

中南林业科技大学

课程设计报告

设计名称： 集合运算

姓 名： 段志超 学 号： 20192902

专业班级： 2019 级计算机科学与技术 3 班

系（院）： 计算机与信息工程学院

设计时间： 2020~2021 学年第一学期

设计地点： 电子信息楼 机房

指导教师评语：

成绩：

项目	内容	得分
答辩情况	把握问题、分析问题以及解决问题等诸多方面的表达能力（20%）	
程序正确性	包括程序整体结构是否合理、编程风格是否规范等（30%）	
工作态度	学生的工作态度、独立工作能力（30%）	
课程设计报告	课程设计报告（含课程设计心得）（20%）	
总分		

签名： _____

年 月 日

1. 课程设计目的

- 1、训练学生灵活应用所学数据结构知识，独立完成问题分析，结合数据结构理论知识，编写程序求解指定问题。
2. 初步掌握软件开发过程的问题分析、系统设计、程序编码、测试等基本方法和技能;
3. 提高综合运用所学的理论知识和方法独立分析和解决问题的能力;
4. 训练用系统的观点和软件开发一般规范进行软件开发，巩固、深化学生的理论知识，提高编程水平，并在此过程中培养他们严谨的科学态度和良好的工作作风。

2. 课程设计任务与要求:

任务

集合运算

功能：使用链表来表示集合，完成集合的交、并、差等操作。

主要包含以下内容：

1. 初步完成总体设计，搭好框架，确定人机对话的界面，确定函数个数；
2. 完成最低要求：完成集合的各种基本运算；

要求：

1、在处理每个题目时，要求从分析题目的需求入手，按设计抽象数据类型、构思算法、通过设计实现抽象数据类型、编制上机程序和上机调试等若干步骤完成题目，最终写出完整的分析报告。前期准备工作完备与否直接影响到后序上机调试工作的效率。在程序设计阶段应尽量利用已有的标准函数，加大代码的重用率。

2、设计的题目要求达到一定工作量（300 行以上代码），并具有一定的深度和难度。

3、程序设计语言推荐使用 C/C++，程序书写规范，源程序需加必要的注释；

4、每位同学需提交可独立运行的程序；

5、每位同学需独立提交设计报告书（每人一份），要求编排格式统一、规范、内容充实，不少于 10 页（代码不算）；

6、课程设计实践作为培养学生动手能力的一种手段，单独考核。

3. 课程设计说明书

一 需求分析

对于集合交、并、差等运算的实现比较简单。

为了实现不同类型数据的集合运算, 需要使用 string 类型, string 类是 STL 中封装好的类。在自定义了单链表结构的基础上, 使用 string 而不是自定义字符串结构可以有效避免数据结构的复杂性。但需要对 STL 中容器操作的使用和原理较为熟悉, 才能定义高效实用且符合要求的数据结构。

要求用到数据结构课上学到的线性表的知识, 所以就要充分而清晰的理解关于线性表的知识。

要求实现的基本功能有集合的交、并、补、差运算, 如果基于链表实现, 需要对链表的元素插入、元素删除、链表合并等操作很熟练。而且在此基础上, 为了完整实现各功能, 还需要熟练掌握链表的其他操作。

这个课题的核心实现集中在对链表的操作上, 但除此之外还要设计完好的用户交互界面, 这需要熟练掌握语言方面的格式化输入输出。

综上, 做这个课题, 要具备的知识就是线性表的基本算法, 链表的基本操作, 必要的 C 或者 C++ 知识 (由于该题的数据结构并不复杂, 且核心便是链表, 因此我基本采用 c 语言来实现), 以及丰富的程序调适经验。

二 概要设计

我将数据结构的定义以及相关算法函数的声明放在头文件 set.h 中, 并将函数的定义以及算法的实现放在 cpp 文件 set.cpp 中, 而将主函数以及用户交互界面相关控制函数放在 main.cpp 中。

考虑到该课题的核心便在于链表的各种操作运算, 且总体上并不复杂, 我基本采用 c 语言来实现了集合数据结构的构建, 为了方便, 只适当地使用了部分 c++ 特有的特性, 因此为 cpp 文件。

对于集合实现, 我使用单链表。对于每一个节点, 由 data 数据域及 next 指针域组成。为了实现不同类型元素的集合运算, 我使用 string 类型作为数据域的类型, 即集合元素的类型。除了将每一个元素和其后继元素的指针封装为一个元素结点 struct 外, 我还将链表的头节点、尾节点、链表长度封装为一个集合 struct。从而方便对链表操作。

