

项目说明文档

数据结构课程设计

——两个有序链表的交集

作者姓名：_____沈星宇_____

学 号：_____1951576_____

指导教师：_____张颖_____

学院、专业：_____软件学院 软件工程_____

同济大学

Tongji University

一、 分析

(1) 应用背景

对两个集合取交集是数学里面常用的基本操作之一，在交集的基础之上可以延伸出更多数学内容。本项目通过链表构造实现取交集的目标。

本项目对两个非降序链表序列 list1 和 list2 进行取交集的操作，设计函数构造出二者公共的部分 resultList。

而对于用户输入的非升序链表会进行提示报错，要求重新输入，并且要能够对用户的不合法输入（例如字母等）进行报错处理，直到用户输入符合要求的序列为止。

(2) 项目功能要求

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。

二、 设计

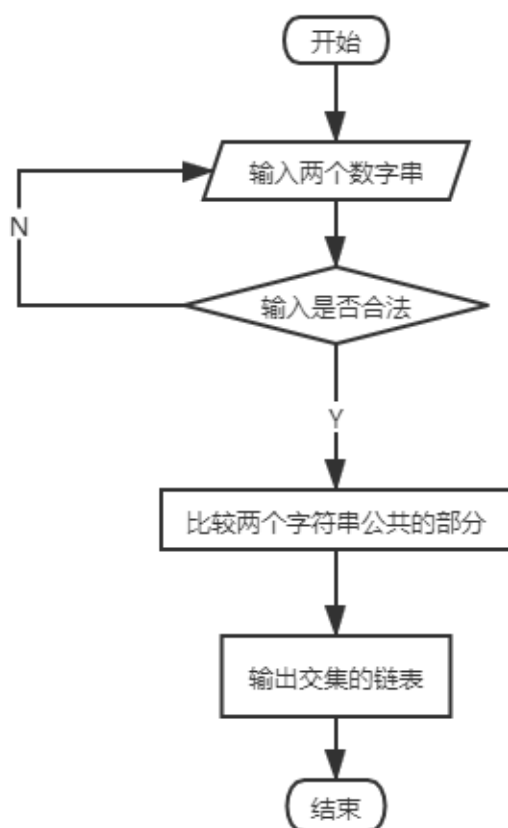
(1) 数据结构设计

```
class ListPoint {
private:
    int data;
    ListPoint* left;
    ListPoint* right;
    friend class List;
public:
    ListPoint(const int& item);
    bool insert(ListPoint*& curPosition, ListPoint*& newNode);
    int getNum();
};
```

```
class List {
private:
    ListPoint* head;
    ListPoint* tail;
    int length;
public:
    List();
    bool makeEmpty();
    bool reset();
    int getLength();
    ListPoint* find(int location);
    bool insert(int value, int location);
    bool remove(int location);
    bool check();
    void compare(List* list2, List*& resultList);
    void print();
};
```

- 1、ListPoint 是结点的结构，内部包含了存储的数字 data 和指针 left、right，为了锻炼写双链表的能力才写成了双链表，其实本项目单链表即可实现
- 2、List 是链表的结构，里面包含了常规的操作的函数，为了使 List 能够访问 ListPoint 的成员，故将其在 ListPoint 类里写作了友元。

(2) 程序流程设计



- 1、首先对链表进行初始化，初始输入一些数据，此处对合法性进行判断，如果输入不合法则会请求重新输入；输入合法数据后会告诉你输入正确
- 2、调用比较函数 `compare()`，同时生成新的链表，将新的链表打印输出
- 3、最后释放动态分配的内存

