**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： 张梓瀚

学 号： 2051943

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 - 3 -](#_Toc121494746)

[1.1项目背景 - 3 -](#_Toc121494747)

[1.2 项目需求分析 - 3 -](#_Toc121494748)

[1.3 项目要求 - 3 -](#_Toc121494749)

[1.3.1 功能要求 - 3 -](#_Toc121494750)

[1.3.2 输入格式 - 3 -](#_Toc121494751)

[1.3.3 输出格式 - 3 -](#_Toc121494752)

[1.3.4 项目示例 - 4 -](#_Toc121494753)

[2 项目设计 - 4 -](#_Toc121494754)

[2.1 数据结构设计 - 4 -](#_Toc121494755)

[2.2 类设计 - 4 -](#_Toc121494756)

[2.2.1 结点类（ListNode） - 4 -](#_Toc121494757)

[2.2.2 链表类（List） - 4 -](#_Toc121494758)

[2.3 求链表交集算法 - 5 -](#_Toc121494759)

[2.3.1 主要思想 - 5 -](#_Toc121494760)

[2.3.2代码 - 6 -](#_Toc121494761)

[2.3.3 流程图 - 7 -](#_Toc121494762)

[3 项目测试 - 8 -](#_Toc121494763)

[3.1 一般情况 - 8 -](#_Toc121494764)

[3.2 交集为空的情况 - 8 -](#_Toc121494765)

[3.3 完全相交的情况 - 8 -](#_Toc121494766)

[3.4 其中一个序列完全属于交集的情况 - 8 -](#_Toc121494767)

[3.5 其中一个序列为空的情况 - 9 -](#_Toc121494768)

[3.6 两个链表序列均为空的情况 - 9 -](#_Toc121494769)

# 1 项目分析

## 1.1项目背景

链表是数据结构中极其重要的一种存储结构（也即物理结构），因此掌握对于链表的各种操作相当重要。

需要注意到的是：链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。因此，对于链表的各种操作不像数组能够实现随机访问，而是要通过一个一个节点依次访问到下一个节点。

在工程中，常常需要应用到链表存储数据。也正是因为这一原因，有时候我们需要求两个不同对象中的公共元素，这个时候往往就可以通过求链表中各个元素的交集来实现。

## 1.2 项目需求分析

针对于求两个有序链表交集的系统，应当满足以下需求：

* 功能完善

系统应当考虑到两个链表内元素的各种情况，如：链表S1长度大于链表S2，链表S1长度等于S2等。

* 执行效率高

对于链表内节点数量比较多的情况，系统也应当具有比较快的处理速度，即时间复杂度应当较低。

* 健壮性

系统应当考虑当所输入的链表为空的情况。

## 1.3 项目要求

### 1.3.1 功能要求

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。

### 1.3.2 输入格式

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。

### 1.3.3 输出格式

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。

### 1.3.4 项目示例

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

# 2 项目设计

## 2.1 数据结构设计

如上述功能分析所述，该项目本质上是要实现一个链表数据结构，并在此结构上实现两个有序链表的交集算法。链表是计算机程序中常用的数据结构，根据节点之间连接方式和存储方式细分，有单链表、双向链表、循环链表、静态链表等种类。在本系统中，封装了一个带头结点的链表，以实现所需要的功能。

## 2.2 类设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统利用了上题中实现过的**链表结点类**与**链表类**。

### 2.2.1 结点类（ListNode）

链表的结点中存储的数据有链表数据data和后继结点next。为提高代码的泛用性，本题中ListNode类采用模板类实现。

### 2.2.2 链表类（List）

链表的实现原理大同小异，不同之处在于：是否带头结点、是否带尾结点、每一个结点是否带前驱结点等。为了使得链表中各种操作的时间复杂度都尽可能低，因此这里选择了带头结点的链表来实现链表中的各种操作，使得头结点的操作与其他结点相同，从而简化了代码结构。

template <typename T>

class List

{

private:

ListNode<T>\* start;

int size = 0;

inline ListNode<T>\* findLastByData(const T& tar);

inline ListNode<T>\* findLastByPos(int pos);

public:

//默认构造函数

List();

//析构函数

~List();

//获取头结点

ListNode<T>\* get\_head();

//查找指定位置的结点

ListNode<T>\* Locate(int pos);

//向链表尾部插入结点

inline void insert(const T& s);

//向特定位置插入结点

inline bool insert(const T& s, int pos);

//根据数据内容查找结点

inline ListNode<T>\* findByData(const T& s);

//根据在链表中的位置查找结点

inline ListNode<T>\* findByPos(int tarID);

//修改结点数据

inline void modify(T tar, const T& new\_s);

//删除指定位置的结点

inline void remove(int tarID);

