《数据结构》课程设计报告

题目： 学生成绩管理系统

学号： 2315080119

姓名： 刘宗铭

日期： 2024年6月20日

集合运算系统

# 一、题目说明与分析

## 1. 题目说明

以单链表的形式表示一个集合，试编写算法实现：

（1）求出两个集合A和B的交集，即 A∩B；

（2）求出两个集合A和B的差集，即 A- B；

（3）求出两个集合A和B的并集，即 A∪B。

要求：A和B集合是由任意输入的数据构建的单链表，不是程序定死的。

## 2. 题目分析

根据上述题目描述可知：

（1）这是一个实现集合运算的程序，需要使用单链表数据结构来存储集合元素。因为输入的集合元素可以是任意类型数据，例如数字、中文、英文或多种混合的类型。所以存储集合元素变量类型应定义为字符串类型。

(2) 程序需要实现集合的交集、差集、并集等基本运算。程序中应包含数据录入、数据显示、集合运算等功能，可以划分为独立的模块，并使用自定义函数来实现。

(3) 为了方便用户操作，需要设计一个用户交互界面，允许用户输入集合元素，并选择需要进行的集合运算。

(4) 需要在部分输入场景考虑对输入数据进行验证，以防止用户输入错误的数据导致程序运行错误。

(5) 因为要求输入数据不能定死，所以本题不考虑使用文本文件来保存集合元素，使用即用即存即算的设计思路。

(6) 可以考虑在用户界面中增加一些辅助功能，例如显示集合元素数量、批量增加集合元素、修改和删除集合元素、清空集合等。

(7) 可以进一步扩展功能，例如实现其他集合运算，或者实现更复杂的数据结构，以提高运算效率。

# 二、系统设计

## 1. 理论基础

本程序的数据是使用单链表来存放及管理，因为数据内容需要实时录入，且插入和删除操作较多，故适宜使用单链表来实现。根据题目分析，应先定义出集合成员结构体，每个节点包含数据域和指向下一个节点的指针域，再将集合成员结构体通过指针相连形成单链表。而数据查询可用单链表的指针操作来实现为了提高代码的健壮性和代码复用能力，使用类来封装链表的系列方法，同时也方便之后定位与拓展操作。

## 2. 数据类型定义

### （1）集合成员节点结构体

使用结构体 Node 来表示单链表中的节点，包含数据域 data 和指向下一个节点的指针域 next。

struct Node {

string data;

Node\* next=NULL;

};

### （2）集合类成员

使用类 LinkedList 来表示单链表，包含头节点指针 head 和链表长度 Length。

class LinkedList {

private:

Node\* head = NULL;

int Length = 0;

public:

*// ...*

};

## 函数定义及说明

### 类内方法函数：

* void insert(const string& value)：在链表头部插入一个新节点，并增加链表长度。
* int deletedata(Node\* finded)：删除链表中指定节点。
* void display() const：显示链表中所有元素。
* Node\* getHead() const：返回链表的头节点指针。
* int length() const：返回链表长度。
* Node\* finddata(string value) const：查找链表中是否包含特定数据，若包含返回节点指针，否则返回 NULL。
* static LinkedList intersection(LinkedList& A, LinkedList& B)：计算两个链表的交集。
* static LinkedList difference(LinkedList& A, LinkedList& B)：计算两个链表的差集。
* static LinkedList unionSet(LinkedList& A, LinkedList& B)：计算两个链表的并集。

### 类外交互函数：

* void displayMenu(const LinkedList& A, const LinkedList& B) : 显示集合运算菜单，并展示当前 A 和 B 集合的元素。
* int oneadd(LinkedList& L, char aorb) : 添加单个元素到集合 L 中，aorb 用于标识是 A 集合还是 B 集合。
* void moreadd(LinkedList& T, char aorb) : 批量添加元素到临时集合 T 中，aorb 用于标识是 A 集合还是 B 集合，输入 “end” 表示结束批量添加。
* int moreaddchange(LinkedList& T) : 修改批量添加元素列表中的某个元素。
* int moreaddview(LinkedList& L, char aorb) : 提供一个视图界面，用于管理批量添加元素的操作，包括添加、修改和提交操作。
* int onechange(LinkedList& L, char aorb) : 修改集合 L 中的单个元素，aorb 用于标识是 A 集合还是 B 集合。
* int onedelete(LinkedList& L, char aorb) : 删除集合 L 中的单个元素，aorb 用于标识是 A 集合还是 B 集合。

## 4. 系统模块结构

根据题目分析及函数说明，本系统的模块结构图如下：

图1 系统模块结构图(\*号内容为拓展功能)