三、 实现

1、main()函数的内部逻辑

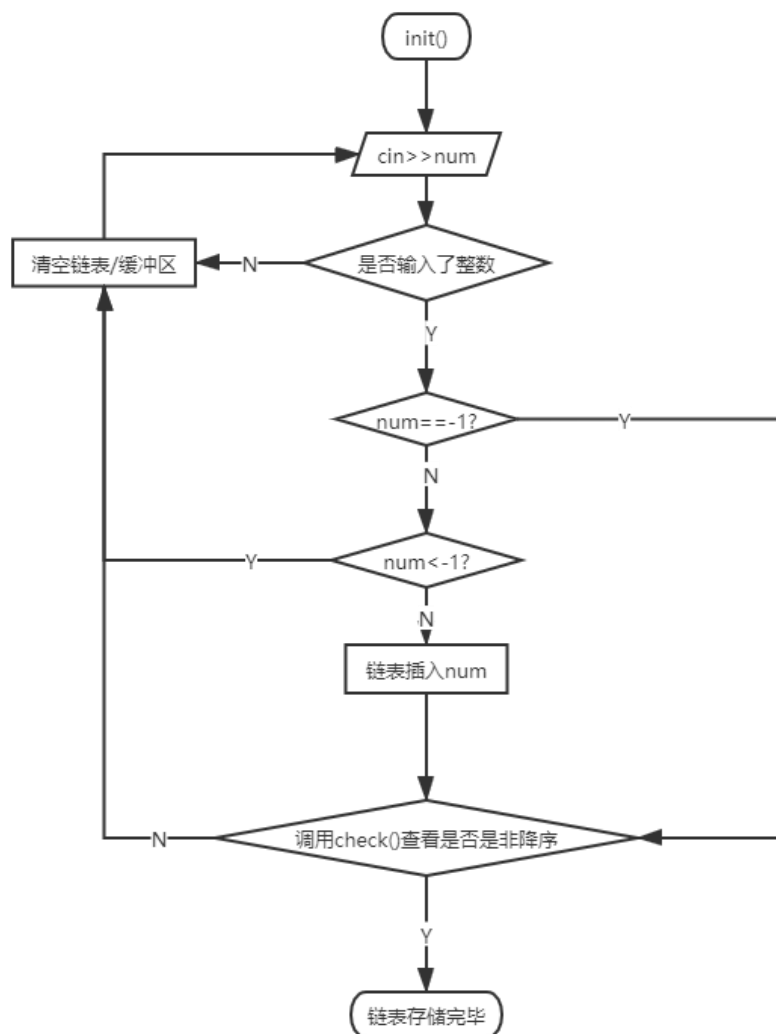
- (1) `main()`函数会首先用 `init()`函数对链表进行初始化，在 `init()`函数内部对输入合法性进行检查。
- (2) 接着调用 `list1` 的 `compare` 函数，传入的引用 `resultPointer` 是指向 `resultList` 的指针，该操作使得调用完比较函数后能使 `resultList` 将交集的部分存储起来。
- (3) 调用 `resultList` 的 `print()`函数将结果打印出来
- (4) 释放链表的内存

```

int main() {
    std::cout << "请输入两串若干个正整数构成的非降序序列，以-1结束输入\n";
    int num;
    List list1;
    List list2;
    List resultList;
    init(num, list1);
    init(num, list2);
    List* resultPointer = &resultList;
    //-----
    list1.compare(&list2, resultPointer);
    resultList.print();
    //-----
    list1.makeEmpty();
    list2.makeEmpty();
    resultList.makeEmpty();
    return 0;
}

```

2、init()的实现



- (1) 对于输入的一串数字，用 `cin>>num` 对单独的数字进行处理，首先用 `cin.fail()` 检测是否输入了不合法的类型
- (2) 然后判断输入的数字是否是-1，读取到-1 则停止插入；如果输入的整数是除-1 以外的其他负数，那么请求从头重新输入链表；输入合法时则将其用尾插法插入链表
- (3) 用 `check()`函数查看是否输入的序列为非降序，如果是不和要求，要请求重新输入新的数据

```
void init(int& num, List& list) {
    while (1) {
        while (1) {
            std::cin >> num;
            if (std::cin.fail()) {
                std::cout << "输入不合法,请输入正整数,并以-1结尾\n";
                list.reset();
                std::cin.clear();
                std::cin.ignore(sizeof(num), '\n');
                continue;
            }
            else {
                if (num == -1) {
                    break;
                }
                else if (num < -1) {
                    std::cout << "输入不合法,请输入正整数,并以-1结尾\n";
                    list.reset();
                    std::cin.clear();
                    std::cin.ignore(sizeof(num), '\n');
                }
                else {
                    list.insert(num, list.getLength());
                }
            }
        }
        if (list.check()) { //如果读入的是降序则继续输入
            std::cout << "输入正确! \n";
            break;
        }
        else {
            list.reset();
        }
    }
}
```

3、List::check()的实现

用于检测输入的序列是否是非降序的序列。

首先考虑链表是空表的情况，即头结点的 `right` 指向的是 `NULL`，空表满足非降序的要求所以返回 `true`。

然后用两个 `ListPoint*`类型的指针 `p` 和 `pNext` 来读取 `p` 指向的结点和 `p` 指向的结点的下一个结点的数据，进行比较，如果非降序则直接返回 `false`。

