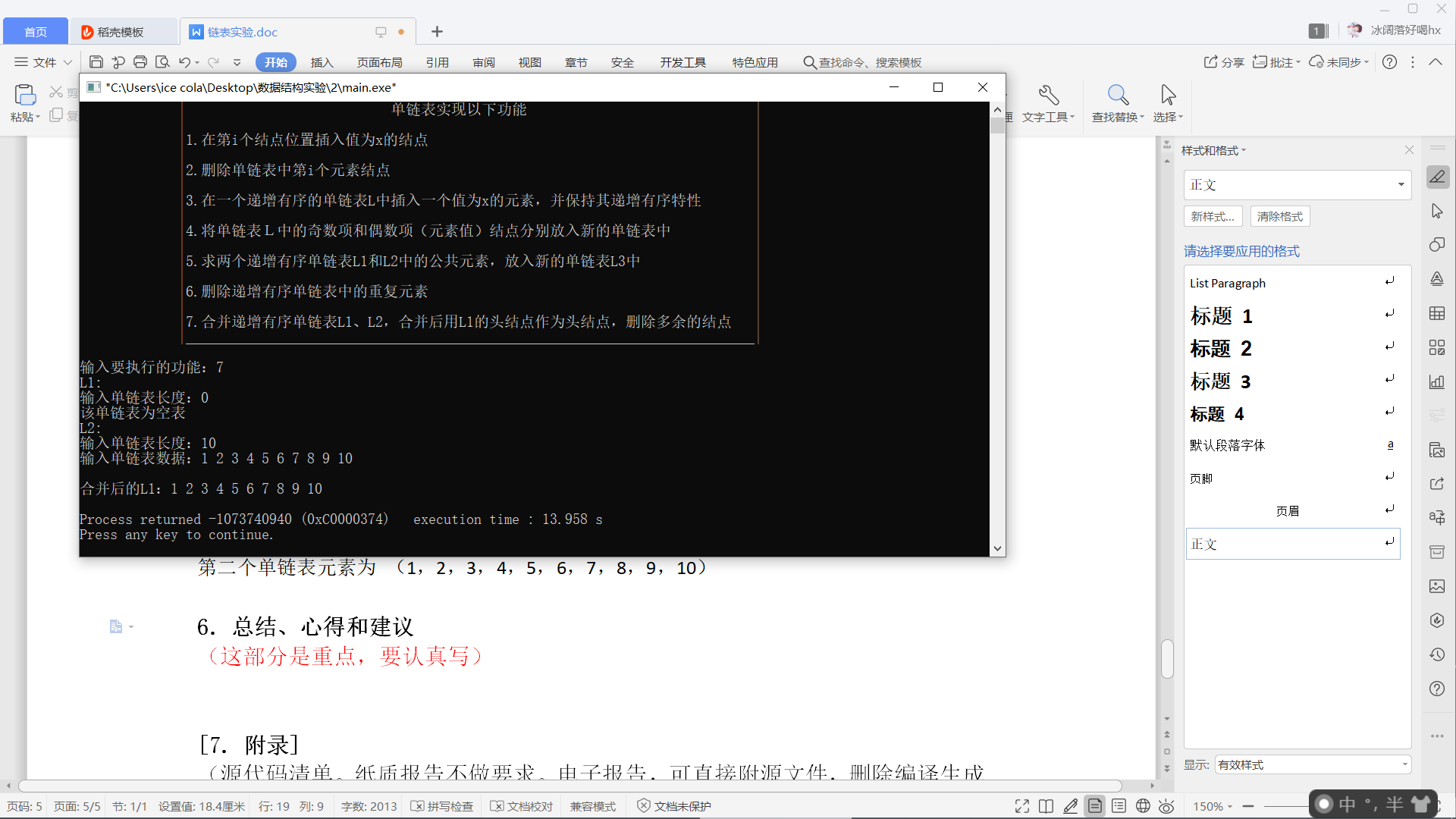
第二个单链表元素为 （2，4，5，7，8，9，12，22）



第三组

第一个单链表元素为 （）

第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）



1. **总结、心得和建议**

本次实验过程遇到了一个问题，expexted ‘}’at end of input,虽然之前遇到过这种情况，但是现在忘了，一直在报错的那一行找错误，结果一直找不到，后来查阅资料发现，应该去报错的前几行找错误，结果发现就是上两行少了一个‘}’。以后对于这种错误要牢记心中，不能再犯。

因为此次实验内容和实验一顺序表的实验内容大同小异，除了因为一些小错误导致改BUG花费了不少时间，本次实验过程整体比较顺利，希望自己继续努力，仔细、认真完成每一次实验，并在实验过程中发现平时学习的疏漏与不足！

**[8. 附录]**

（源代码清单。纸质报告不做要求。电子版报告，可直接附源文件，删除编译生成的所有文件）

