**数据结构课程设计**

**项目说明文档**

**两个有序链表序列的交集**

作 者 姓 名： 苏家铭

学 号： 2151299

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

**Tongji University**



**目录**

[1 项目分析 1](#_Toc32459)

[1.1 项目背景 1](#_Toc1034)

[1.2项目要求 1](#_Toc26071)

[1.2.1 功能要求 1](#_Toc17736)

[1.2.2 输入要求 1](#_Toc19480)

[1.2.3 输出要求 1](#_Toc27948)

[1.2.4 项目实例 1](#_Toc15000)

[2 项目设计及实现 2](#_Toc3274)

[2.1 数据结构设计思路 2](#_Toc16769)

[2.2 类设计 2](#_Toc20843)

[2.2.1 链表 2](#_Toc18353)

[2.3 项目算法 3](#_Toc8766)

[2.3.1 实现思路 3](#_Toc3110)

[2.3.2 代码实现 4](#_Toc3252)

[3 项目测试 4](#_Toc3943)

[3.1 一般情况 4](#_Toc31909)

[3.2 交集为空的情况 5](#_Toc3479)

[3.3 完全相交的情况 5](#_Toc26284)

[3.4 其中一个序列完全属于交集的情况 5](#_Toc13791)

[3.5 其中一个序列为空的情况 6](#_Toc32352)

[3.6 输入非法字符的情况（健壮性） 6](#_Toc29253)

[4 算法性能分析 7](#_Toc8015)

[4.1 正确性 7](#_Toc27442)

[4.2 可使用性 7](#_Toc1775)

[4.3 可读性 7](#_Toc3126)

[4.4 效率 7](#_Toc11991)

[4.5 健壮性 7](#_Toc15299)

# 1 项目分析

## 项目背景

交集、并集、补集、差集等概念是离散数学的集合论中非常基础而又重要的概念，而链表又是一个非常基础且被广泛应用的物理结构。实际上，在生活中求交集的情况十分常见，诸如已知每个班选修某门课的学生集合求若干个班同时选修一门课的学生集合、分别已知两个城市高于100m的建筑集合求两个城市高于100m的建筑集合等等。

本项目依托求两个集合的交集反映出工程现实中的多种求交集情形，并限定将求交集这一操作与链表相结合，通过链表存储集合中的元素，链表中的下一节点指针指向下一个元素的地址，从而实现一个个结点的顺序访问。

## 1.2项目要求

### 1.2.1 功能要求

通过链表实现，已知两个非降序链表序列S1和S2，设计函数构造出S1和S2的交集新链表S3。

### 1.2.2 输入要求

输入分2行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。

### 1.2.3 **输出要求**

在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出NULL。

### 1.2.4 项目实例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 1 2 5 -1  2 4 5 8 10 -1 | 2 5 | 一般情况 |
| 2 | 1 3 5 -1  2 4 6 8 10 -1 | NULL | 交集为空的情况 |
| 3 | 1 2 3 4 5 -1  1 2 3 4 5 -1 | 1 2 3 4 5 | 完全相交的情况 |
| 4 | 3 5 7 -1  2 3 4 5 6 7 8 -1 | 3 5 7 | 其中一个序列完全属于交集的情况 |
| 5 | -1  10 100 1000 -1 | NULL | 其中一个序列为空的情况 |

# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

根据项目要求，本例应当使用链表作为存储结构，而链表分为单链表、双链表、循环链表、静态链表，又可根据是否带附加头结点细分为更多种类。为了实现代码的可复用性，以至于能在更多场合下使用一种链表，本人实现了带附加头结点的双向循环链表，基本形式为first<-->a1<-->a2<-->...<-->an<-->first。

