****

项目说明文档

**数据结构课程设计**

**——两个有序链表序列的交集**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 功能分析 2](#_Toc58716126)

[第2章 设计 3](#_Toc58716127)

[2.1 数据结构设计 3](#_Toc58716128)

[2.2 类结构设计 3](#_Toc58716129)

[2.3 成员与操作设计 3](#_Toc58716130)

[2.4 系统设计 4](#_Toc58716131)

[第3章 设计 5](#_Toc58716132)

[3.1 求交集功能的实现 5](#_Toc58716133)

[3.1.1 求交集功能流程图 5](#_Toc58716134)

[3.1.2 求交集功能核心代码 6](#_Toc58716135)

[3.1.3 求交集功能截屏示例 6](#_Toc58716136)

[3.2 总体系统的实现 7](#_Toc58716137)

[3.2.1 总体系统流程图 7](#_Toc58716138)

[3.2.3 总体功能截屏示例 8](#_Toc58716139)

[第4章 测试 9](#_Toc58716140)

[4.1 功能测试 9](#_Toc58716141)

[4.1.1 插入功能测试 9](#_Toc58716142)

[4.2 边界测试 9](#_Toc58716143)

[4.2.1 输入空链表 9](#_Toc58716144)

[4.2.2 链表无交集 10](#_Toc58716145)

[4.3 出错测试 11](#_Toc58716146)

[4.3.1 输入无序序列 11](#_Toc58716147)

[4.3.2 输入负数序列 11](#_Toc58716148)

[4.3.3 输入非数值序列 11](#_Toc58716149)

# 第1章 功能分析

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。要求采用链表

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。

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。

# 第2章 设计

## 2.1 数据结构设计

如上所述，要求使用链表数据结构。同时，为了实现简易，在第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。为了满足各种各样的输入类型需求，使用string型作为实例化List模板。

## 2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（Node）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。为方便处理，本系统采用模板struct结构体描述链表结点类（Node<T>），这样使得模板链表结点类（List<T>）可以访问链表结点。

## 2.3 成员与操作设计

**模板链表节点结构体（Node）**

template <class T>

struct Node {

    T \_data;      //数据

    Node \*\_next;  //指针域

    Node();

};

**模板链表类（Node）**

template <class T>

class List {

   private:

    Node<T> \*\_head;  //表头结点

    int \_len;        //链表长度

   public:

    T &operator[](int i); //直接取出第n个节点的数据

    void ListIns(int n, T data);  //将数据插入第n个节点中

    T ListPop(T data);    //查找链表中的数据返回并删除节点

    int ListFind(T id);   //查找数据在链表中的位置

    int ListLen() const;  //返回链表长度

    bool Empty() const;   //返回是否为空

    void ListChange(int n, T data);      //修改链表第n个节点的值

    std::string ListPrint(int n) const;  //将链表输出

    List();

    ~List();

};