集合的数据结构定义如下:

```
typedef string ElemType;

typedef struct Node {
    ElemType data; // 数据元素
    struct Node* next; // 下一个节点的地址
} Node, * PNode; // 单个节点的结构体

typedef struct List {
    PNode head; // 集合的头节点
    PNode tail; // 集合中的最后一个元素所在节点
    int length; // 集合中元素的个数
} List, * PList;
```

对于集合各项运算, 我基于单链表实现。对于单链表的各运算算法, 除了基本的操作外, 为了方便后续集合算法的实现, 我新增了如链表排序等函数。

各函数原型及其功能、使用说明如下:

```
/* 创建集合 */
bool CreateSet(List& L);
/* 创建集合_重载 */
```

```
bool CreateSet(List& L, ElemType* s, int length);

/* 销毁集合 */
bool DestroySet(List& L);

/* 并集 */
bool Union(List L1, List L2, List& L3);

/* 交集 */
bool Intersection(List L1, List L2, List& L3);

/* 补集 */
bool Complement(List L1, List L2, List& L3);

/* 差集 */
bool Difference(List L1, List L2, List& L3);

/* 拷贝线性表 */
bool Copy(List L1, List& L2);

/* 求线性表长度 */
bool ListLength(List L, int& index);

/* 初始化线性表 */
bool InitList(List& L); // 初始化线性表

/* 销毁线性表 */
bool DestroyList(List& L); // 销毁线性表

/* 清空线性表 */
bool ClearList(List& L); // 清空线性表

/* 操作:          获取线性表index位置的元素          */
/*          L 指向一个已初始化的链表          */
/*          index 为查找位置          */
/*          e 返回第i个位置元素的值          */
/* 后置条件:      查找成功返回true, 查找失败返回false */
bool GetElem(List L, int index, ElemType& e); // 获取线性表index位置的元素

/* 操作:          查找线性表中值为e的元素          */
/*          L 指向一个已初始化的链表          */
/*          e 为要查找的值          */
/* 后置条件:      查找成功返回元素位置, 查找失败返回 0 */
bool LocateElem(List L, ElemType e); // 判断e是否在L中, 不在返回false
```

```
/* 操作:          添加e到线性表末尾          */
/*              L 指向一个已初始化的链表      */
/*              e 为要查找的值                  */
/* 后置条件:      查找成功返回元素位置, 查找失败返回 0 */
bool Append(List& L, ElemType e); //添加e到线性表末尾

/* 操作:          在线性表第i个位置插入元素      */
/*              L 指向一个已初始化的链表      */
/*              index 为要插入的位置            */
/*              e 为要插入的值                  */
/* 后置条件:      插入成功返回true, 插入失败返回false */
bool InsertElem(List& L, int index, ElemType e); //插入e到index位置

/* 操作:          删除线性表中值为e的元素        */
/*              L 指向一个已初始化的链表      */
/*              e 为要删除的元素值              */
/* 后置条件:      删除成功返回true, 删除失败返回false */
bool removeElem(List& L, ElemType e); //删除线性表中e元素

/* 操作:          删除线性表中第i个元素          */
/*              L 指向一个已初始化的链表      */
/*              index 为要删除的位置            */
/* 后置条件:      删除成功返回true, 删除失败返回false */
bool removeIndex(List& L, int index); //删除线性表中index号元素

/* 操作:          将集合元素排序                */
/*              L 指向一个已初始化的集合      */
/* 后置条件:      集合元素递增排列              */
bool sort(List& L); //给线性表排序, 默认升序

/* 输出链表 */
bool printList(List L);

/* 操作:          判断集合中元素是否是数字      */
/*              L 为需要判断的集合            */
/* 后置条件:      是数字, 返回true; 否则返回false */
bool isNum(string str);
```

三 详细设计

1) 宏定义, 定义全局变量、结构体:

```
/* 特定程序的声明 */
#define ERROR false
#define OK true
#define TRUE true
#define FALSE false

/* 一般类型定义 */
typedef string ElemType;

typedef struct Node {
    ElemType data; // 数据元素
    struct Node* next; // 下一个节点的地址
} Node, * PNode; // 单个节点的结构体

typedef struct List {
    PNode head; // 集合的头节点
    PNode tail; // 集合中的最后一个元素所在节点
    int length; // 集合中元素的个数
} List, * PList;
```