## 2.2 **类设计**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

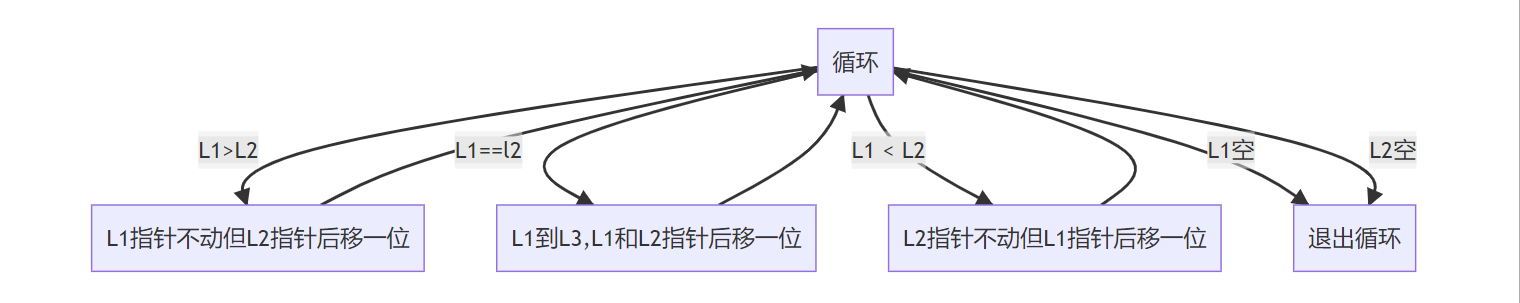
主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

## 2.3 **项目算法**

### 2.3.1 实现思路

申请一段存储空间作为结果链表的存储位置，在一个循环中使用两个指针分别指向两个链表的头结点，依次比较，循环终止条件为L1或者L2链表为空。因为题设中两个链表中的元素是非降序的，所以两个指针所指向的元素比较后只有五种情况：1、L1>L2 2、L1=L2 3、L1<L2 4、L1空 5、L2空，针对这五种情况作出五种操作：1、L1指针不动，L2指针后移一位，2、将L1中的元素复制到结果链表L3中，L1和L2指针都向后移动一位，3、L1指针后移一位，L2指针不动，4/5、退出循环。



### 2.3.2 代**码实**现

template<class T>

DblList<T> intersection(const DblList<T>&L1, const DblList<T>&L2) {

DblNode<T>\* p1, \* p2;

DblList<T> ans;

p1 = L1.getHead();

p2 = L2.getHead();//分别指向头指针

while (p1->rLink != L1.getHead() && p2->rLink != L2.getHead()) {

if (p1->rLink->data < p2->rLink->data) {

p1 = p1->rLink;

}

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

p2 = p2->rLink;

}

else {

ans.push\_back(p1->rLink->data);

p1 = p1->rLink;

p2 = p2->rLink;

}

}//end of while

return ans;

