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**

作 者 姓 名： 吴英豪

学 号： 1953608

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

1.分析

1.1功能分析

通过链表实现以下的功能。已知两个非降序链表序列S1和S2，设计函数构造出S1和S2的交集链表S3。

输入说明：输入分两行，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，结果不能有多余空格；若链表为空，输出NULL。

2.设计

2.1 数据结构与算法设计

由于此项目已经要求使用链表，故采用链表的数据结构实现本项目。鉴于此系统需要逐个输入链表的数据，所以使用尾指针\_list,以及函数append()将输入的数据构造成结点插入到表尾。

构造好两个链表（链表1与链表2）之后，需要引入两个辅助指针从头开始遍历链表，指针对应数据相等则同时向后移动；若链表一的数据大于链表二的数据，则将链表二的指针后移；若链表一的数据小于链表二的数据，将链表一此结点删除，指针后移。直到辅助指针移到某一个链表结束为止。

说明：用于设计删除操作，采用双向链表的形式以简化删除操作。最终结果存储在链表一中，所以当链表二结束但是链表一没有结束的时候，我们将链表一里剩余的部分全部删去（void deleteNode(linkNode<T>\* cur\_node);）

最后只需要使用

printResult()函数输出结果即可

2.2类结构设计

一共有两个类:结点类（linkNode）和链表类(linkList).结点类主要用于存放数据，链表类主要用于实现一定的操作（做集合的交集，结点删除等）。链表有头指针\_first和尾指针\_last,且是个双向链表，便于数据的查询和删除。在构造类时选择了模板类，提高了复用性。在最终使用的时候按照题意选择了正整数unsigned int

有关于类的定义以及 放在了头文件link.h中。与界面展示有关的函数以及main()函数被放在了02\_1953608\_WuYinghao.cpp文件中。

注：因为链表需要访问结点的成员（尤其是私有成员），所以将链表Linklist设置成了结点linkNode的友元

2.3成员与操作设计

**结点类（linkNode）**

私有成员

T \_data; //链表结点上的值

linkNode<T>\* \_rLink; //指向下一个结点

linkNode<T>\* \_lLink; //指向前一个结点

公有操作

friend class linkList<T>; //结点的友元类

linkNode(); //默认构造函数，默认是int类型的0

linkNode(T number); //给定参数的构造

~linkNode(); //析构函数

**链表类(linkList)**

私有成员：

linkNode<T>\* \_first; //头指针，用于查询

linkNode<T>\* \_last; //尾指针，用于队尾插入

公有操作

linkList(); //构造函数

~linkList(); //析构函数

void append(T data); //在链表尾部插入元素

void intersect(linkList<T>& list); //做交集的运算

void deleteNode(linkNode<T>\* cur\_node); //删除从此节点后的所有结点

void printResult(); //最后的打印操作

bool isEmpty() const; //判断链表是否为空

T getFinalData() const; //返回链表末尾的数值

2.4 文件设计

本系统支持windows系统和Linux系统。

(1)Windows系统

windows系统的相关文件在文件夹02\_1953608\_WuYinghao\_Windows中，

其中类的定义以及其成员函数的定义在link.h中;main函数以及相关的函数定义在02\_1953608\_WuYinghao.cpp中.02\_1953608\_WuYinghao.exe为Windows平台下的可执行文件。

**02\_1953608\_WuYinghao.cpp和link.h的编码格式均为**

**简体中文(GB2312)-代码页 936**

(2)Linux 系统

Linux系统的相关文件在文件夹02\_1953608\_WuYinghao\_Linux中,

其中类的定义以及其成员函数的定义在02\_1953608\_WuYinghao\_Linux.h中;main函数以及相关的函数定义在02\_1953608\_WuYinghao\_Linux.cpp中.02\_1953608\_WuYinghao.out为Linux平台下的可执行文件。

**02\_1953608\_WuYinghao\_Linux.cpp和02\_1953608\_WuYinghao\_Linux.h的编码格式均为Unicode(UTF-8 无签名)-代码页65001**

3.功能实现

3.1尾端插入功能的实现

**3.1.1功能说明：**

由于用户在输入数据的时候是逐个输入，所以采用尾端插入的方式将数据对应的结点通过\_last插入在链表的末尾,函数的参数就是输入进入的数据data;