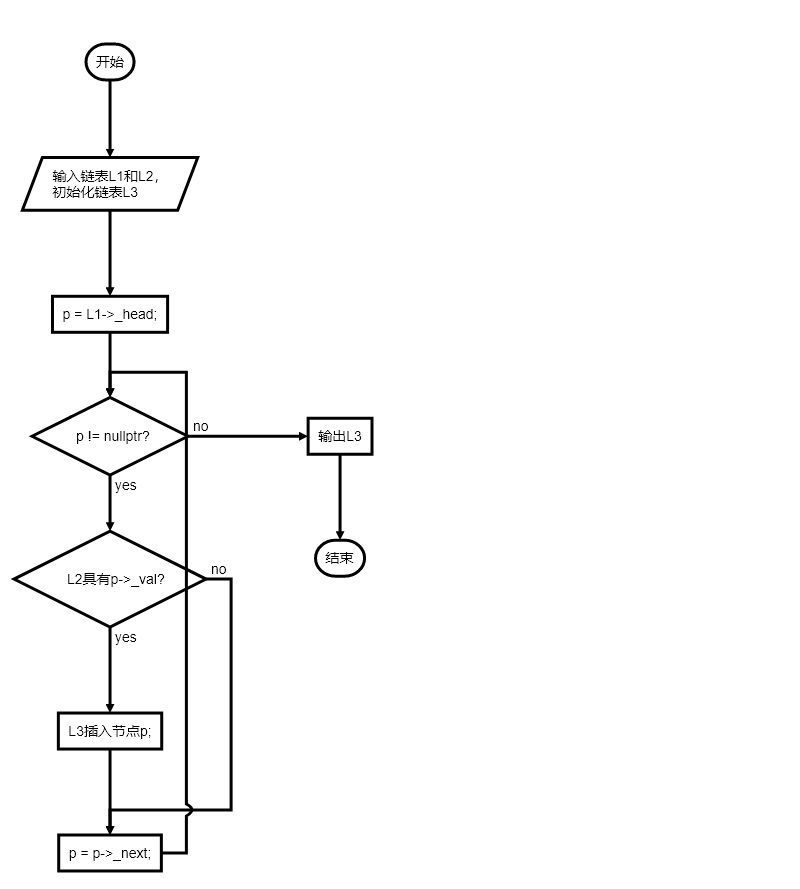
## 2.4 系统设计

系统首先调用opening ()函数实现对屏幕的初始化，使用insert ()函数完成对链表L1和L2的创建和输入数据工作，然后通过findIntersection ()函数计算出L1和L2的交集L3，并使用print ()函数输出。

# 第3章 实现

## 3.1 求交集功能的实现

### 3.1.1 求交集功能流程图



### 3.1.2 求交集功能核心代码

template <class T>

List<T> findIntersection(List<T> &list\_a, List<T> &list\_b) {

    List<T> list\_c;

    for (int i = 0; i < list\_a.ListLen(); i++) {

        if (list\_b.ListFind(list\_a[i]) != -1)

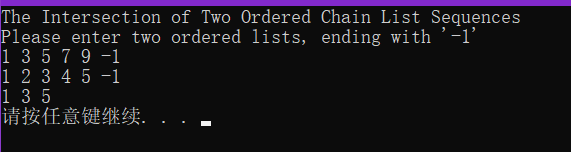
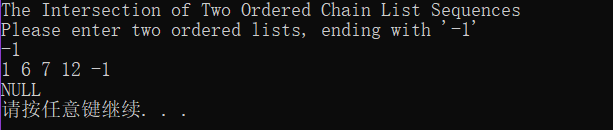
            list\_c.ListIns(list\_c.ListLen(), list\_a[i]);

    }

    return list\_c;

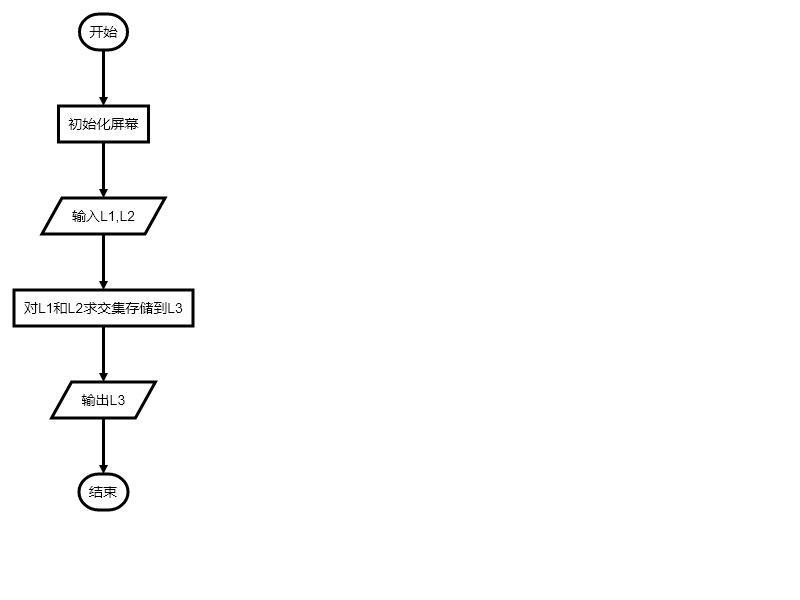
}

### 3.1.3 求交集功能截屏示例

## 3.2 总体系统的实现

### 3.2.1 总体系统流程图

3.2.2 总体系统代码实现

void opening() {

    std::cout << "The Intersection of Two Ordered Chain List Sequences"

              << std::endl

              << "Please enter two ordered lists, ending with '-1'"

              << std::endl;

}

template <class T>

void insert(List<T> &list) {

    T temp;

    while (1) {

        std::cin >> temp;

        if (temp == "-1") break;

        list.ListIns(list.ListLen(), temp);

    }

}

template <class T>

void print(std::ostream &os, List<T> &list) {

    if (list.ListLen() == 0)

        os << "NULL";

    else

        for (int i = 0; i < list.ListLen(); i++)

os << list.ListPrint(i) << ' ';

    std::cout << std::endl;

