两个有序链表的交集

1651573 刘客

功能分析

- **输入说明** 输入分两行,分别输入两个非降序正整数序列,用-1作结尾(-1不属于这个序列)
- **输出说明** 输出两个序列的交集,如果不存在,则输出为NULL

编译说明

- 在windows平台下的.exe文件
- 在Linux平台下的out文件

一 . 设计

1. 数据结构设计

   > 题目要求明确,求两个有序链表序列的交集。故数据结构采用链表。

2. 类的设计

- ## Node 类
  - 成员变量

成员名称	属性	类型	描述
num	private	int	存储的数值
next	private	Node*	所链接的下一个节点

  - 成员函数

函数名称	返回值类型	描述
Node	无	构造函数
getNext	Node*	获得下一个节点
setNext	void	设置下一个节点

- **NodeList 类**
  - 成员变量

成员名称	属性	类型	描述
first	private	Node*	头结点
tail	private	Node*	尾节点

  - 成员函数

函数名称	返回值类型	描述
NodeList	无	构造函数
~NodeList	无	析构函数
Insert	void	插入一个节点
getFirst	Node*	获得头结点
getTail	Node*	获得尾结点
deleteNode	void	将全部节点删除并释放内存
setTail	void	设置尾结点
setFirst	void	设置头结点

二 . NodeList类的部分实现

- 构造函数

> 当开辟内存失败时,输出错误信息

```
  NodeList() {
          first = new Node(0);
          if (first == NULL) {
              cerr << "链表首指针动态内存分配失败,请重试" << endl;
              return ;
          }
          tail = new Node(0);
          if (tail == NULL) {
              cerr << "链表尾指针动态内存分配失败,请重试" << endl;
              return ;
          }
      }
```

- 析构函数

> 将链表里的节点全部删除,释放内存

```
  NodeList::~NodeList() {
      deleteNode();

      delete getTail();
      delete getFirst();
  }
```

- 插入函数

> 首先判断首指针是否有头结点,若没有则初始化头结点,否则则利用尾指针进行插入操作

```
void NodeList::Insert(Node * node) {
    //如果表中无元素,即头指针的子节点为NULL
    if (getFirst()->getNext() == NULL) {
        //则对头结点和尾节点进行初始化
        setFirst(node);
        setTail(node);
    }
    else
    {
        //如果表中有元素,则表明尾节点所指向非空
        Node* tempNode = getTail()->getNext();
        tempNode->setNext(node);
        getTail()->setNext(node);
    }

}
```

- 删除函数

> 利用循环释放内存

```
NodeList::~NodeList() {
    deleteNode();
    delete getTail();
    delete getFirst();
}
```

三 . 主程序实现

- 函数

函数名称	返回值类型	描述
cin_num	void	输入函数
outIntersection	void	进行交集操作并输出
main	int	主函数

- cin_num 函数实现

> 利用cout对用户给出提示信息 在循环中读取正整数 当用户输错数字时,给出提示,并令其重新输入 通过将原来的指针删除,新建链表来实现数据的重置

```
```
void cin_num(NodeList* list) {
    cout << "请输入您的正整数!非降序!序列,输入-1则代表输入结束" << endl;
    int num;
    bool minus = false;
    do {
        cin >> num;
        if (num == -1) {
            if(minus){
                cout << "请输入正整数!!!请重新输入\n";
                list->deleteNode();
                minus = false;
                continue;
            }
            break;
        }
        if (num < 0) {
            minus = true;
        }
        Node* tempNode = new Node(num);
        if (tempNode == NULL) {
            cerr << "当前输入节点分配内存失败,请重试";
            return;
        }
        list->Insert(tempNode);
    } while (1);
}
```
```

