数据结构项目文档

> 题目：两个有序链表序列的交集
>
> 指导教师：张颖
>
> 姓名：王亮
>
> 学号：1653340

一、题目分析

1. 项目简介

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。

2. 功能需求

1. 输入说明：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。
2. 输出说明：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。
3. 测试用例：

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

3. 设计思路

首先确定适合解决此题目的数据结构为链表，实现数据结构相关的类，然后实现项目中需要的求交集功能，最后在主函数中运行验证即可。

二、设计

1. 数据结构设计

题目中已经明确要求使用链表实现对有序序列的存储，且由于单链表便于进行按序访问结点，且增删元素方便，适合完成有序序列求交集的任务，因此数据结构选择单链表即可。

2. 类的设计

实现一个链表需要 LinkNode 和 LinkList 两个类，采用复合的方式进行两个类的定义。LinkNode 结点中的数据域保存 int 型数据。

1.1 LinkNode

链表结点包含数据域和指针域，数据域保存int型数据。

提供无参和有参两个构造函数。

提供对数据域和指针域的公共访问函数。

```
class LinkNode
{
    friend class LinkList;    //声明LinkList类为友元类

private:
    int data;                 //数据元素域
    LinkNode* next;           //链指针域

public:
    LinkNode()               //无参构造
    {
        this->data = 0;
        this->next = nullptr;
    }

    LinkNode(const int& input, LinkNode* ptr = nullptr)     //有参构造
    {
        this->data = input;
        this->next = ptr;
    }

    LinkNode* getNext() const    //返回结点指针域的值
    {
        return next;
    }

    int getData() const          //返回结点数据域的值
    {
        return data;
    }

};
```

1.2 LinkList

链表包含头指针、尾指针、结点计数共三个成员数据。

主要实现了后插法建立链表、向链表尾部插入数据、输出链表等成员函数，用于完成求交集的功能需求。

```
class LinkList
{
private:
    LinkNode* first;          //链表头指针
    LinkNode* last;           //链表尾指针
```

```
    int cntNode;                //结点计数

public:
    LinkList(){first = new LinkNode(); last = first; cntNode = 0;}       //
无参构造函数
    ~LinkList(){makeEmpty(); delete first;}                               //析
构函数

    void makeEmpty();                                                     //将链表
置为空表
    LinkNode* getHead() const {return first;}                             //获取链
表头

    void init();                                                          //后插法
建立链表
    void insertToTail(int newData);                                       //向链表
尾部插入数据
    void output() const;                                                  //输出

};
```

3. 函数设计

3.1 交集函数

类似数据结构课上讲的利用链表完成多项式加法，我们同样需要有三个链表结点指针，分别指向两个参与交集运算的链表和结果链表（初始为空）的头部，设三个指针分别为p1，p2，p3。

比较p1和p2两个指针指向值的大小，如果两数不相等，则应将值小的数的指针后移；如果两数相等，则此数属于两个有序序列的交集部分，应将这个数字链入结果链的尾部，三个指针都向后移移位。直到其中一个序列所有元素全部比较完。