# 三、系统实现

各项功能实现的程序流程图如下所示：

## 集合信息录入

### 单个新增模式：

**开始(main)**

**结束**

**Temp==”1”**

**输入新增集合成员**

返回主页面

**选择新增集合对象**

进入单个新增函数

False True

图2-1 集合信息单个录入程序流程图(oneadd函数)

### \*批量新增模式：

**开始(main)**

**结束**

**Temp==”1”**

**While(输入新增集合成员!=”end”)**

返回主页面

**选择新增集合对象**

进入批量新增函数

False

True

==3放弃新增 ==1(继续新增)

**chose**

==2修改某个新增 ==0(提交新增并返回主页面)

return 0/1(成功/放弃修改)

Moreaddchange（）

图2-2 集合信息批量录入程序流程图(moreaddview函数)

## 集合成员删除\*

**开始(main)**

**结束**

**Temp==”1”**

**输入删除集合成员**

返回主页面

**选择删除集合对象**

进入单个删除函数

false

true

finddata()

return null

(未找到该成员)

return else(找到该成员)

value

N/n

Y/y

Deletedata()

图3 集合成员删除程序流程图(onedelete函数)

## 集合成员修改\*

**开始(main)**

**结束**

**Temp==”1”**

**输入修改集合成员**

返回主页面

**选择修改集合对象**

进入单个修改函数

false

true

finddata()

return null

(未找到该成员)

return find(找到该成员)

输入修改值value

find->data=value

修改成功

图3 集合成员修改程序流程图(onechange函数)

## 集合运算

**开始(main)**

**结束**

chose

==9 ==12

(A∩B)交集运算 ==10 ==11 (A∪B)并集运算

(A - B)运算 (B - A)差集运算

\*\*\*

LinkedList::difference(B, A)

LinkedList::difference(B, A)

LinkedList::difference(A, B)

LinkedList::intersection(A, B)

Display()

图3 集合各个运算程序流程图

## 核心函数

### Display()函数

**开始**

**结束**

**current?**

current = head

F

Cout<<”空”;

T

current != NULL

F

T

cout << current->data

current = current->next;

输出和推进指针

图5-1 显示函数流程图

### insert函数

传入const string& value

**开始**

**结束**

Node\* newNode = new Node;新建节点

newNode->data = value;

newNode->next = head;//头插

head = newNode;

Length++;//增加private成员

图5-2 插入链表函数流程图

### Deletedata函数

传入Node\* finded

**开始**

**结束**

**Fhead?**

Node\* Fhead = head

F

Return 0;

T

Fhead

F

T

False

Fhead->next == finded？

finded可能是第一个

Fhead==finded？

True 即找到finded的上一个

F

finded->next

F T

T 即finded是最后一个 即finded不是链表最后一个

head=finded->next;delete finded;

Fhead->next=NULL;

Node\* Flast=finded->next;//Flast指向后位delete finded;

Fhead->next=Flast;//next链上后位

Fhead=Fhead->next;//推进指针寻找

图5-3 删除函数流程图

### finddata函数

传入string value

**开始**

**结束**

Node\* Fhead = head;//存头节点

推进指针

Fhead=Fhead->next;

Fhead

False

True 未找到，返回空指针

Fhead->data == value

False

True

返回该值指针

Return Fhead

Return NULL

图5-4 查找链表成员函数流程图

### intersection函数

传入LinkedList& A，LinkedList& B

**开始**

**结束**

Node\* a =A.getHead();

拿A头指针

a != NULL

False

True 未找到，返回空指针

Node\* b = B.getHead();

拿B头指针

b != NULL

True

False

false

a->data == b->data

false

true

result.insert(a->data)

b = b->next;

a = a->next

图5-5 交集函数流程图

### difference函数

传入LinkedList& A，LinkedList& B

**开始**

**结束**

LinkedList result;

Node\* a =A.getHead();

创建结果实例

先拿A头指针

a != NULL

False

True 未找到，返回空指针

Node\* b = B.getHead();

bool found = false

循环拿B头指针

found做找到标记

b != NULL

True

False

false

a->data == b->data

false

Found

True

False 找到a,b相等 true

result.insert(a->data)

found = true

插入result中 标记

b = b->next;

a = a->next

图5-6 差集函数流程图

### unionSet函数

传入LinkedList& A，LinkedList& B

**开始**

**结束**

LinkedList result;

Node\* a =A.getHead();

创建结果实例

先拿A头指针

true

result.insert(a->data);

a = a->next;

a != NULL

False

遍历a中赋值给result对象

Node\* b = B.getHead()