（在执行尾端插入过程时，尾指针需要有所变动）

**3.1.2 尾端插入功能核心代码**

template<class T>

void linkList<T>::append(T data)

{

linkNode<T>\* newNode = new linkNode<T>(data);

if (NULL == newNode)

{

cerr << "无法创建此结点";

system("pause");

exit(1);

}

else

{

newNode->\_rLink = \_last->\_rLink->\_rLink;

\_last->\_rLink->\_rLink = newNode;

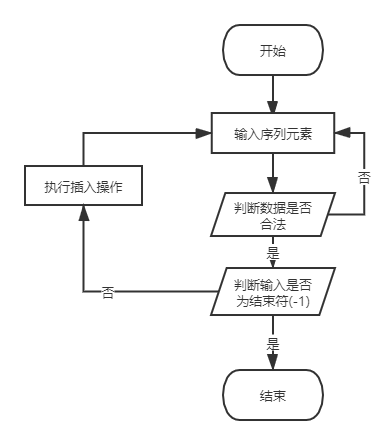
newNode->\_lLink = \_last->\_rLink;

\_last->\_rLink = newNode;

}

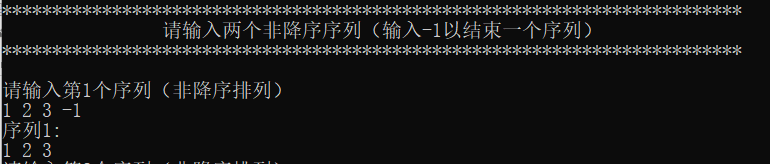
}

**3.1.3插入功能流程图**



**3.1.4插入功能截屏展示**

注：这里的插入功能会在用户输入数据之后自动调用，不需要用于给出执行指令。执行这一个功能之后，链表创建成功。



3.2交集运算的实现

3.2.1功能说明

引入两个辅助指针分别指向两个链表的第一个元素（在链表不为空的时候）。通过辅助指针比较数据。

若两个数据相同（数据应该属于交集中），则两个指针同时后移。**笔者把最后的结果存在了第一个链表中，所以不需要额外新建一个链表去存放结果。**

当链表二的数据小于链表一的数据时，链表二对应的辅助指针后移。当链表一的数据大于链表二的数据时，将此结点删除（因为链表一也被用于存放结果）并将链表一的辅助指针后移。

当链表二或链表一结束的时候停止操作。若链表二结束但链表一尚未结束，则需要从对应位置开始将链表一中剩余的元素删去。最终得到的链表一就是交集运算后的结果。

**注意：**

**（1）使用这个方法需要建立在一个基础之上：链表是非降序排列的。尽管此项目在用户输入时做了是否为非降序排列的操作，但还是希望用户在使用的时候能够按照要求正确使用。（输入-1结束输入）**

（2）根据题解算法，做完交集运算之后一定有一个链表是被遍历完的。若第一个链表（及存放结果的链表）被遍历完，则不会影响最终的结果；若是第二个链表被遍历完，这是后需要将第一个链表多出来的部分删除。（通过while循环调用删除操作实现这一功能）

3.2.2交集运算核心代码

template<class T>

void linkList<T>::intersect(linkList<T>& list)

{

linkNode<T>\* p\_1 = this->\_first->\_rLink->; //最后的结果存在p\_1所对应的链表中

linkNode<T>\* p\_2 = list.\_first->\_rLink->\_rLink;

while ((p\_1 != NULL) && (p\_2 != NULL))

{

if (p\_1->\_data == p\_2->\_data)

{

p\_1 = p\_1->\_rLink;

p\_2 = p\_2->\_rLink;

}

else if (p\_1->\_data>p\_2->\_data)

{

p\_2 = p\_2->\_rLink;

}

else if (p\_1->\_data < p\_2->\_data)

{

linkNode<T>\* current = p\_1;

p\_1 = p\_1->\_rLink;

deleteNode(current);

}

}

/\*

\* 比较结束之后一定会有一个链表先查询结束，

\* 若是链表一先结束，不影响实际的结果

\* 若是链表二先结束，需要把链表一多出来的部分删除

\*/

if (NULL == p\_2)//链表二先结束

{

while (p\_1 != NULL)

{

linkNode<T>\* current = p\_1;

p\_1 = p\_1->\_rLink;