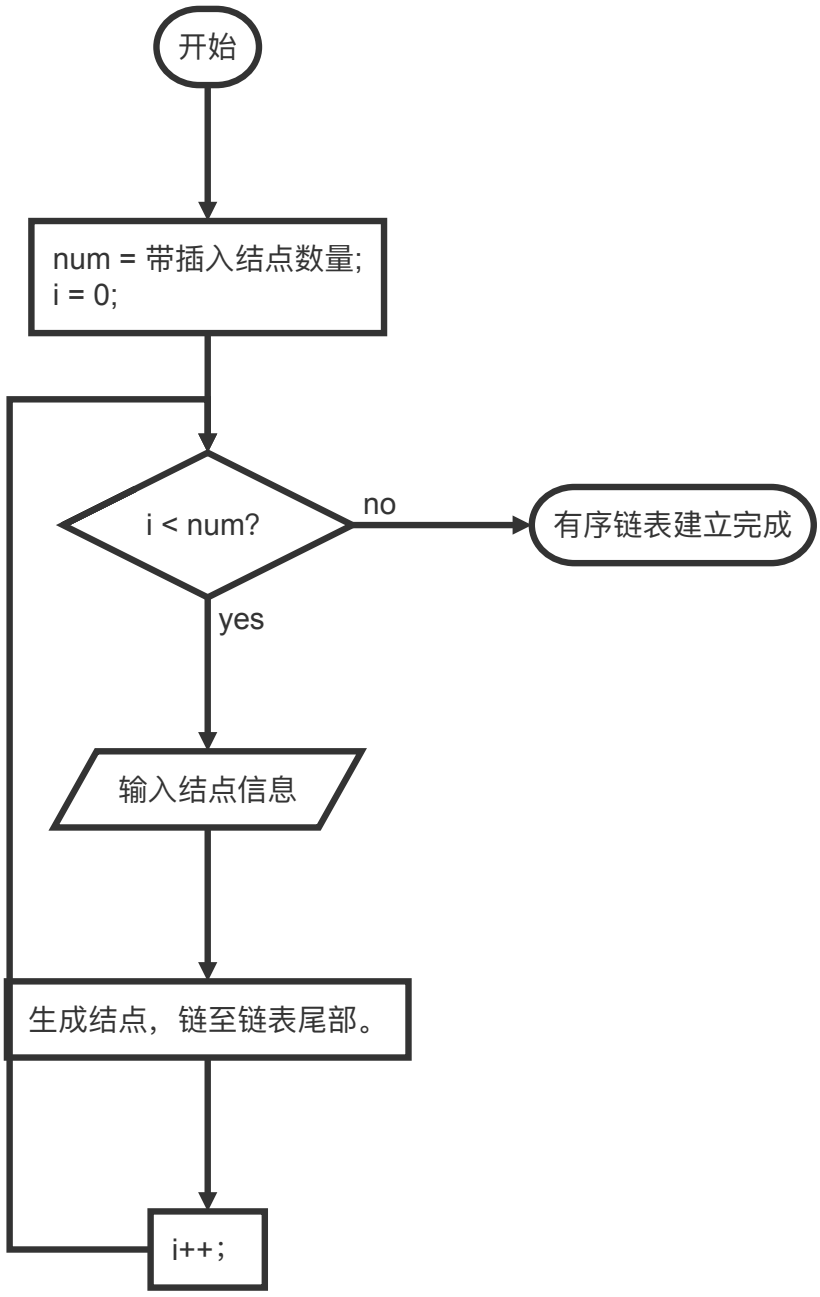
3.2 主函数

创建三个链表，其中两个链表用于存放输入的两个有序序列，然后调用求交集的函数，将结果存放在第三个链表中，最后输出结果链的内容。

三、实现

1. 尾插法建立链表

1.1 流程图

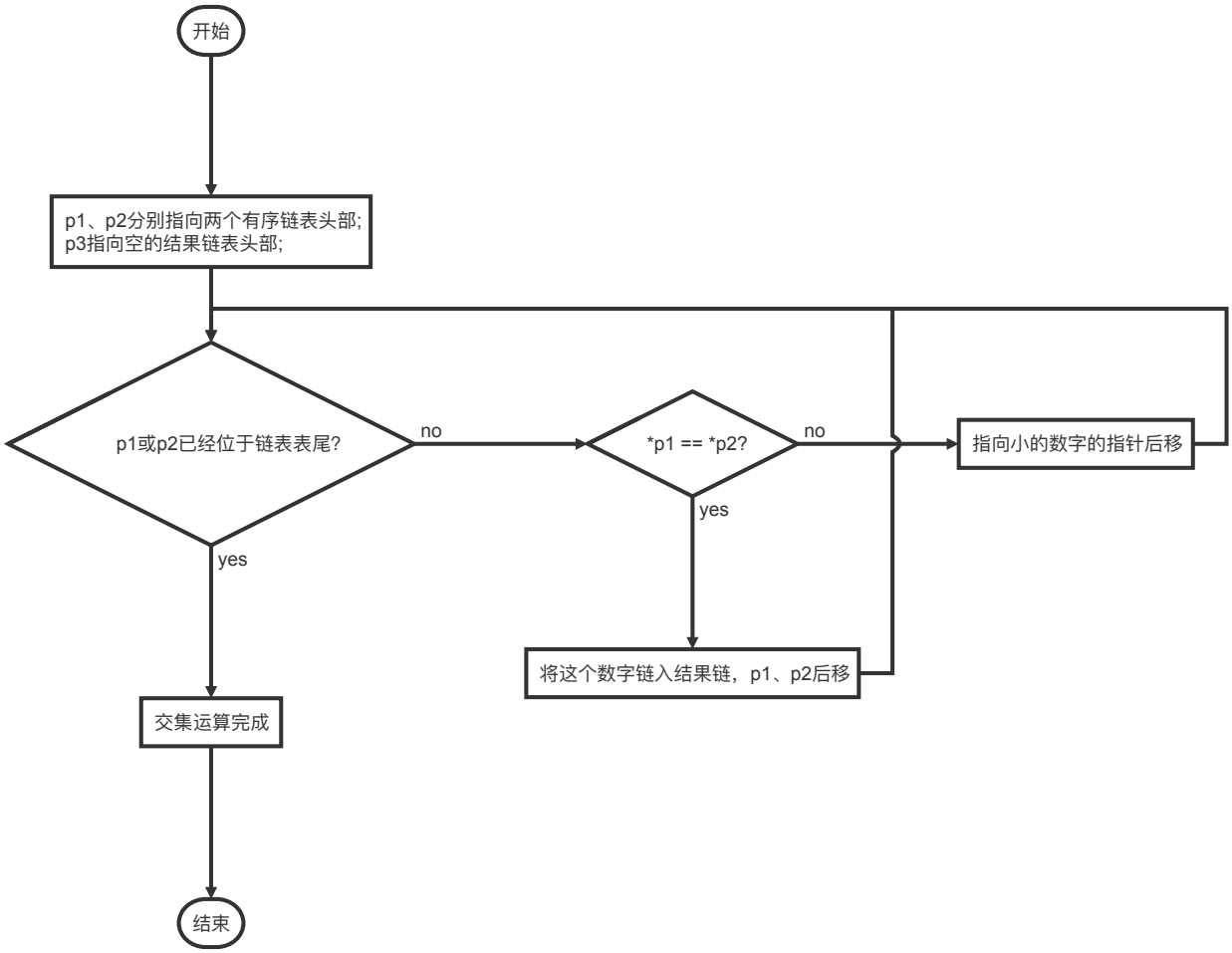


1.2 关键代码

```
void LinkList::init()
{
    int temp;
    std::cin>>temp;
    while(temp != -1)
    {
        insertToTail(temp);
        std::cin>>temp;
    }
}
```