拿B头指针遍历找result中没有的插入

b != NULL

False

True 未找到，返回空指针

Node\* temp = result.getHead();

bool found = false

循环拿result头指针

found做找到标记

temp != NULL

True

False

false

temp->data == b->data

false

Found

True

False 找到a,b相等 true

result.insert(b->data)

found = true

将没发现的插入result中 标记

b = b->next;

b = b->next

图5-7 并集函数流程图

# 四、程序测试与结果分析

## 程序运行步骤

1. 运行程序。
2. 选择合适的集合输入操作输入想要的集合，可以选择增删和修改的方式输入集合成员。
3. 选择需要执行的集合运算。
4. 程序会显示计算结果。

## 测试用例输入格式及说明

### 情况1.正常集合运算

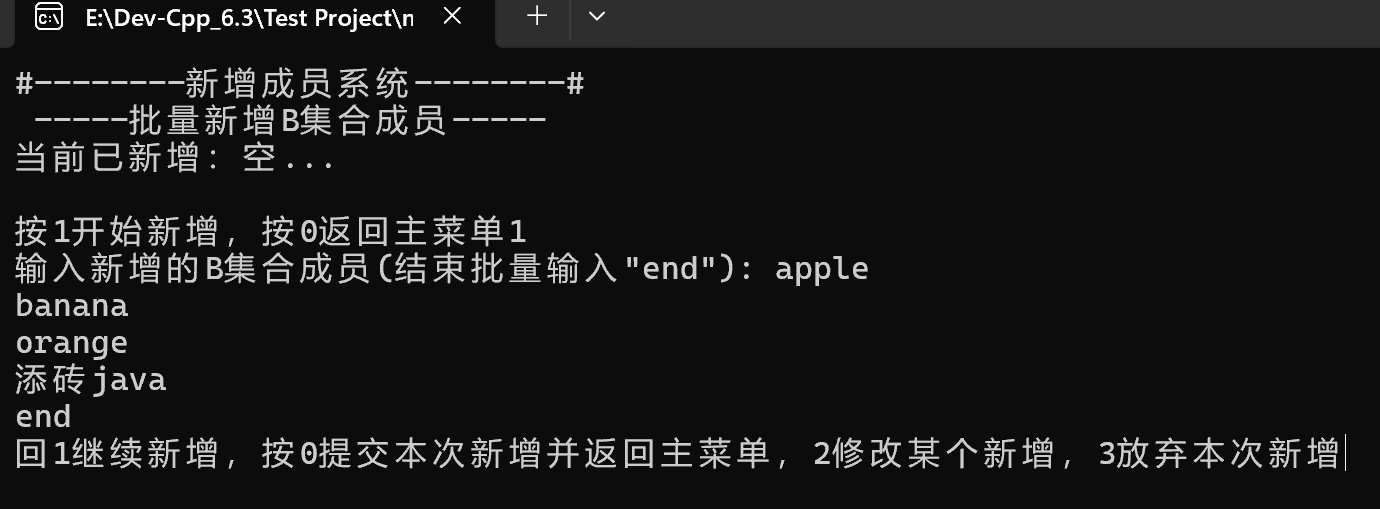
选择1-4的输入系统输入集合元素。



#### 输入A集合元素：集合元素以字符串形式输入，例如选择3进入批量添加A模式：“apple”, “banana”, “orange”,”三十3nnnV”最后以”end”结束输入。按0保存新增并返回主页面

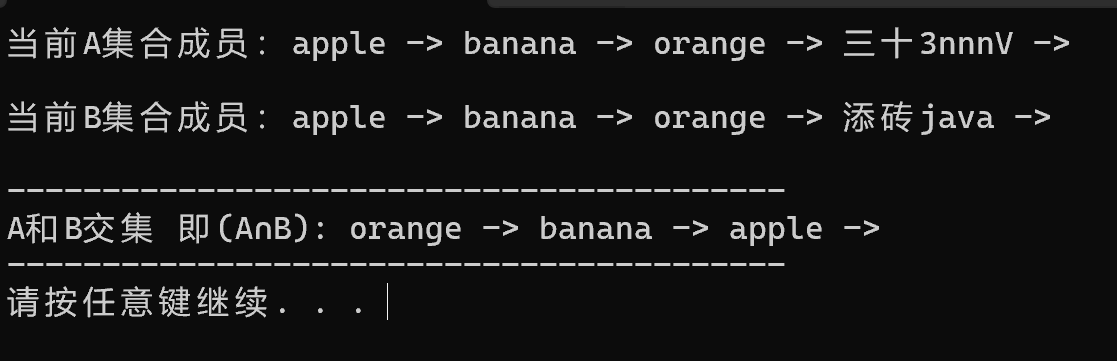


