项目说明文档

数据结构课程设计

——两个有序链表的交集

作 者 姓 名： 张喆

学 号： 1754060

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 分析 1](#_Toc529865720)

[1.1 背景分析 1](#_Toc529865721)

[1.2 功能分析 1](#_Toc529865722)

[2 设计 2](#_Toc529865723)

[2.1 数据结构设计 2](#_Toc529865724)

[2.2 系统设计 2](#_Toc529865725)

[3 实现 3](#_Toc529865726)

[3.1 读入功能的实现 3](#_Toc529865727)

[3.1.1 读入功能流程图 3](#_Toc529865728)

[3.1.2 读入功能核心代码 4](#_Toc529865729)

[3.1.3 读入功能截屏示例 4](#_Toc529865730)

[3.2 输出功能的实现 5](#_Toc529865731)

[3.2.1 输出功能流程图 5](#_Toc529865732)

[3.2.2 输出功能核心代码 5](#_Toc529865733)

[3.2.3 输出功能截屏示例 6](#_Toc529865734)

[3.3 求交集功能的实现 7](#_Toc529865735)

[3.3.1 求交集功能流程图 7](#_Toc529865736)

[3.3.2 求交集功能核心代码 7](#_Toc529865737)

[3.3.3 求交集功能截图示例 9](#_Toc529865738)

[3.4 总体系统的实现 10](#_Toc529865739)

[3.4.1 总体系统流程图 10](#_Toc529865740)

[3.4.2 总体系统核心代码 11](#_Toc529865741)

[3.4.3 总体系统截屏示例 11](#_Toc529865742)

[4 测试 12](#_Toc529865743)

[4.1 功能测试 12](#_Toc529865744)

[4.1.1 输入功能测试 12](#_Toc529865745)

[4.1.2 交集功能测试 12](#_Toc529865746)

[4.1.3 输出功能测试 13](#_Toc529865747)

[4.2 边界测试 14](#_Toc529865748)

[4.2.1 交集为空 14](#_Toc529865749)

[4.2.2 完全相交 14](#_Toc529865750)

[4.2.3 其中一个序列完全属于交集 15](#_Toc529865751)

[4.2.4 其中有一个序列为空 16](#_Toc529865752)

[4.3 出错测试 17](#_Toc529865753)

[4.3.1 输入为非降序序列 17](#_Toc529865754)

[4.3.2 输入中首元素是负数 17](#_Toc529865755)

[4.3.3 输入中有负数 17](#_Toc529865756)

# 1 分析

## 1.1 背景分析

两个有序链表的交集运算是数据结构课程中的有关链表和集合的重要操作，当用链表表示集合的时候求交集的运算正是本题设计的算法，交集运算更是为集合之间更复杂的运算打下了基础。

当集合的元素代表比较有现实意义的元素的时候，交集的运算更是能提供一种找到两个集合共有之处的方法，对数据分析和各个行业的分析设计至关重要。随着集合中元素数量的增加，如何求两个如此大的集合的交集变得极为复杂，传统的人工计算工作量十分大而且容易出错。

随着计算机科学技术的不断成熟，使用计算机求两个集合的交集运算，具有手工管理所无法比拟的优势。这些优点能够极大地提高科研工作者和相关使用集合的行业工作者的效率，也是社会走向信息化、科学化、国际化的重要条件。因此，开发两个有序链表的交集具有十分重要的意义。

## 1.2 功能分析

做两个有序链表的交集运算，首先应该有的功能就是读入两个用户提供的非降序链表序列，并能在用户有简单的错误输入时提出警告，并允许用户重新输入。

其次，能用适当的方法正确计算两个正确链表的交集。最后，还应该能将结果的链表正确反馈给用户，让用户可以观察到交集链表。同时还应该确保软件可以正常关闭，以及如果用户输入的数据有问题的时候可以做出简单的检查和提示。