循环终止的条件是 `pNext` 指针指向了 `NULL`，即比较到了链表的结尾。

```

bool List::check() {
    ListPoint* p = head->right; //指向头结点之后的下一个结点
    if (p == NULL) {
        return true; //是空表的情况
    }
    ListPoint* pNext = p->right;
    if (pNext == NULL) { //只有一个结点的情况
        return true;
    }
    //-----
    while (pNext != NULL) {
        if (!(p->getNum() <= pNext->getNum())) {
            std::cout << "输入不合法,请输入非降序序列\n";
            return false;
        }
        p = pNext;
        pNext = p->right;
    }
    return true;
}

```

4、List::insert()的实现

insert()函数其实是按位后插，即在 location 的后面插入一个结点。

(1) 首先会对 location 的合法性进行检测（虽然本项目按照顺序插入在队尾不会涉及到这个内容，但是为了安全性和整体性仍然进行检测）。禁止在第 0 位及第 0 位之前插入结点；在超出链表的最后一个结点的时候在最后一个结点插入。

(2) 然后调用 find()函数去查找到 location 位置的结点，用 malloc 为新的结点动态分配一个内存，修改新节点 newNode 的 data 并且调用 newNode 的 insert() 函数将它的指针链接上去。

(3) 调整尾指针，使链表的长度增加。

```

bool List::insert(int value, int location) {
    if (location < 0) {
        return false;
    }
    else if (location > length) {
        location = length;
    }

    ListPoint* p = head;
    p = find(location); //后插操作要找到location的前一个结点

    ListPoint* newNode = (ListPoint*)malloc(sizeof(ListPoint));
    newNode->data = value;
    if (!newNode->insert(p, newNode)) {
        return false;
    }
    if (newNode->right == NULL) {
        tail = newNode;
    }
    length++;
    return true;
}

```

5、List::find()的实现

按位查找，用一个指针从第一个结点指向最后一个结点，该函数返回所寻找位置的结点指针。

```
ListPoint* List::find(int location) {
    ListPoint* p = head;
    int count = 0;
    while (1) {
        if (p == NULL || count == location) {
            break;
        }
        p = p->right;
        count++;
    }
    return p;
}
```

6、ListPoint::insert()的实现

后插操作要不存在 curPositon==NULL 的情况，即不在空指针后面插入内容。把结点用指针链接起来。因为最后一个结点的 right 指针指向的是 NULL，如果是在链表的末尾插入结点，那么不需要将 newNode 之后的结点(其实是空指针)的 left 指针指向 newNode，因为空指针不存在 left 指针。

```
bool ListPoint::insert(ListPoint*& curPosition, ListPoint*& newNode) {
    if (curPosition == NULL) { //后插操作，所以不存在curPosition==NULL的插入
        return false;
    }
    newNode->left = curPosition;
    newNode->right = curPosition->right;
    if (curPosition->right != NULL) { //在末尾插入就不需要后面的NULL指针指向newNode了
        newNode->right->left = newNode;
    }
    curPosition->right = newNode;
    return true;
}
```

7、List::reset()和 List::makeEmpty()的实现

(1) 在程序接收到不合法的输入时，程序会将链表清空，并且请求用户重新输入数字序列，直到输入的序列合法为止。实现这一部分功能的是 reset() 和 makeEmpty() 函数。

(2) makeEmpty() 是释放内存的作用，在 main() 函数的末尾、程序结束时也会被调用，防止碎片化内存块的出现。它通过指针 p 去访问链表的每一个结点，将每一个结点的内存释放，然后重置链表的长度 length = 0。

(3) 对于 reset() 函数，用于清空重置链表，内部会先调用 makeEmpty() 释放内存，然后将头指针的 right 指针指向 NULL (防止访问脏数据)，将尾指针指向头结点。

```
bool List::makeEmpty() {
    ListPoint* p = head->right;
    ListPoint* freePoint = p;
    while (1) {
        if (p == NULL) {
            length = 0;
            return true;
        }
        freePoint = p;
        p = p->right;
        free(freePoint);
    }
}

bool List::reset() {
    if (!makeEmpty()) {
        return false;
    }
    head->right = NULL;
    tail = head;
    return true;
}
```