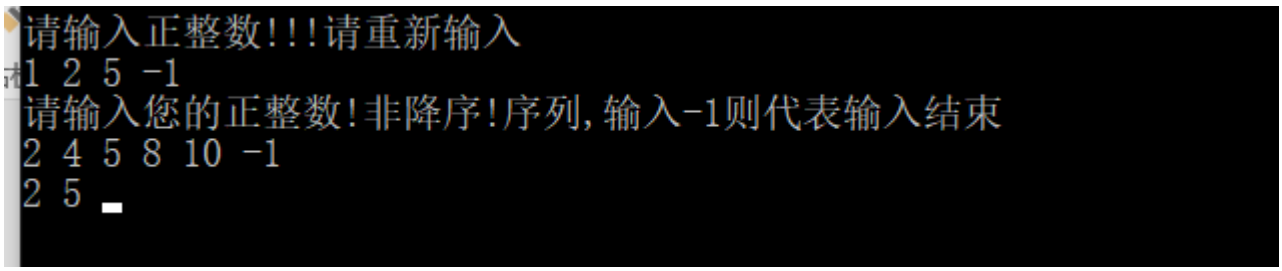
- outIntersection 函数实现

> 从两个链表的头结点开始比较,如果数字相同,则将该数字输出(表示在并集中),如果数字不同,则数字小的那个取其下一个节点(有序链表),直到一方为空,循环结束

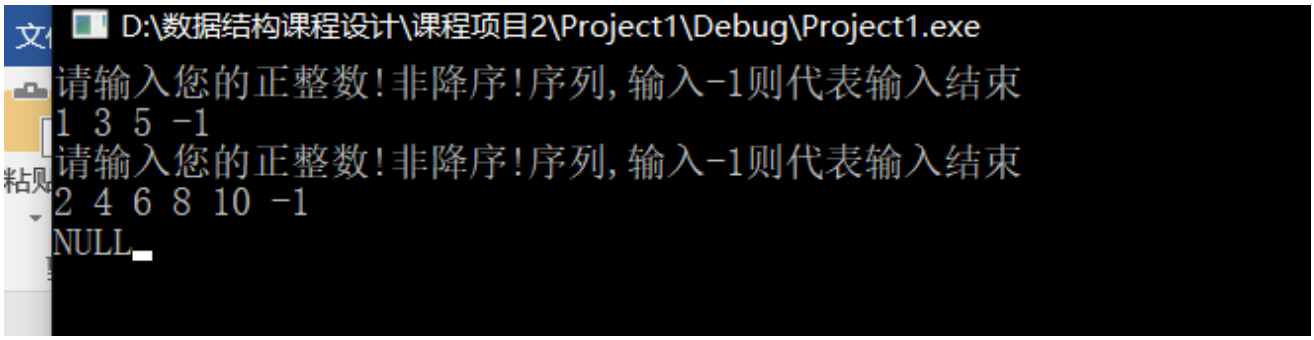
```
void outIntersection(NodeList* s1, NodeList* s2) {
    //取S1表中首元素(非头结点)
    Node* First1 = s1->getFirst()->getNext();
    //取S2表中首元素(非头结点)
    Node* First2 = s2->getFirst()->getNext();
    //用来计数,看是否在遍历过程中,交集为空
    int count = 0;


    //如果首元素不存在
    if (First1 == NULL || First2 == NULL) {
        cout << "NULL";
        return;
    }

    do {
        //如果两个元素相等,则输出,并两个表指针同时移动
        if (First1->getNum() == First2->getNum()) {
            cout << First1->getNum() << " ";
            First1 = First1->getNext();
            First2 = First2->getNext();
            count++;
        }
        //如果S1元素>S2元素,则S2指针移动
        else if (First1->getNum() > First2->getNum()) {
            First2 = First2->getNext();
        }
        //反之,S1指针移动
        else {
            First1 = First1->getNext();
        }
        //当存在一张表被便利完毕时,结束程序
    } while (First1 != NULL && First2 != NULL);

    //如果遍历完之后并集为空,则输出NULL
    if (count == 0) {
        cout << "NULL";
    }
}
```