综上所述，一个有序链表的交集运算至少应该具有读入非降序链表序列、求两个链表的交集、退出、健壮性的检测等功能。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求使用链表作为数据结构储存。本设计使用C++ STL提供的list模板类作为链表结构，不仅有丰富的操作可以使用，也可以简化程度，是代码的功能更突出，可读性和可移植性更高。并使用list的iterator使操作实现起来更加优美。

## 2.2 系统设计

系统首先创建两个链表list1和list2用于储存接下来用户即将输入的两个链表序列，然后调用ListRead()函数两次，实现对两个非降序链表的初始化。

接下来设计tag位用于区分用户想要使用的算法，之后调用Intersection()求出两个非降序链表的交集。

最后输出结果序列。

在相应的部分判断用户的输入或者链表的状态是否合法，如果非法则做出相应的操作提示用户，并适当允许用于重复操作。

# 3 实现

## 3.1 读入功能的实现

### 3.1.1 读入功能流程图



### 3.1.2 读入功能核心代码

int now;

bool flag = true;

while (cin >> now)

{

if (-1 == now) { break; }

else if ((!buf.empty() && now < buf.back()) || now <= 0 )

//如果输入序列不是递减序列,或者不是正数, flag = false, 并让用户重新输入

{

flag = false;

}

else

{

buf.push\_back(now);

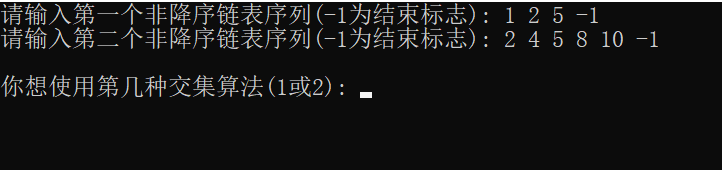
}

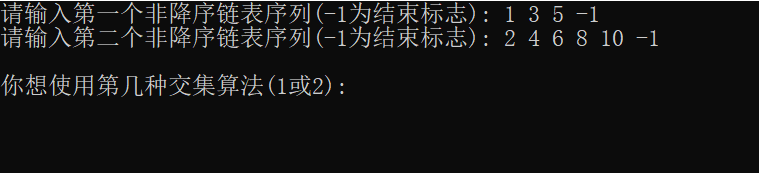
}

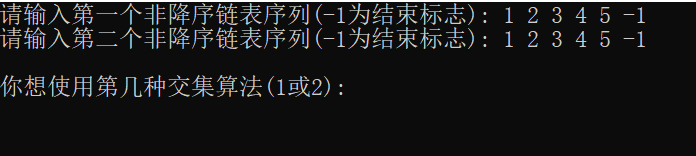
if (!flag) { buf.clear(); } //如果是非法输入, 清空当前链表

return flag;

### 3.1.3 读入功能截屏示例







## 3.2 输出功能的实现

### 3.2.1 输出功能流程图



### 3.2.2 输出功能核心代码

if (buf.empty())

//链表为空则输出NULL

{

cout << "NULL" << endl;

}

else

{

//结尾的空格不能输出,这里对最后一个元素做特殊处理

int save = buf.back();

buf.pop\_back();

list<int>::iterator iter;

for (iter = buf.begin(); iter != buf.end(); ++iter)

{

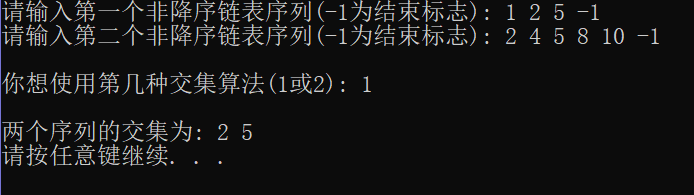
cout << \*iter << ' ';