8、List::compare()的实现

因为序列是非降序的，那么用两个指针指向两个链表的结点，比较结点的 data 值，将共有的 data 存入新的链表 resultList 即可。

循环停止的条件是任意一个指针到达了链表的结尾。

```
void List::compare(List* list2, List*& resultList) {
    ListPoint* p1 = head->right;
    ListPoint* p2 = list2->head->right;
    int p1Num, p2Num;
    while (1) {
        if (p1 == NULL || p2 == NULL) {
            break;
        }
        p1Num = p1->getNum();
        p2Num = p2->getNum();
        if (p1Num == p2Num) {
            resultList->insert(p1Num, resultList->getLength());
            p1 = p1->right;
            p2 = p2->right;
        }
        if (p1Num < p2Num) {
            p1 = p1->right;
        }
        else if (p1Num > p2Num) {
            p2 = p2->right;
        }
    }
}
```

9、List::print()的实现

同样用指针 p 去访问结果链表的每个结点，输出内部的值。


注意在最后一个元素的后面不能多空格。

```
void List::print() {
    ListPoint* p = head->right;
    if (p == NULL) {
        std::cout << "NULL";
    }
    while (1) {
        if (p == NULL) {
            break;
        }
        std::cout << p->getNum();
        p = p->right;
        if (p != NULL) {
            std::cout << ' ';
        }
    }
    std::cout << "\n";
}
```

四、 测试

1、初始化表格的合法性检测

- (1) 输入非-1 的负数、浮点数和字符要求重新输入
- (2) 输入非降序序列要求重新输入
- (3) 输入人数为非降序正整数序列则进行下一步

 Microsoft Visual Studio 调试控制台

```
请输入两串若干个正整数构成的非降序序列，以-1结束输入
a -1
输入不合法, 请输入正整数, 并以-1结尾
1 -2
输入不合法, 请输入正整数, 并以-1结尾
3 2 1 -1
输入不合法, 请输入非降序序列
1 1 -1
输入不合法, 请输入正整数, 并以-1结尾
1 2 5 -1
输入正确!
```

2、一般情况


- (1) 两个序列为一般的非空非降序序列
- (2) 两个序列有共有元素
- (3) 输出两个序列的交集

```
1 2 5 -1
输入正确!
2 4 5 8 10 -1
输入正确!
2 5

D:\VS文件\数据结构课程设计\Project2\Debug\Project2.exe
按任意键关闭此窗口. . .
```

3、交集为空的情况

- (1) 两个序列为一般的非空非降序序列
- (2) 两个序列有共有元素
- (3) 输出 NULL


 Microsoft Visual Studio 调试控制台

```
请输入两串若干个正整数构成的非降序序列，以-1结束输入
1 3 5 -1
输入正确!
2 4 6 8 10 -1
输入正确!
NULL

D:\VS文件\数据结构课程设计\Project2\Debug\Project2.exe (j
按任意键关闭此窗口. . .
```

4、完全相交的情况

- (1) 两个序列为一般的非空非降序序列
- (2) 两个序列完全相交
- (3) 输出交集


 Microsoft Visual Studio 调试控制台

```
请输入两串若干个正整数构成的非降序序列，以-1结束输入
1 2 3 4 5 -1
输入正确！
1 2 3 4 5 -1
输入正确！
1 2 3 4 5

D:\VS文件\数据结构课程设计\Project2\Debug\Project2.exe
按任意键关闭此窗口. . .
```

5、其中一个序列完全属于交集的情况

- (1) 两个序列为一般的非空非降序序列
- (2) 其中一个序列完全属于交集
- (3) 输出交集


 Microsoft Visual Studio 调试控制台

```
请输入两串若干个正整数构成的非降序序列，以-1结束输入
3 5 7 -1
输入正确！
2 3 4 5 6 7 8 -1
输入正确！
3 5 7

D:\VS文件\数据结构课程设计\Project2\Debug\Project2.exe (
按任意键关闭此窗口. . .
```

6、其中一个序列为空的情况

- (1) 其中一个序列为空
- (2) 输出 NULL

 Microsoft Visual Studio 调试控制台

```
请输入两串若干个正整数构成的非降序序列，以-1结束输入
-1
输入正确！
10 100 1000 -1
输入正确！
NULL

D:\VS文件\数据结构课程设计\Project2\Debug\Project2.exe
按任意键关闭此窗口. . .
```