#### 输入B集合元素：集合元素以字符串形式输入，例如选择4进入批量添加B模式： “apple”, “banana”, “orange”,”添砖java”最后以”end”结束输入。按0保存新增并返回主页面



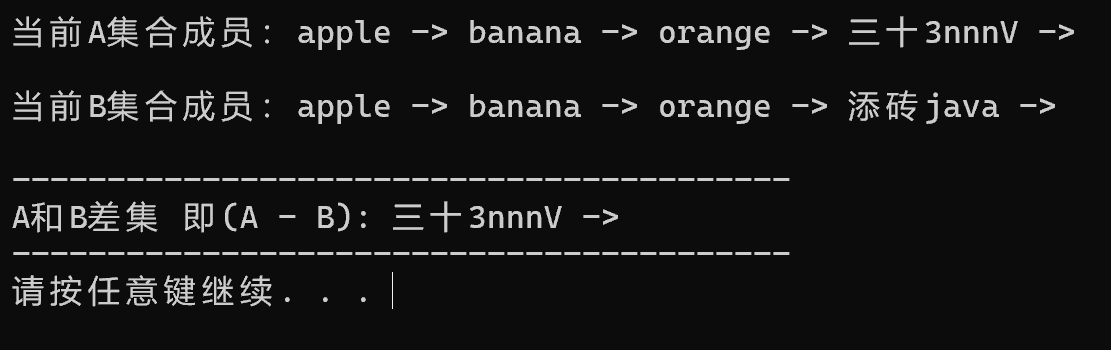
#### 输入选择的集合运算：选择菜单选项对应不同的运算。

##### 1)在上例的输入条件下，选择交集操作9



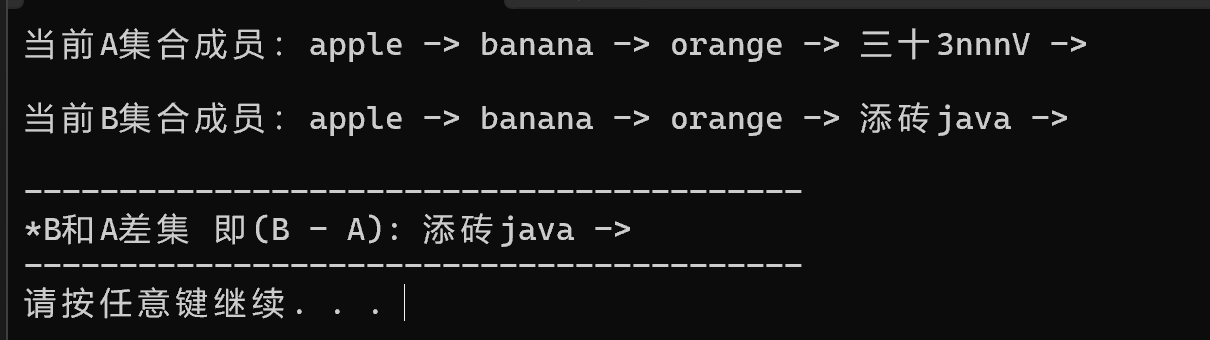
* 交集运算：两个集合 A 和 B中分别包含 “apple”, “banana”, “orange” ,”三十3nnnV”和 “apple”, “banana”, “orange”,”添砖java”。交集结果为 “apple”, “banana”, “orange”。

##### 2) 在上例的输入条件下，选择差集操作10



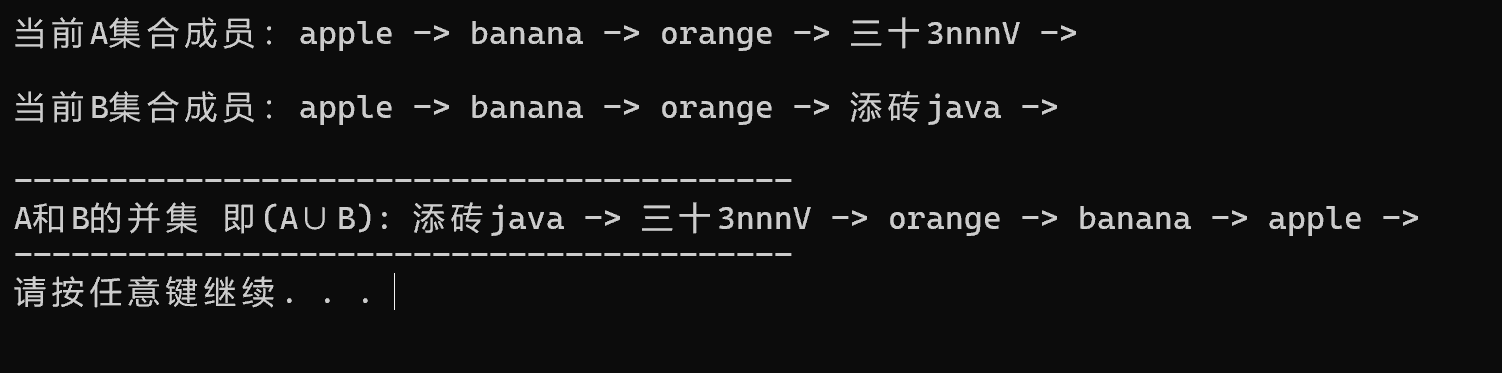
* 差集(A - B)运算：两个集合 A 和 B中分别包含 “apple”, “banana”, “orange” ,”三十3nnnV”和 “apple”, “banana”, “orange”,”添砖java”。差集(A - B)结果为 ”三十3nnnV”。