2. 求交集函数

2.1 流程图



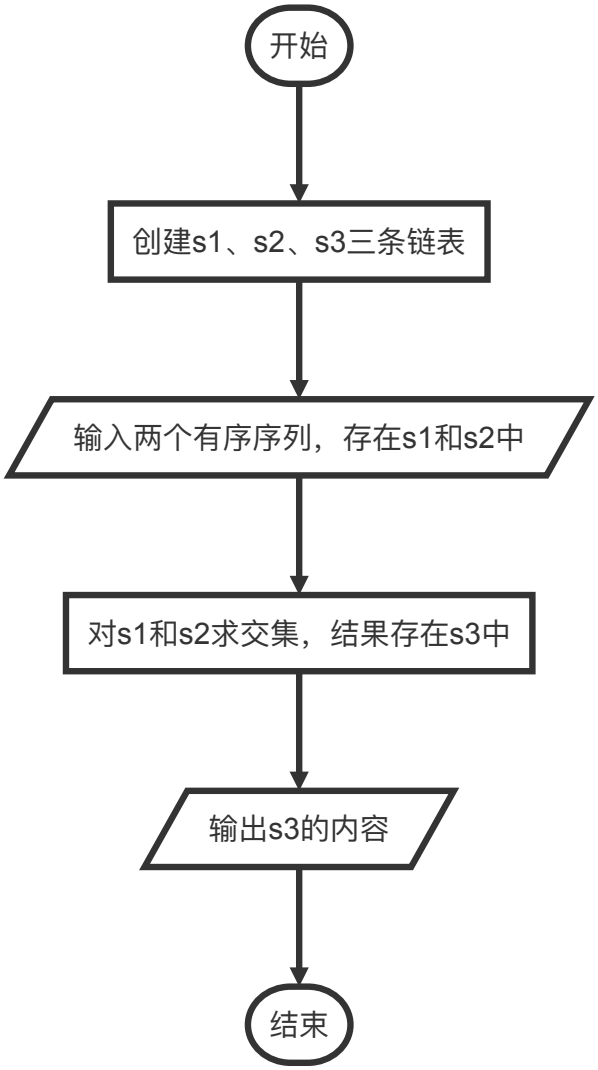
2.2 关键代码

```
/*
 @brief:求两个链表的非降序链表的交集，并保存在链表result中
 */
```

```
void intersection(LinkList* input1, LinkList* input2, LinkList* result)
{
    LinkNode* p1 = input1->getHead()->getNext();
    LinkNode* p2 = input2->getHead()->getNext();

    while(p1!=nullptr && p2!=nullptr)
    {
        if(p1->getData() == p2->getData())
        {
            result->insertToTail(p1->getData());
            p1 = p1->getNext();
            p2 = p2->getNext();
        }
        else if(p1->getData() > p2->getData())
        {
            p2 = p2->getNext();
        }
        else
        {
            p1 = p1->getNext();
        }
    }

}
```

3. 主函数

3.1 流程图



3.2 关键代码

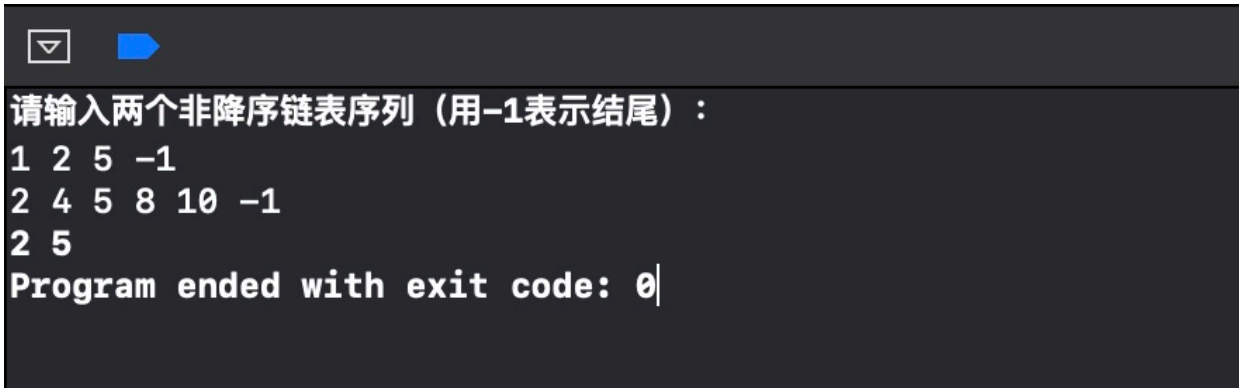
```
int main(int argc, const char * argv[])
{
    LinkList* s1= new LinkList();
    LinkList* s2= new LinkList();
    LinkList* s3= new LinkList();

    std::cout<<"请输入两个非降序链表序列（用-1表示结尾）："<<std::endl;
    s1->init();
    s2->init();

    intersection(s1, s2, s3);

    s3->output();

    delete s1;
    delete s2;
    delete s3;

    return 0;
```

```
}
```

四、测试

1. 功能测试（包括边界情况）

1.1 一般情况



测试结果正确。

1.2 交集为空的情况

输出为NULL，测试结果正确。

1.3 完全相交的情况

测试结果正确。

1.4 其中一个序列完全属于交集的情况

测试结果正确。

1.5 其中一个序列为空的情况

输出为NULL，测试结果正确。