两个有序链表序列的交集

作 者 姓 名： 李翠琪

学 号： 1751022

指 导 教 师： 张颖 \_\_

学院、 专业： 软件工程 \_\_

同济大学

Tongji University

目录

[1 分析 1](#_Toc25172270)

[1.1 功能要求 1](#_Toc25172271)

[1.2 功能分析 1](#_Toc25172272)

[2 设计与实现 2](#_Toc25172273)

[2.1 Node类设计与实现 2](#_Toc25172274)

[2.2 LinkedList类设计与实现 2](#_Toc25172275)

[2.3 主函数设计与实现 3](#_Toc25172276)

[3 测试 4](#_Toc25172277)

[3.1 功能测试 4](#_Toc25172278)

[3.1.1 测试1 4](#_Toc25172279)

[3.1.2 测试2 5](#_Toc25172280)

[3.1.3 测试3 5](#_Toc25172281)

[3.1.4 测试4 5](#_Toc25172282)

[3.2 边界测试 6](#_Toc25172283)

[3.2.1 其中一个链表为空 6](#_Toc25172284)

[3.2.2 两个链表为空 6](#_Toc25172285)

# 1 分析

## 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. 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 |

## 1.2 功能分析

首先需要构造两条链表分别存储所给的两列信息，然后从链表头开始逐渐对比，并且把相同的元素打印显示，直到走到其中一个链表的尽头。

# 2 设计与实现

## 2.1 Node类设计与实现

Node类为LinkedList的基础节点，是一个模板类。它能够存储一个T类型的值以及一个指向下一Node节点的next成员变量。

template <class T> class Node  
{  
public:  
  
 Node(const T & value, Node<T> \*next= nullptr) : value(value), next(next) {}  
 Node(Node<T> \*next= nullptr) : next(next) {}  
  
 T value;  
 Node<T> \*next;  
};

## 2.2 LinkedList类设计与实现

模板类，有两个Node类型的成员变量first以及last，分别指向链表头和末尾。

只有一个add（）函数，负责将数据加入链表中。添加的过程是：如果头尾指针都为空的话，就让头尾指针指向这个元素，否则就在尾指针之后添加这个元素，并且让尾指针指向它。（而且如果first->next为空，得让它指向last，不这样的话first只是一个单独的节点，和后续节点没有产生联系）。

template<class T>class LinkedList  
{  
public:  
 LinkedList():first(nullptr),last(nullptr)  
 {  
  
 }  
 void add(T t)  
 {  
 if(first== nullptr && last== nullptr)  
 {  
 first = new Node<T>(t, nullptr);  
 last = new Node<T>(t, nullptr);  
 }  
 else  
 {  
 Node<T> \*temp = new Node<T>(t, nullptr);  
 last->next = temp;  
 last=last->next;  
 if(first->next== nullptr)  
 {  
 first->next = last;  
 }  
 }  
 }  
  
 Node<T> \* first;  
 Node<T> \* last;  
};

## 2.3 主函数设计与实现

首先使用LinkedList类创建两个链表，然后让用户输入两行数字。一开始让用户输入的数字通过LinkedList的add函数加入到第一个链表里面，如果输入的数字中遇到了-1，那么通过一个bool类型的变量控制使得之后输入的数字放入第二个链表中，直到输入结束。（-1不管是哪个链表都不存）

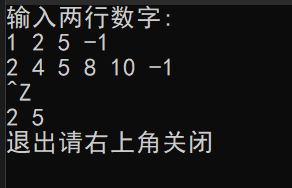
之后用cur1，cur2分别指向两个链表的开头，并且通过一个bool类型的hasPrint控制如果他们没有相同的元素（即没有进入打印的那个if语句）就打印一个NULL。因为是两个非降序的链表，所以我们只需要不断的比较两个链表的头节点（这里即是cur1，cur2），如果他们值相等，就打印出来，并且让任意一个链表头节点指向他的下一个节点即可（这里是cur1）。如果他们不等，那么就让较小的那个节点指向他的下一个节点，然后通过循环判断他们两个是否为空，并且开始下一轮的比较。因为每一次比较之前都会判断他们是否为空，且每一次比较不管是相等还是不等，都会有且仅有一个链表的指针向下移动一位，且之后又会判断他们两个链表是否为空，所以这样比较是安全的且时间复杂度最低的。直到其中一条链表为空后，退出循环，并且由用户决定是否退出。

int main()  
{  
 LinkedList<int> list1, list2;  
 cout<<"输入两行数字:";  
 int index;  
 bool goFirst = true;  
 while(cin>>index)  
 {  
 if(index==-1)  
 {  
 goFirst=false;  
 }  
 else if(goFirst)//-1都不需要  
 {  
 list1.add(index);  
 }  
 else if(!goFirst)  
 {  
 list2.add(index);  
 }  
 }  
 Node<int> \* cur1 = list1.first;  
 Node<int> \* cur2 = list2.first;  
 bool hasPrint = false;  
 while(cur1 && cur2)  
 {  
 if(cur1->value<cur2->value)  
 {  
 cur1 = cur1->next;  
 }  
 else if(cur1->value>cur2->value)  
 {  
 cur2 = cur2->next;  
 }  
 else  
 {  
 hasPrint=true;  
 cout << cur1->value << " ";  
 cur1 = cur1->next;  
 }  
 }  
 if(!hasPrint)  
 {  
 cout << "NULL";  
 }  
 cout <<endl<< "退出请右上角关闭";  
 while (true)  
 {  
 }  
 return 0;  
}

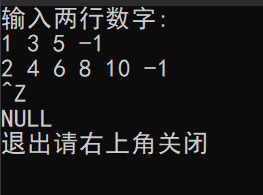
# 3 测试

## 3.1 功能测试

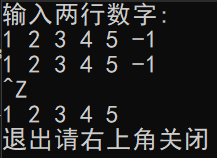
### 3.1.1 测试1



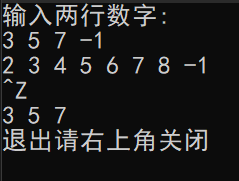
### 3.1.2 测试2



### 3.1.3 测试3

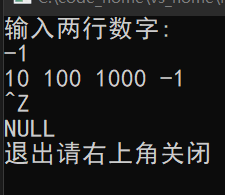


### 3.1.4 测试4



## 3.2 边界测试

### 3.2.1 其中一个链表为空



### 3.2.2 两个链表为空