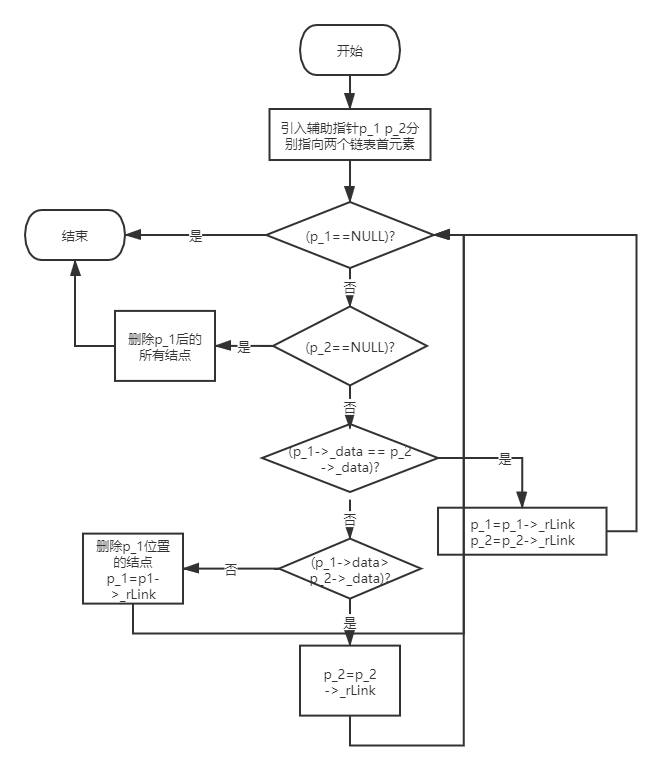
deleteNode(current);

}

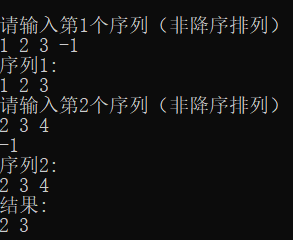
}

}

3.2.3交集运算功能流程图



3.2.4交集运算功能截屏示例



**3.3 删除节点**

**3.3.1 功能说明**

删除链表中参数p指针对应的结点。因为这个链表是双向链表，所以便于将应删除的接管P之前与之后的指针连接起来。

注意这个方法只能删除一个结点，若想删除所有的结点则需要循环调用此函数

**3.3.2 删除功能核心代码**

template<class T>

void linkList<T>::deleteNode(linkNode<T>\* cur\_node)

{

//由于已经有了附加头结点，所以不考虑第一个结点的删除，只额外考虑最后一个结点的删除

if (cur\_node->\_rLink == NULL) //删除的是最后一个结点

{

cur\_node->\_lLink->\_rLink = NULL;

delete cur\_node;

}

else //删除的是中间的结点

{

cur\_node->\_lLink->\_rLink = cur\_node->\_rLink;

cur\_node->\_rLink->\_lLink = cur\_node->\_lLink;

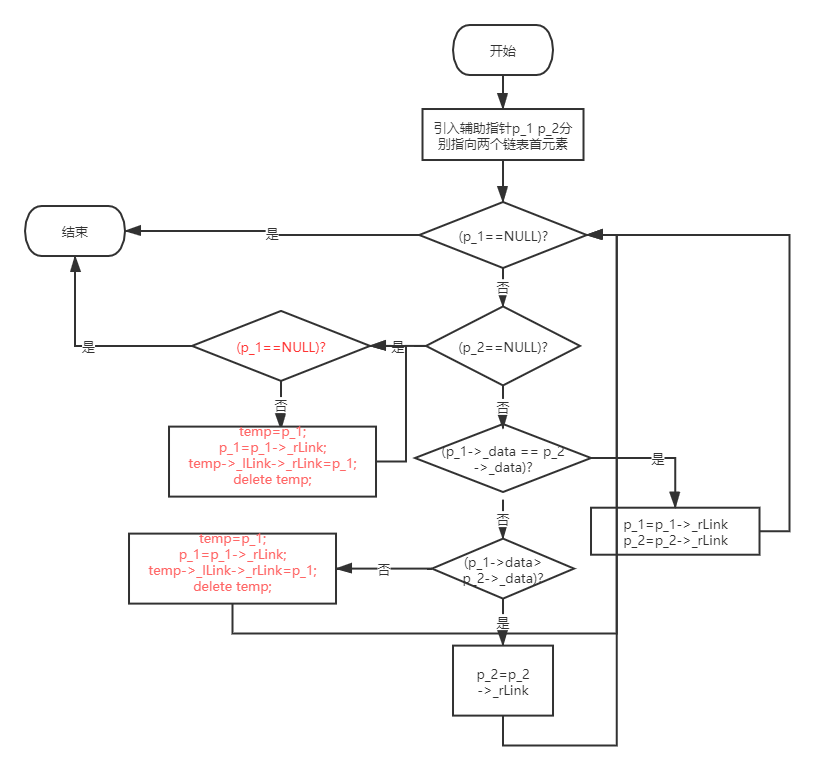
delete cur\_node;