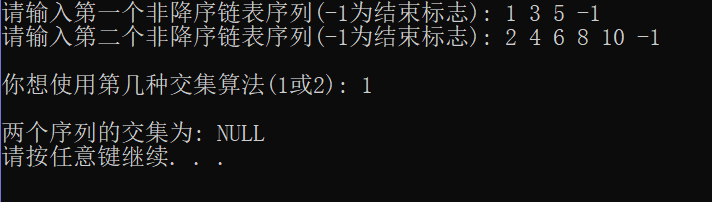
}

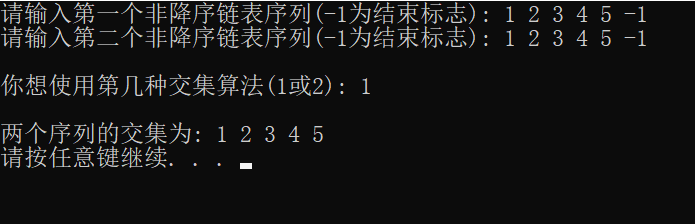
cout << save << endl;

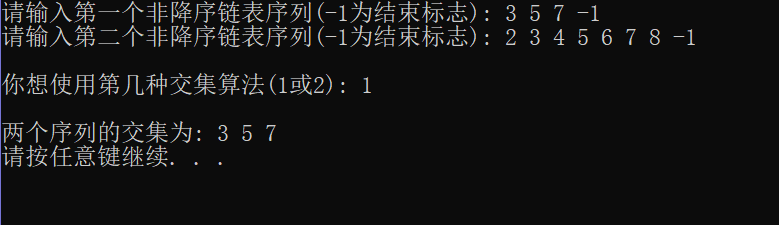
}

### 3.2.3 输出功能截屏示例









## 3.3 求交集功能的实现

### 3.3.1 求交集功能流程图



### 3.3.2 求交集功能核心代码

if (iList1.empty() || iList2.empty())

//有一个序列为空, 交集一定为空

{

resList.clear();

return;

}

else

{

if (1 == tag)

//第一种算法

//在第一个链表中循环一遍,对于每一个元素,如果在第二个链表中出现

//则加入结果链表, 并在第二个链表中删除该元素

{

list<int>::iterator iter1, iter2;

for (iter1 = iList1.begin(); iter1 != iList1.end(); ++iter1)

{

iter2 = find(iList2.begin(), iList2.end(), \*iter1);

if (iter2 != iList2.end()) //如果在第二个链表中找到了

{

resList.push\_back(\*iter1);

iList2.erase(iter2, ++iter2);

}

}

}

else if (2 == tag)

//第二种算法

//使用两个迭代器分别指向两个链表, 比较两个迭代器的值并做相应操作

{

list<int>::iterator iter1 = iList1.begin(), iter2 = iList2.begin();

//使用两个迭代器分别指向两个链表的头

while (iter1 != iList1.end() && iter2 != iList2.end())

//任何一个链表到头后另一个链表的剩余部分不需要再进行操作,一定不会出现在结果链表中

{

if (\*iter1 == \*iter2)

//如果两项相等则加入结果链表中

//并把两个迭代器各自向后移一格

{

resList.push\_back(\*iter1);

++iter1;

++iter2;

}

else if (\*iter1 < \*iter2)

//如果第一个链表中的当前元素较小则把第一个迭代器后移一格

{

++iter1;

}

else

//如果第二个链表中的当前元素较小则把第二个迭代器后以一格

{

++iter2;