##### 3) \*在上例的输入条件下，选择差集操作11



* \*差集(B - A)运算：两个集合 A 和 B中分别包含 “apple”, “banana”, “orange” ,”三十3nnnV”和 “apple”, “banana”, “orange”,”添砖java”。差集(B - A)结果为 ”添砖java”。

##### 4) 在上例的输入条件下，选择并集操作12



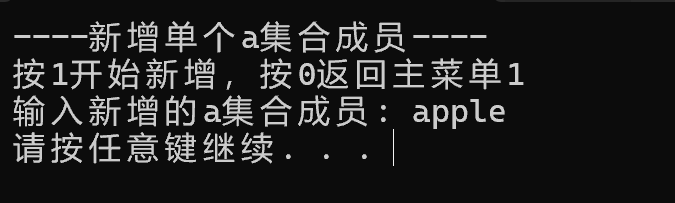
* 并集(A∪B)运算：两个集合 A 和 B中分别包含 “apple”, “banana”, “orange” ,”三十3nnnV”和 “apple”, “banana”, “orange”,”添砖java”。并集(A∪B)结果为 “apple”, “banana”, “orange” ,”三十3nnnV”,”添砖java”。

### 情况2.运算前的集合成员增删改：

#### <增>

##### 1.单个增

* 选择1-2的输入系统输入集合元素。集合元素以字符串形式输入，例如选择1进入单个添加A模式：“apple”



##### \*2.批量增及其中修改

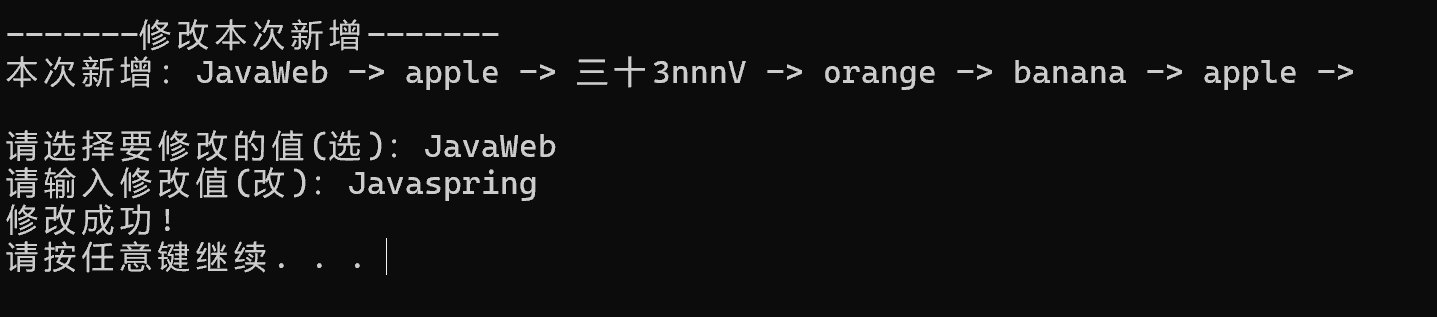
* 选择3-4的输入系统输入集合元素。集合元素以字符串形式输入，例如选择3进入批量添加A模式：“apple”, “banana”, “orange”,”三十3nnnV”最后以”end”结束输入。



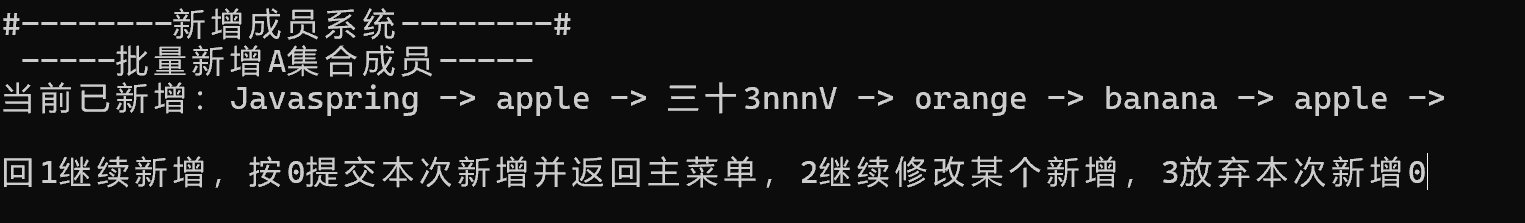
* 选择0,1,2,3进行下一步
* 选择0则保存新增更改，并返回主页面，结果与情况1中的输入A集合相同，这里不再演示。
* 选择1则继续新增，重复批量新增。例如再输入：“apple” ,”JavaWeb”最后以”end”结束输入。