}

}

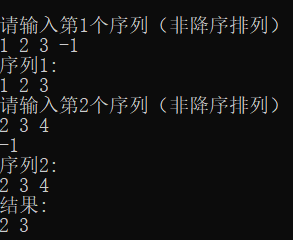
**3.3.3 删除部分流程图**

**注：图中红色部分为删除操作**



**3.3.4 删除部分程序截图**

注意：删除操作是交集运算中被调用的，用户不能主动调用。若最后输出成功则表示删除成功。



**3.4 判断链表是否为空以及返回最后一个元素**

**3.4.1 功能说明**

输入的链表有可能为空链表，同时交集运算的结果也是空集，这时候链表的值应该为NULL。从附加头开始，如果附加头的右指针是空指针，说明此链表为空，返回true,否则返回false.

输入数据的时候应该输入非降序的数，但为了避免用户错误输入，需要做一定的检测。每次向链表尾端插入数据的时候，若此数据小于链表尾端的数据，则需要用户重新输入。

**3.4.2. 核心代码展示**

bool linkList<T>::isEmpty() const

{

linkNode<T>\* cur = this->\_first->\_rLink->\_rLink; //指向附加头的下一个结点

if (NULL == cur)

return true;

else

return false;

}

template<class T>

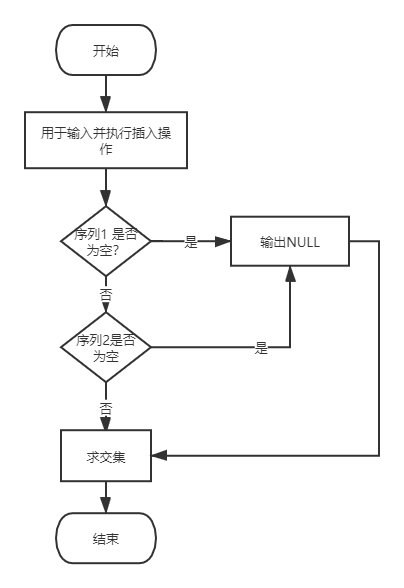
T linkList<T>::getFinalData()const

{

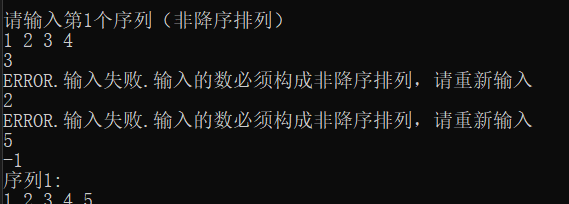
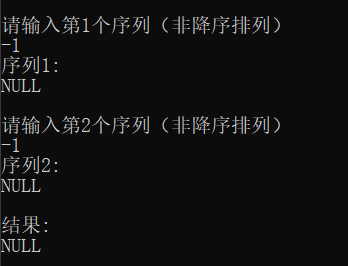
return this->\_last->\_rLink->\_data;