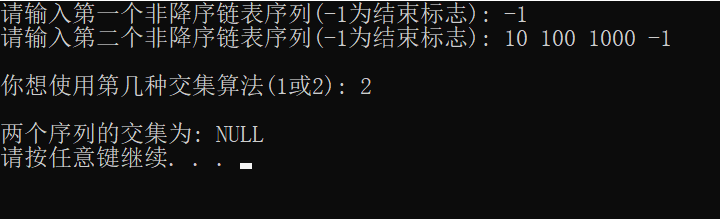
}

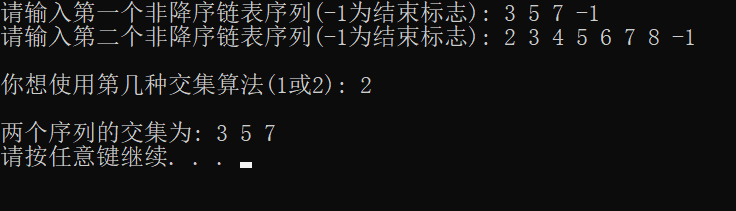
}

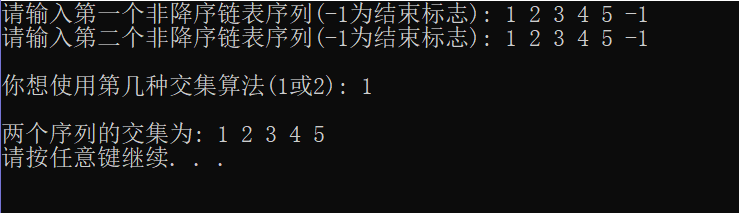
}

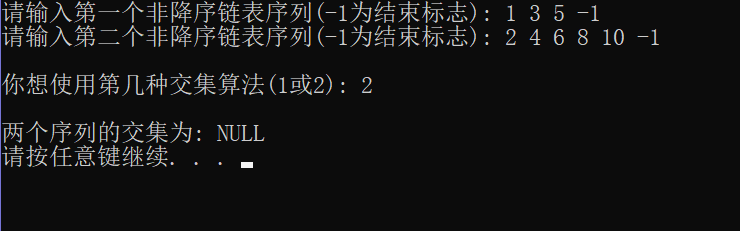
}

### 3.3.3 求交集功能截图示例









## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统核心代码

/\*读入两个非降序链表序列\*/

list<int> iList1, iList2;

printf("请输入第一个非降序链表序列(-1为结束标志): ");

while (!ListRead(iList1))

{

printf("输入非法, 请重新输入非降序序列: ");

}

printf("请输入第二个非降序链表序列(-1为结束标志): ");

while (!ListRead(iList2))

{

printf("输入非法, 请重新输入非降序序列: ");

}

/\*交集运算\*/

int tag;

printf("\n你想使用第几种交集算法(1或2): ");

while (cin >> tag)

{

if (tag == 1 || tag == 2) { break; }

else

{

printf("输入非法, 请重新输入: ");

}

}

list<int> resList;

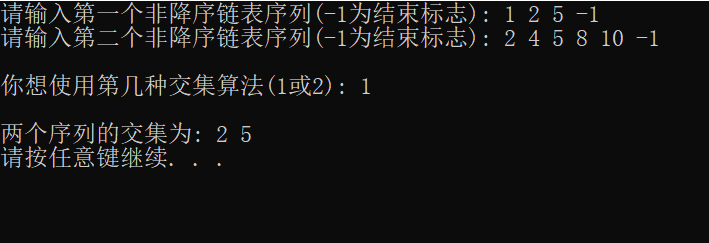
Intersection(iList1, iList2, resList, tag);

/\*输出结果\*/

cout << "\n两个序列的交集为: ";

DisplayList(resList);