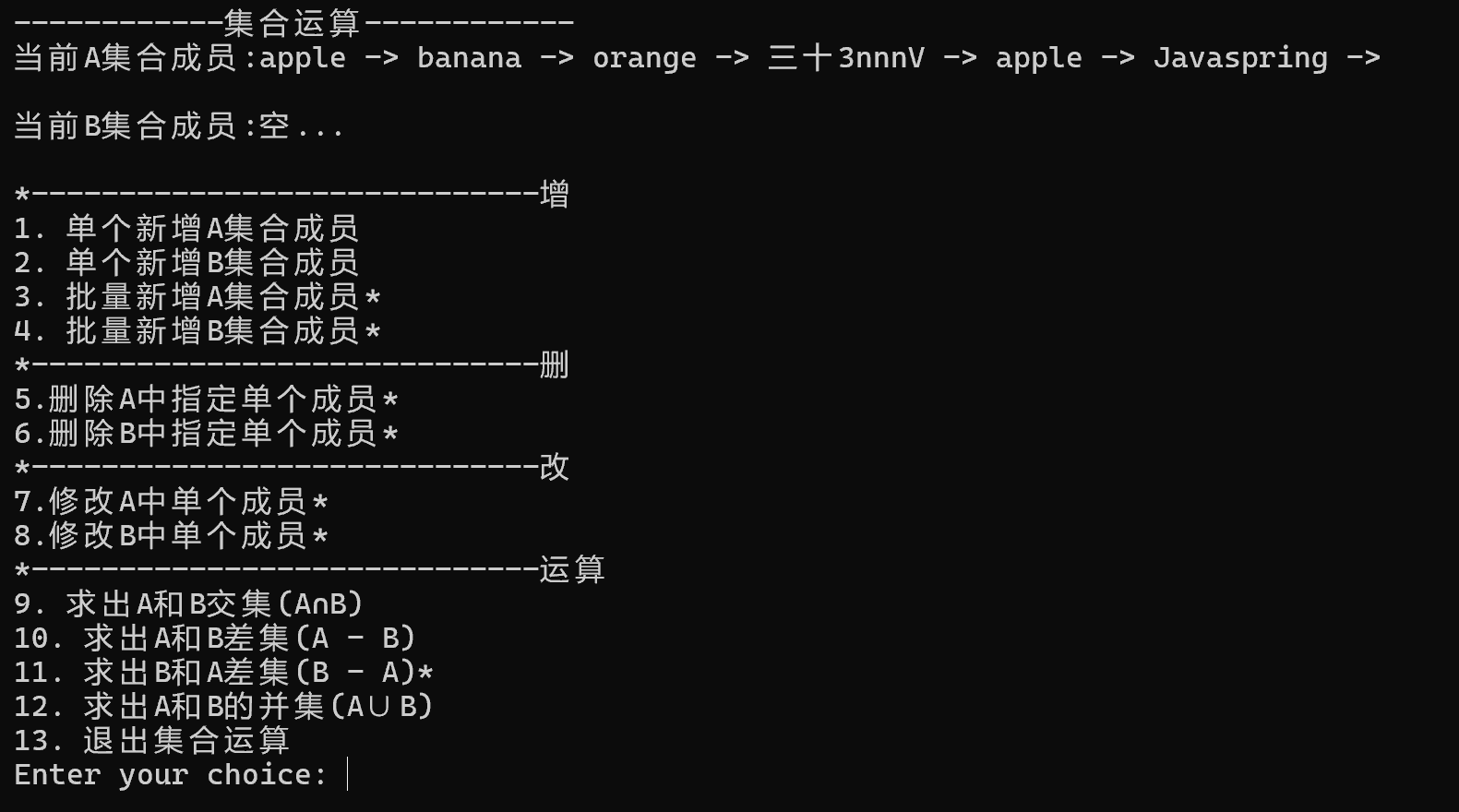
* 选择2对”JavaWeb”进行修改。输入2，输入”JavaWeb”,输入”Javaspring”。