}

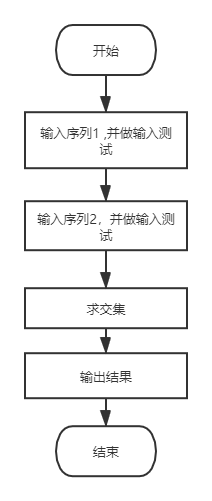
**3.4.3.流程图**

****

**3.4.4.程序截图**



**3.5 程序总流程图**

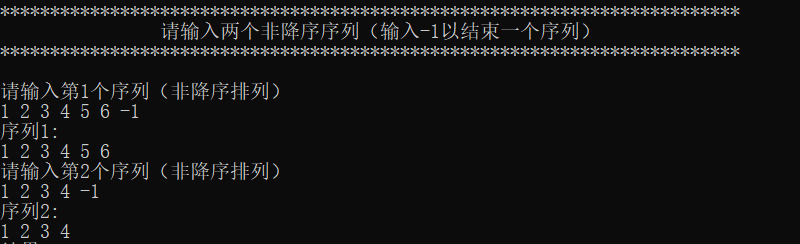
****

**4.测试**

**4.1 功能测试**

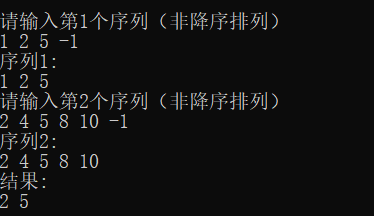
**4.1.1输入功能测试**

当输入两个链表集合完毕时，会将最终的集合显示出来。

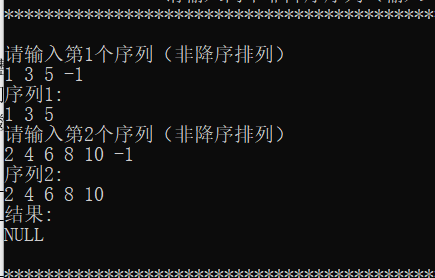


**4.1.2交集功能测试**

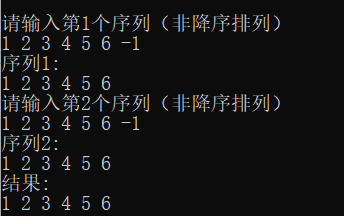
1.一般情况



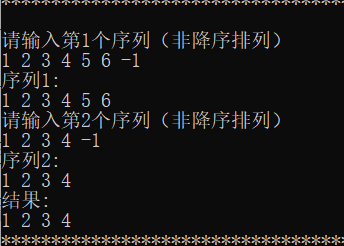
2.交集为空的情况



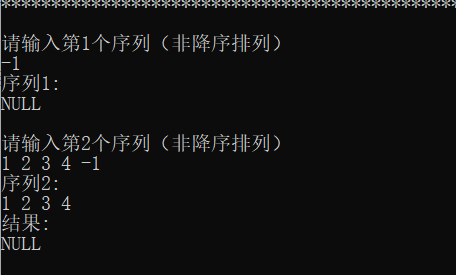
3.完全相交的情况



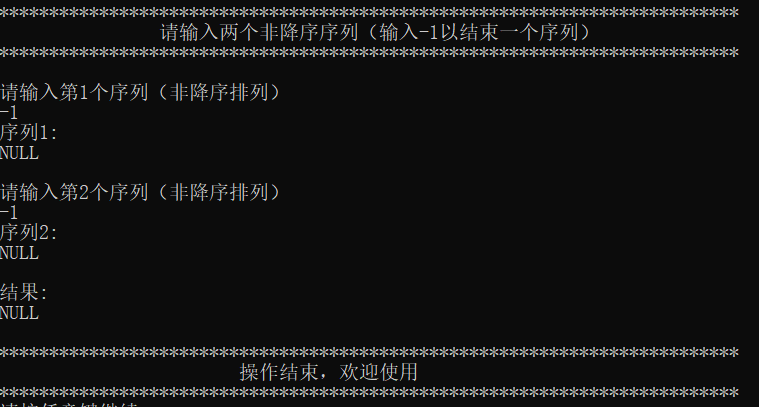
4.一个序列完全包含于另一个交集



5.其中一个序列为空的情况

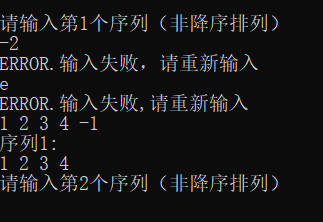


**4.2 边界测试**

当两个序列其中一个或者两个都是空序列时，程序不崩溃，正常运行。

**4.3 出错提示**

4.3.1 输入不合法的内容（除-1的负数或字符），有相应提示，程序不崩溃



4.3.2 当输入数字不构成非降序时，会给出相应提示，程序不崩溃