### 3.4.3 总体系统截屏示例



# 4 测试

## 4.1 功能测试

### 4.1.1 输入功能测试

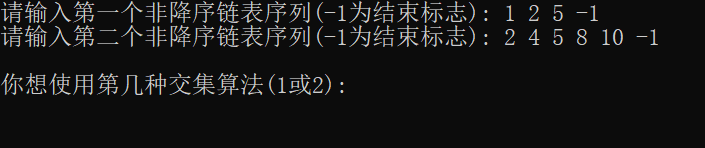
**输入功能测试用例**：1 2 5 -1

2 4 5 8 10 -1

**预期结果**：

弹出“你想使用第几种交集算法（1或2）”询问输入

**实验结果:**

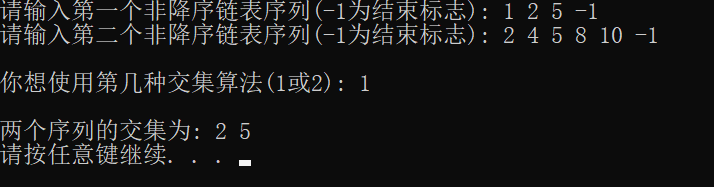


### 4.1.2 交集功能测试

**第一种交集算法**

**预期结果：**2 5

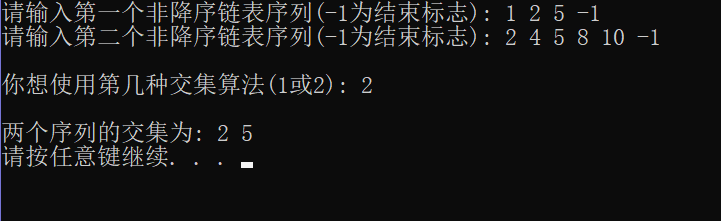
**实验结果：**



**第二种交集算法**

**预期结果：**2 5

**实验结果：**

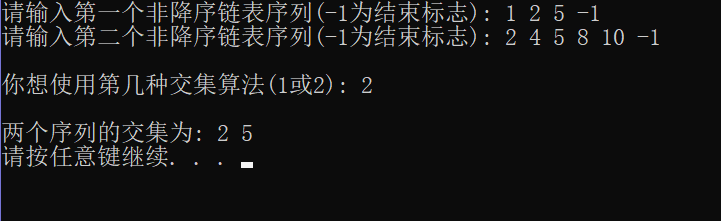


### 4.1.3 输出功能测试

**预期结果：**

2 5

**实验结果：**



## 4.2 边界测试

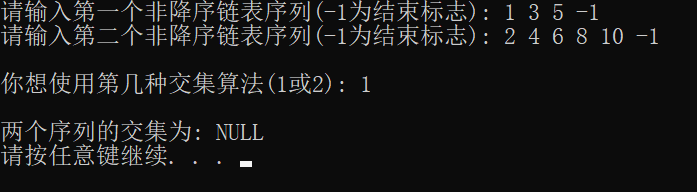
### 4.2.1 交集为空

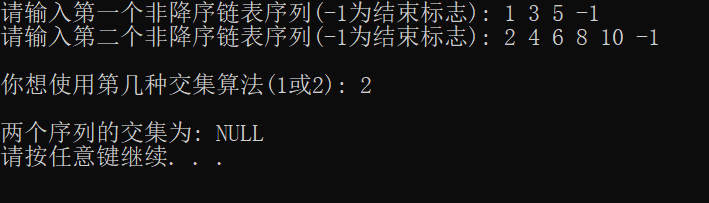
**测试用例：**1 3 5 -1

2 4 6 8 10 -1

**预期结果：输出NULL**

**实验结果：**





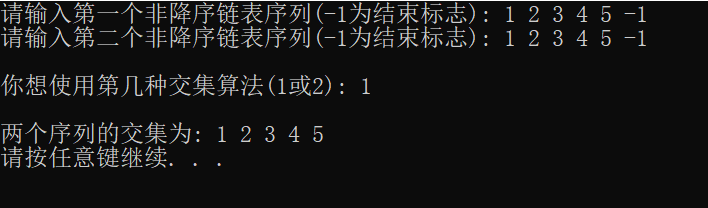
### 4.2.2 完全相交

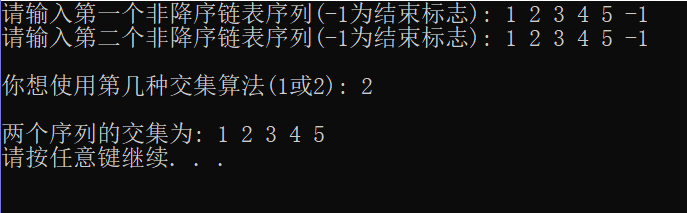
**测试用例：**1 2 3 4 5 -1

1 2 3 4 5 -1