}

# 3 项目测试

## 3.1 **一般情况**

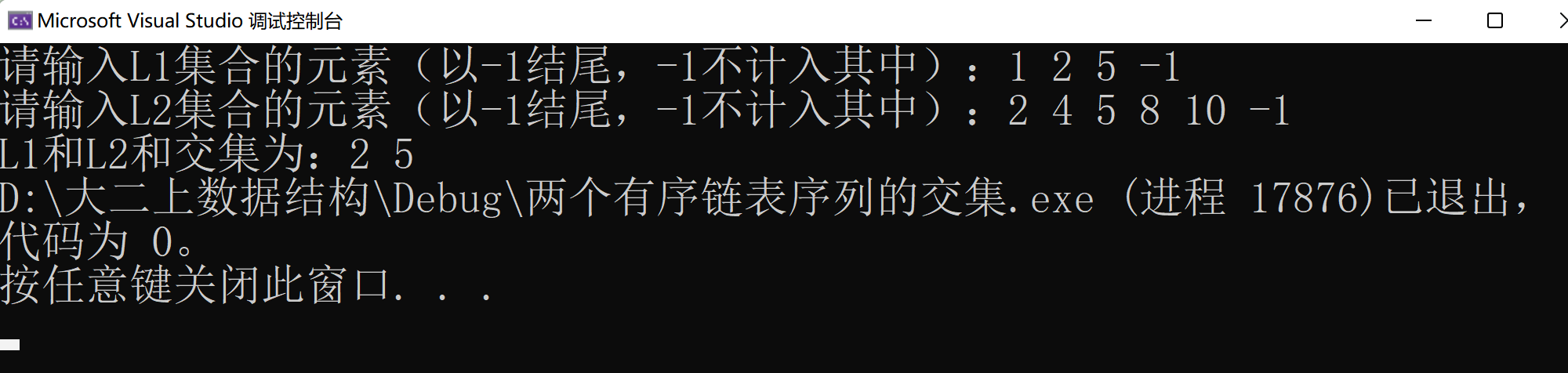
测试输入：

1 2 5 -1

2 4 5 8 10 -1

预期输出：2 5

实验结果：



## 3.2 交集为空的情况

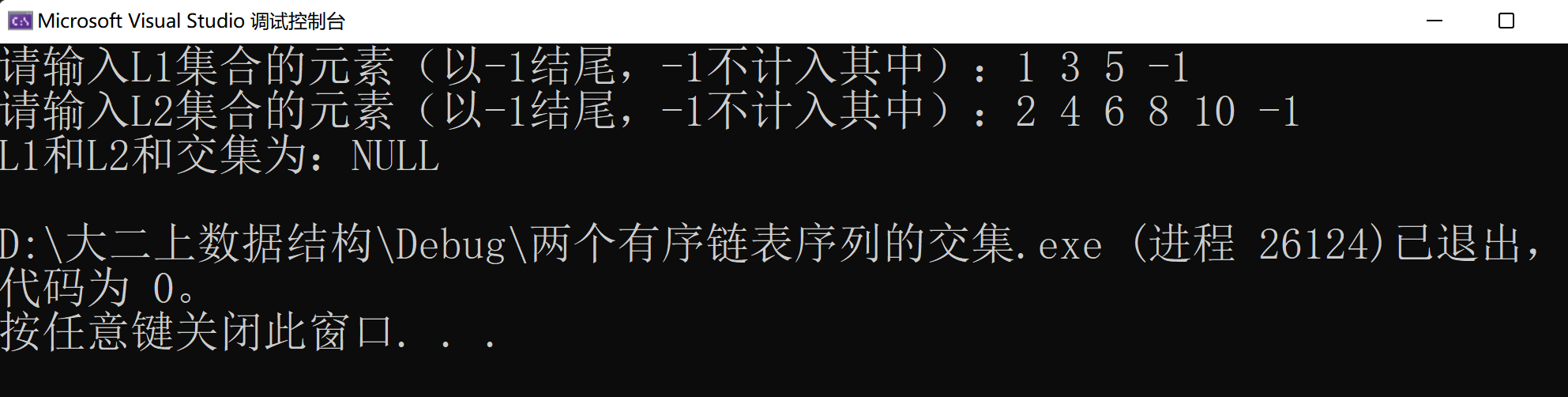
测试输入：

1 3 5 -1

2 4 6 8 10 -1

预期输出：NULL

实验结果：



## 3.3 完全相交的情况

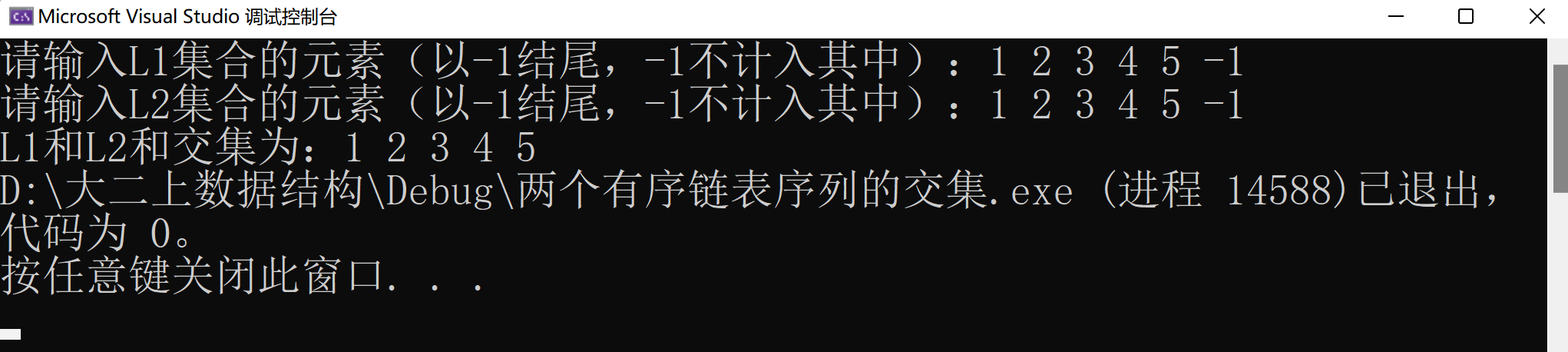
测试输入：

1 2 3 4 5 -1

1 2 3 4 5 -1

预期输出：1 2 3 4 5

实验结果：