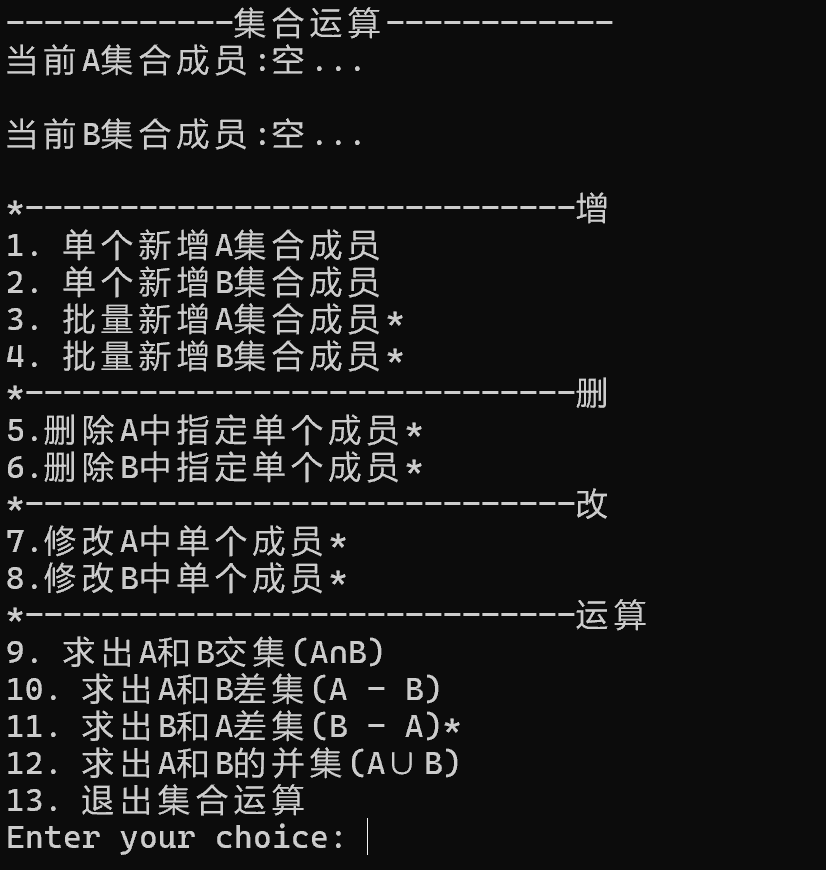


接下来选择0提交新增并返回主页面，集合显示新增成功。



选择0提交后主页面如下图：

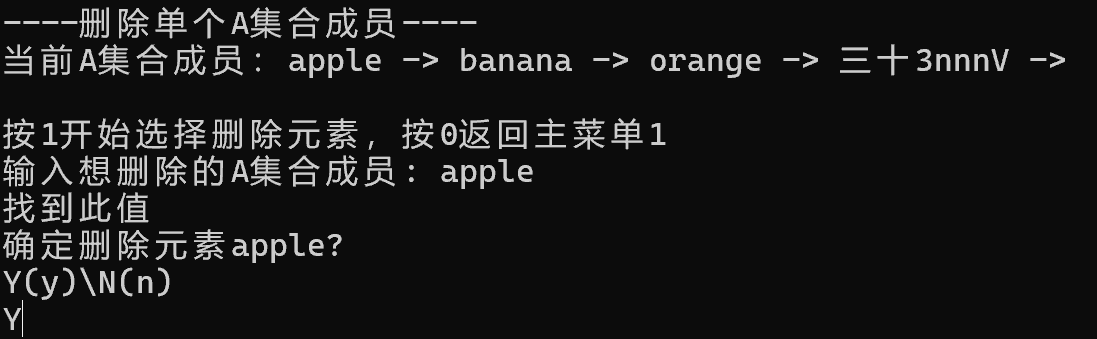
 此处如果选择3，也会返回主页面，但新增不会保存，结果如下图：



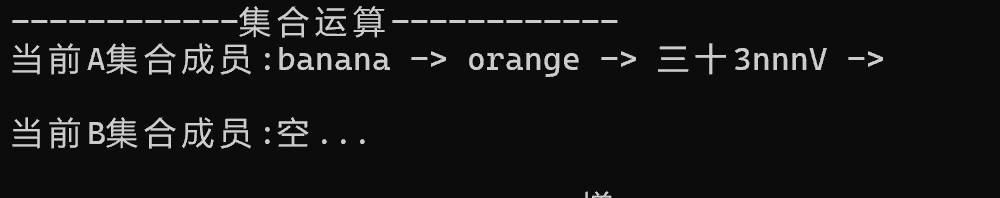
#### \*<删>

##### 删除指定集合的单个成员

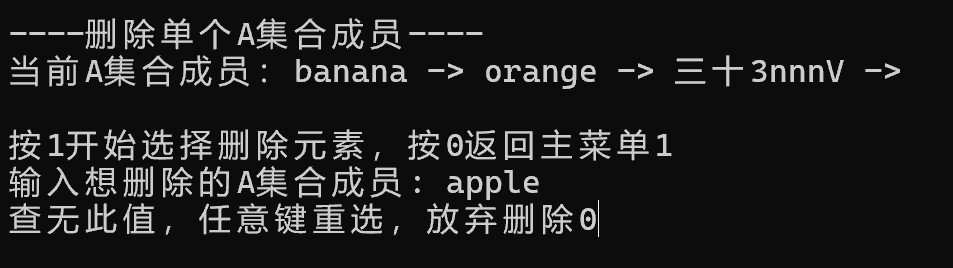
* 选择5-6的输入系统输入集合元素。集合元素以字符串形式输入例如在A集合中有:“apple”, “banana”, “orange”,”三十3nnnV”的情况下，选择5进入删除A模式，输入“apple”,显示”找到该值”，后选择”Y”。