四. 程序运行效果

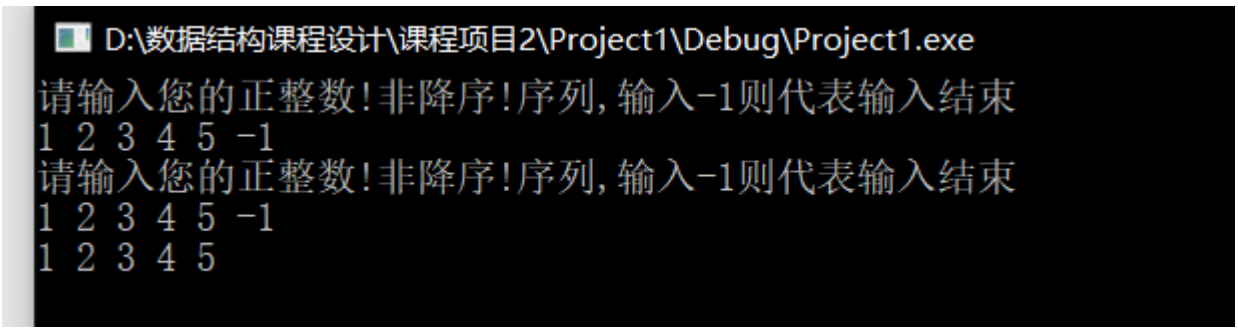
- 测试样例1



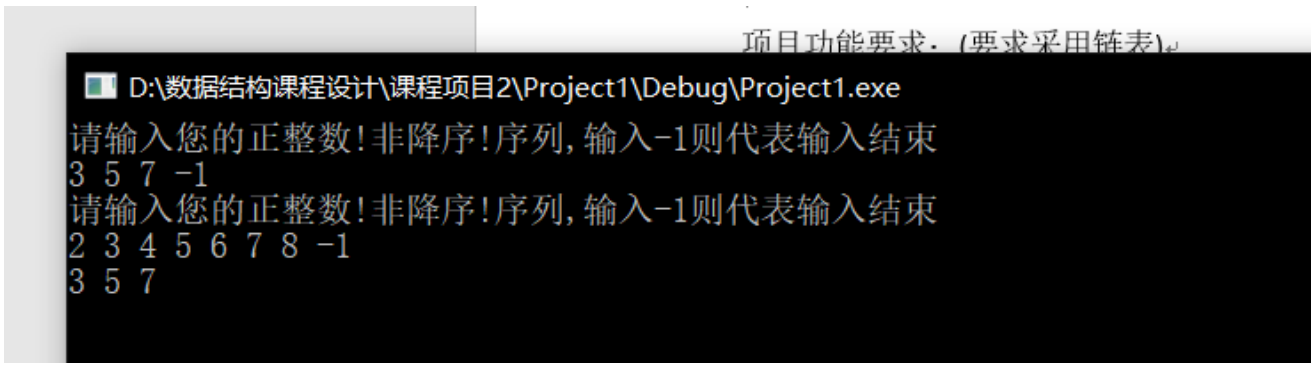
- 测试样例2



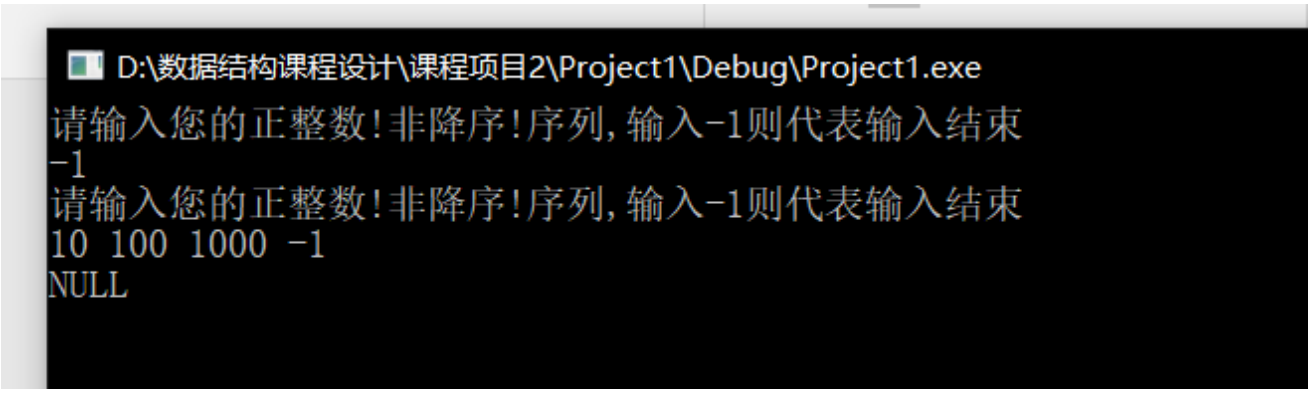
- 测试样例3



- 测试样例4



- 测试样例5



五. 错误检测

- 当输入负数时