//删除指定内容的结点

inline void remove(const T& tar);

//按序打印链表所有结点

inline void printAll(ostream& os = cout);

//获取链表长度

int get\_list\_size();

};

## 2.3 求链表交集算法

### 2.3.1 主要思想

初始化结果链表为空，分别定义两个链表上的迭代器1和2。

当两个迭代器均不指向链表末尾时进行循环：比较迭代器1和迭代器2所指向的元素，若两个元素相等，则将该元素插入到结果链表末端，两个迭代器加1；否则只将元素较小的迭代器加1。

### 2.3.2代码

template<class T>

void get\_intersection(const List<T>& l1, const List<T>& l2, List<T>& result)

{

auto it1 = l1.get\_head();

auto it2 = l2.get\_head();

while (it1 != nullptr && it2 != nullptr)

{

int num1 = it1->get\_data(), num2 = it2->get\_data();

if (num1 < num2)

it1 = it1->get\_next();

else if (num1 > num2)

it2 = it2->get\_next();

else

{

result.insert(num1);

it1 = it1->get\_next();

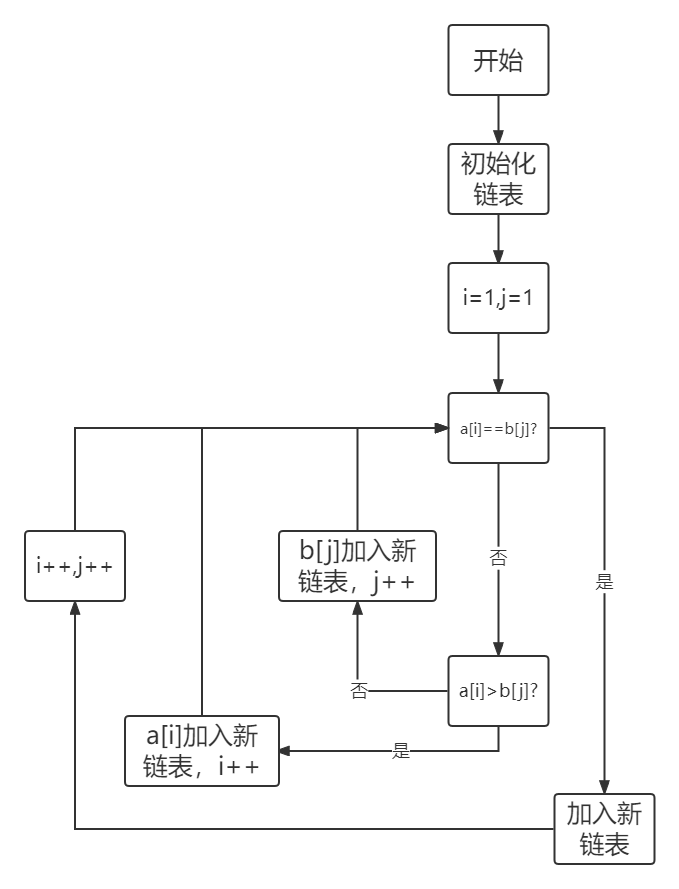
it2 = it2->get\_next();

}

}

}

### 2.3.3 流程图



# 3 项目测试

## 3.1 一般情况

测试用例：1 2 5 -1 2 4 5 8 10 -1

预期结果：2 5

实验结果：

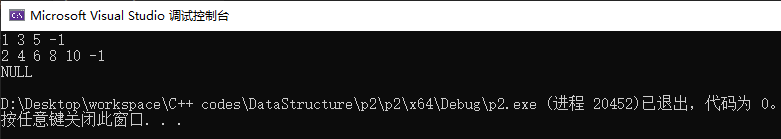


## 3.2 交集为空的情况

测试用例：1 3 5 -1 2 4 6 8 10 -1

预期结果：NULL

实验结果：

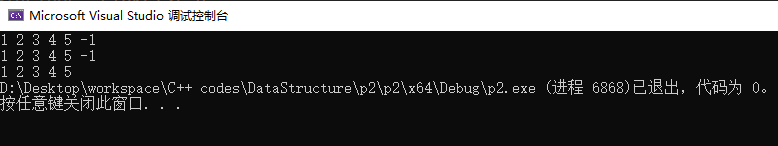


## 3.3 完全相交的情况

测试用例：1 2 3 4 5 -1 1 2 3 4 5 -1

预期结果：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

测试用例：3 5 7 -1 2 3 4 5 6 7 8 -1

预期结果：3 5 7

实验结果：



## 3.5 其中一个序列为空的情况

测试用例：-1 10 100 1000 -1

预期结果：NULL

实验结果：



## 3.6 两个链表序列均为空的情况

测试用例：-1 -1

预期结果：NULL

实验结果：