## 3.4 其中一个序列完全属于交集的情况

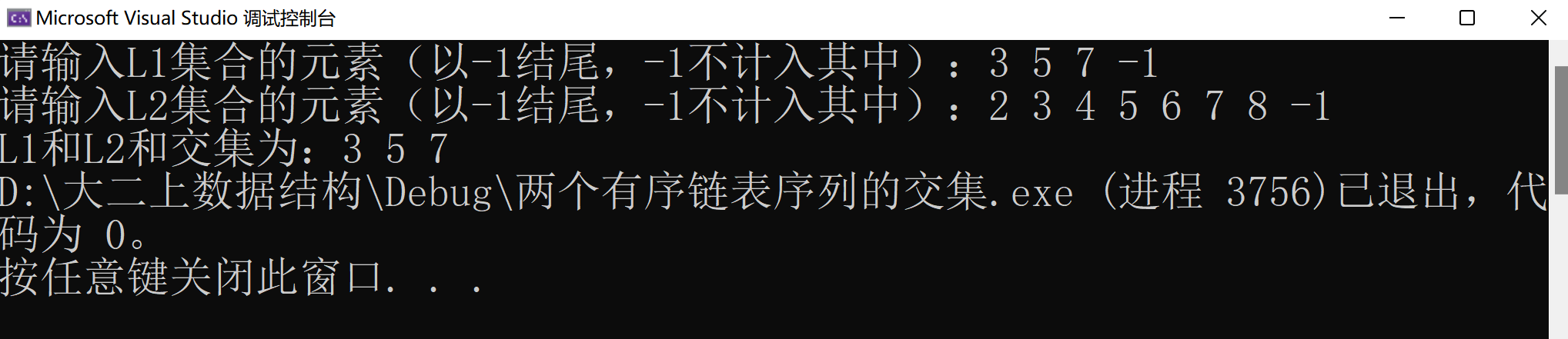
测试输入：

3 5 7 -1

2 3 4 5 6 7 8 -1

预期输出：3 5 7

实验结果：



## 3.5 其中一个序列为空的情况

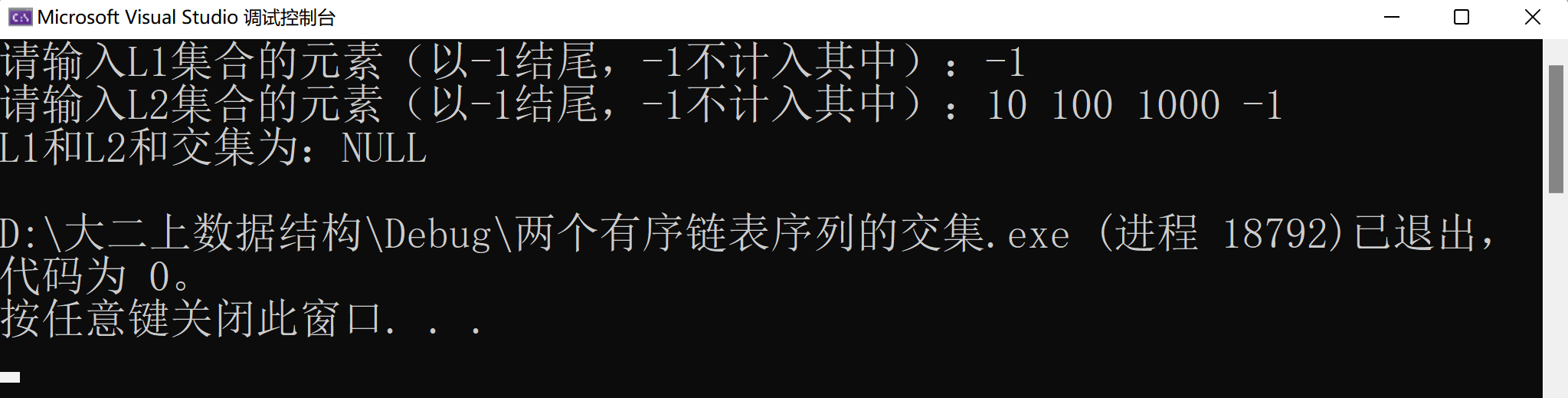
测试输入：

-1

10 100 1000 -1

预期输出：NULL

实验结果：



## 3.6 输入非法字符的情况（健壮性）

测试输入：

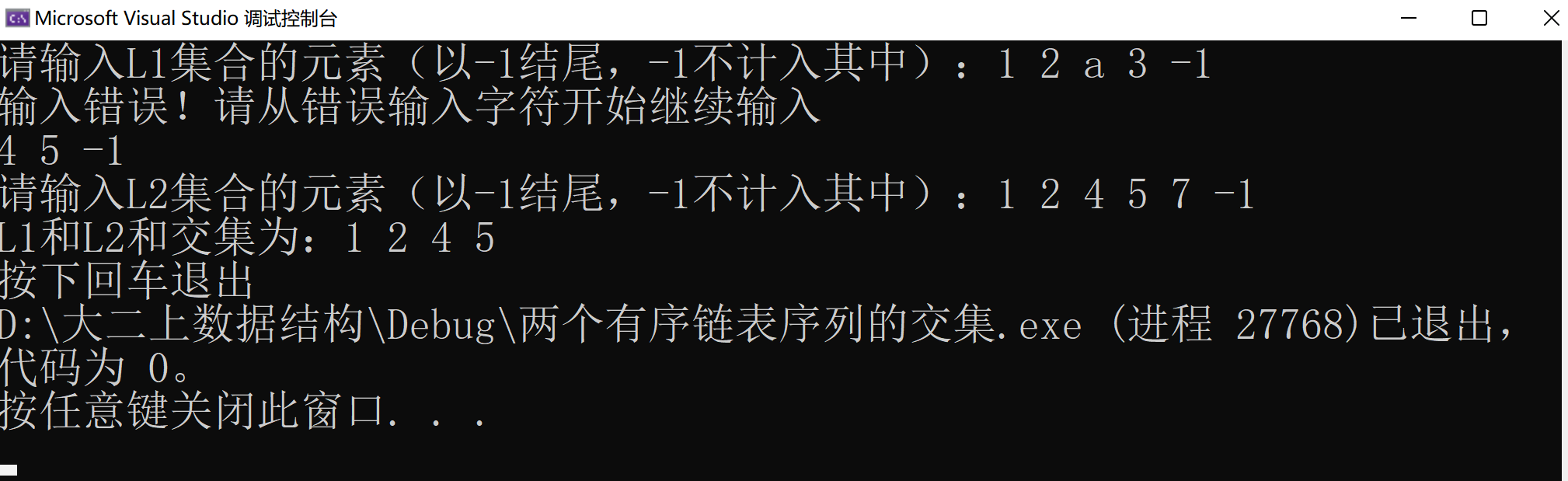
1 2 a 3 -1

错误提示后：4 5 -1

1 2 4 5 7 -1

预期输出：有错误提示+弥补措施+结果为1 2 4 5

实验结果：



# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由以上版块证明本项目与例题的五类情况比较一致，满足正确性。

## 4.2 可使用性

本算法可以很方便地使用，求交集的功能已封装在一个函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

链表的各个操作的时间复杂度已经降为O(n),且求交集的算法使用了两个指示指针，使得算法时间复杂度降为O(Min{n,m}),其中n=length(L1),m=length(L2)。

## 4.5 健壮性

本算法对于边界条件，诸如：交集为空的情况、完全相交的情况、其中一个序列完全属于交集的情况、其中一个序列为空的情况都有相应的判断，并且对于链表中新结点的申请失败、输入错误的情况也有相应的错误提示。