}

bool loop(){

    opening();

    List<std::string> list\_a;

    List<std::string> list\_b;

    List<std::string> list\_c;

    insert(list\_a);

    insert(list\_b);

    findIntersection(list\_a, list\_b, list\_c);

    print(std::cout, list\_c);

    std::cout << "Again? (y for yes, n for no)";

    char c;

    std::cin >> c;

    if (c == 'y')

        return true;

    else

        return false;

}

int main() {

    while (loop())

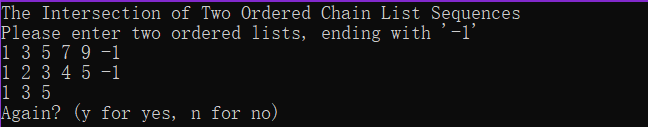
        ;

    std::cout << "Thanks for using it. See you next time! " << std::endl;

    return 0;

}

### 3.2.3 总体功能截屏示例



# 第4章 测试

## 4.1 功能测试

### 4.1.1 求交集功能测试

**测试用例**：

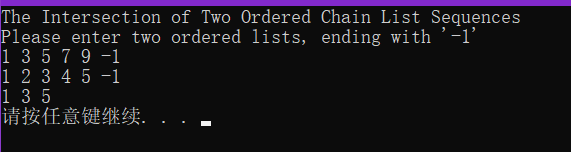
1 3 5 7 9 -1

1 2 3 4 5 -1

**预期结果**：

1 3 5

**实验结果**



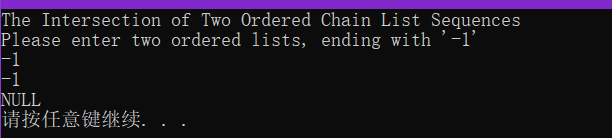
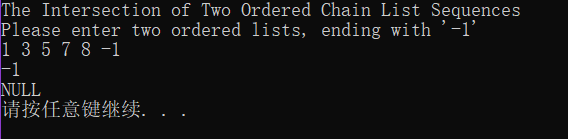
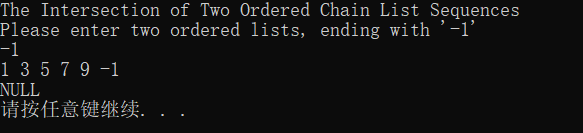
## 4.2 边界测试

### 4.2.1 输入空链表

**测试用例：**初始输入中一个或两个链表为空

**预期结果：**输出NULL。

**实验结果：**



### 4.2.2 链表无交集

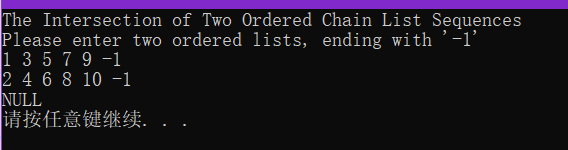
**测试用例：**

1 3 5 7 9 -1

2 4 6 8 10 -1

**预期结果：**输出NULL。

**实验结果：**



## 4.3 出错测试

### 4.3.1 输入无序序列

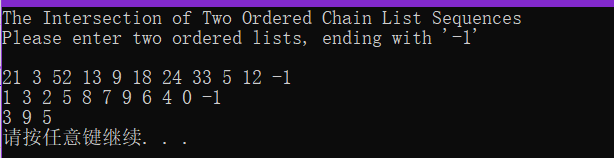
**测试用例：**

21 3 52 13 9 18 24 33 5 12 -1

1 3 2 5 8 7 9 6 4 0 -1

**预期结果：**因算法与链表是否有序无关，故正常输出3 5 9

**实验结果：**



### 4.3.2 输入负数序列

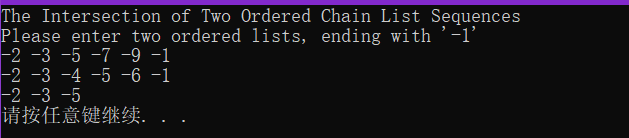
**测试用例：**

-2 -3 -5 -7 -9 -1

-2 -3 -4 -5 -6 -1

**预期结果：**因判断输入结束的标志为是否输入-1，故输入其他非-1的负数仍能正常运行。

**实验结果：**



### 4.3.3 输入非数值序列

**测试用例：**分别输入字符、字符串型数据

**预期结果：**因使用string型作为List类模板，故程序正常运行不崩溃。

**实验结果：**