结果如下：



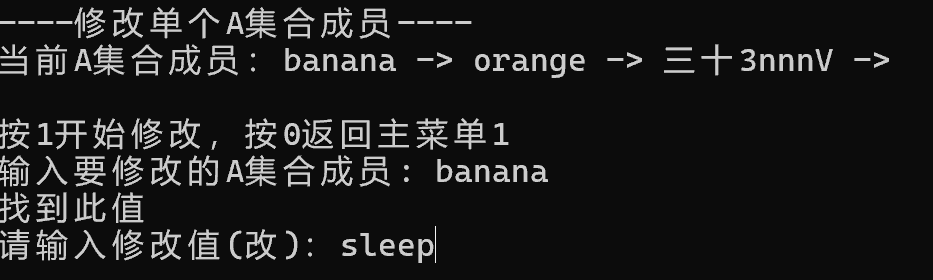
如果再次选择5进入删除A模式，输入“apple”,显示”查无此值”,可以任意键重选或者回0放弃删除返回主页面。



#### \*<改>

##### 修改指定集合的单个成员

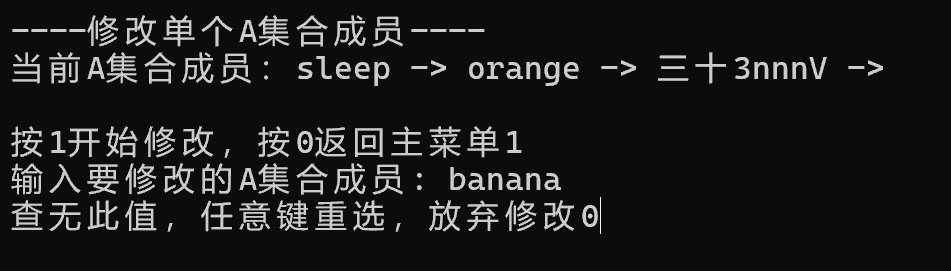
* 选择7-8的输入系统输入集合元素。集合元素以字符串形式输入，例如在A集合中有: “banana”, “orange”,”三十3nnnV”的情况下，选择7进入修改A模式，输入“banana”,显示”找到该值”，后输入修改值”sleep”，修改成功。



结果如下：



如果再次选择7进入修改A模式，输入“banana”,显示”查无此值”,可以任意键重选或者回0放弃删除返回主页面。



# 五、总结

系统优势

* 代码结构清晰，易于理解和维护。
* 功能完善，能够实现基本的集合运算。
* 用户界面简单易用，方便用户进行操作。
* 使用了类封装集合基本方法，便于之后的功能升级，提高了代码复用性和可维护性
* 在oneadd、moreadd 、onechange、onedelete函数输入中使用string作为输入类型提高了函数中抗乱输入能力:，

系统不足

* 缺乏对内存泄漏的处理机制，可能会导致程序运行效率低下。
* 用户界面设计较为简陋，可以进一步完善。
* 页面逻辑跳转代码使用了goto语句，可能会导致部分读者不理解跳转逻辑。
* 使用string作为输入类型，需要对每种输入情况使用if判断不能用switch
* 部分函数缺乏对输入数据类型和格式的验证，可能会导致程序运行错误。在部分函数如main、moreaddchange、moreaddview函数因输入情况过多，仍采用switch判断，抗乱输入能力不足。
* 部分代码可能在devcpp有不识别问题：开发过程中moreadd函数对错误输入的处理部分代码：” cin.ignore(numeric\_limits<streamsize>::max(), '\n');”有概率报错” streamsize”和” numeric\_limits”不存在std空间中，虽然经查询” streamsize”其实就是存在于std中，” numeric\_limits”的类型包含于<limits>头文件中，且在报错时头文件已引用std已声明，但考虑到编译过程出现过报错，如果在编译过程中出现类似报错，请尝试注释该行代码。

设想和可扩展功能

* 增加对数据类型和格式的验证，提高程序的健壮性。
* 增加对内存泄漏的处理机制，提高程序的运行效率。
* 设计更加完善的用户界面，增加更多功能，例如撤销、重做等操作。
* 扩展集合运算功能，例如对称差集、补集等