**预期结果：**输出1 2 3 4 5

**实验结果：**





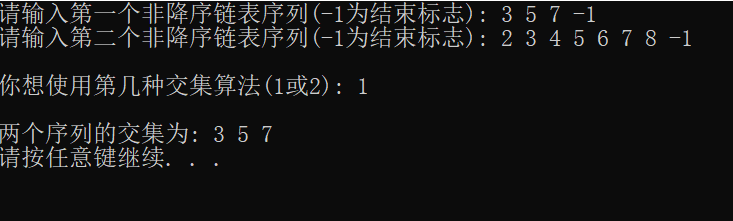
### 4.2.3 其中一个序列完全属于交集

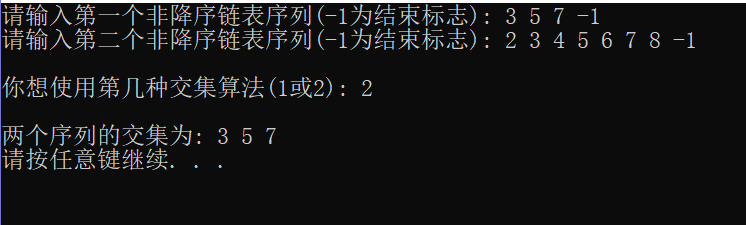
**测试用例：**3 5 7 -1

2 3 4 5 6 7 8 -1

**预期结果：**输出3 5 7

**实验结果：**





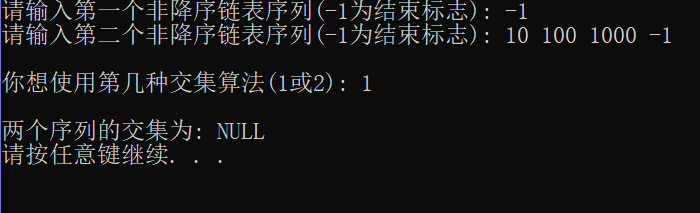
### 4.2.4 其中有一个序列为空

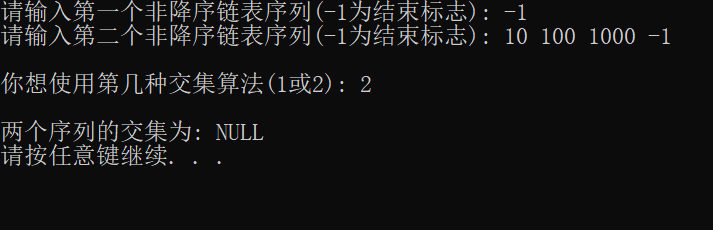
**测试用例：**-1

10 100 1000 -1

**预期结果：**输出NULL

**实验结果：**





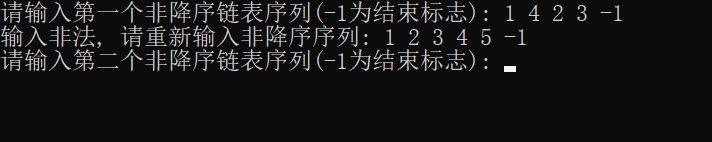
## 4.3 出错测试

### 4.3.1 输入为非降序序列

**测试用例：**1 4 2 3 -1

**预期结果：**给出错误提示，程序运行正常不崩溃，并提示用户重新输入。

**实验结果：**

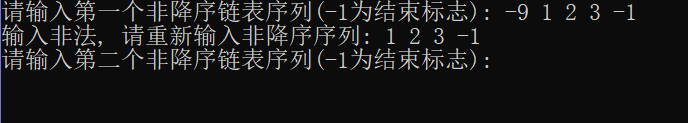


### 4.3.2 输入中首元素是负数

**测试用例：**-9 1 2 3 -1

**预期结果：**程序正常运行，不崩溃，并提示用户重新输入。

**实验结果：**



### 4.3.3 输入中有负数

**测试用例：**-9 -3 1 2 3 -1

**预期结果：**程序正常运行，不崩溃，并提示用户重新输入。

**实验结果：**