2) 主函数处理输入的算法如下:

```
while (true) {
    cout << "\n请选择操作: ";
    int ch;
    cin >> ch;
    switch (ch) {
        case 1:
            Intersection(set1, set2, set3);
            cout << set3; ClearList(set3);
            break; // 交集
        case 2:
            Union(set1, set2, set3);
            cout << set3; ClearList(set3);
            break; // 并集
        case 3:
            Complement(set1, set2, set3);
            cout << set3; ClearList(set3);
            break; // !A (AUB为全集)
        case 4:
            Complement(set2, set1, set3);
            cout << set3; ClearList(set3);
            break; // !B (AUB为全集)
```

```
case 5:
    Difference(set1, set2, set3);
    cout << set3; ClearList(set3);
    break;//A-B
case 6:
    Difference(set2, set1, set3);
    cout << set3; ClearList(set3);
    break;//B-A
case 7:
    DestroySet(set1);
    DestroySet(set2);
    DestroySet(set3);
    cout << endl;
    goto Lab;//重新输入集合
case 8:
    DestroySet(set1);
    DestroySet(set2);
    DestroySet(set3);
    return 0;//退出
default:
    cout << "操作有误!";
    break;
}
}
```

3) 创建集合的算法如下:

```
void inputSet(List& set) {
    ElemType* deal;
    int length;
    while (true) {
        string str;
        cin >> str;

        if (str.length() == 2) {
            deal = new string[0];
            length = 0;
            break;
        }

        int former = str.find("{");
        int later = str.find("}");
        int count_former = count(str.begin(), str.end(), '{');
        int count_later = count(str.begin(), str.end(), '}');
        int count_Elem = count(str.begin(), str.end(), ',') + 1;
```

```
//检查输入合法性
if (former != 0 || later != str.length() - 1 || count_former != 1 || count_later != 1) {
    cout << "输入有误:请重新输入:";
}
else {
    deal = new string[count_Elem];

    str[0] = ',';
    str[str.length() - 1] = ',';
    //提取元素
    for (unsigned current = 0, next, i = 0; current < str.length() && i < count_Elem; i++) {
        next = str.find_first_of(',', current + 1);
        deal[i] = str.substr(current + 1, next - current - 1);
        current = next;
    }
    length = count_Elem;

    break;
}

CreateSet(set, deal, length);
delete[] deal;
}
```

4) 实现集合排序的算法如下:

```
bool sort(List& L) {
    int flag = 1;
    ElemType temp;
    PNode current;

    current = L.head->next;
    for (int i = 1; i < L.length; i++) {
        if (!isNum(current->data)) {
            flag = 0;
            break;
        }
        current = current->next;
    }

    if (flag) { //数字排序
        for (int i = L.length; i >= 2; i--) {
            current = L.head->next;
            for (int j = 1; j < i; j++) {
```



```
        if (stringToNum(current->data) > stringToNum(current->next->data)) {
            temp = current->data;
            current->data = current->next->data;
            current->next->data = temp;
        }
        current = current->next;
    }
}

else { //字符串排序
    for (int i = L.length; i >= 2; i--) {
        current = L.head->next;
        for (int j = 1; j < i; j++) {
            if (current->data > current->next->data) {
                temp = current->data;
                current->data = current->next->data;
                current->next->data = temp;
            }
            current = current->next;
        }
    }
}

return OK;
}
```

5) 实现集合的交、并、差、补算法如下:

// 求两集合的补集

```
bool Complement(List L1, List L2, List& L3) {
    PNode node_l2 = L2.head;

    for (int i = 1; i <= L2.length; i++) {
        node_l2 = node_l2->next;
        if (!LocateElem(L1, node_l2->data)) {
            Append(L3, node_l2->data);
        }
    }

    sort(L3);
    return OK;
}
```

//求两集合的差集

```
bool Difference(List L1, List L2, List& L3) {
    PNode node_l1 = L1.head;
```

```
for (int i = 1; i <= L1.length; i++) {  
    node_l1 = node_l1->next;  
    if (!LocateElem(L2, node_l1->data)) {  
        Append(L3, node_l1->data);  
    }  
}  
sort(L3);  
return OK;  
}
```

//求两集合的交集

```
bool Intersection(List L1, List L2, List& L3) {  
    PNode node_b = L2.head;  
  
    for (int i = 1; i <= L2.length; i++) {  
        node_b = node_b->next;  
        if (LocateElem(L1, node_b->data)) {  
            Append(L3, node_b->data);  
        }  
    }  
    sort(L3);  
    return OK;  
}
```

//求两集合并集

```
bool Union(List L1, List L2, List& L3) {  
    PNode node_l2 = L2.head;  
    Copy(L1, L3);  
  
    for (int i = 1; i <= L2.length; i++) {  
        node_l2 = node_l2->next;  
        if (!LocateElem(L3, node_l2->data)) {  
            Append(L3, node_l2->data);  
        }  
    }  
    sort(L3);  
    return OK;  
}
```

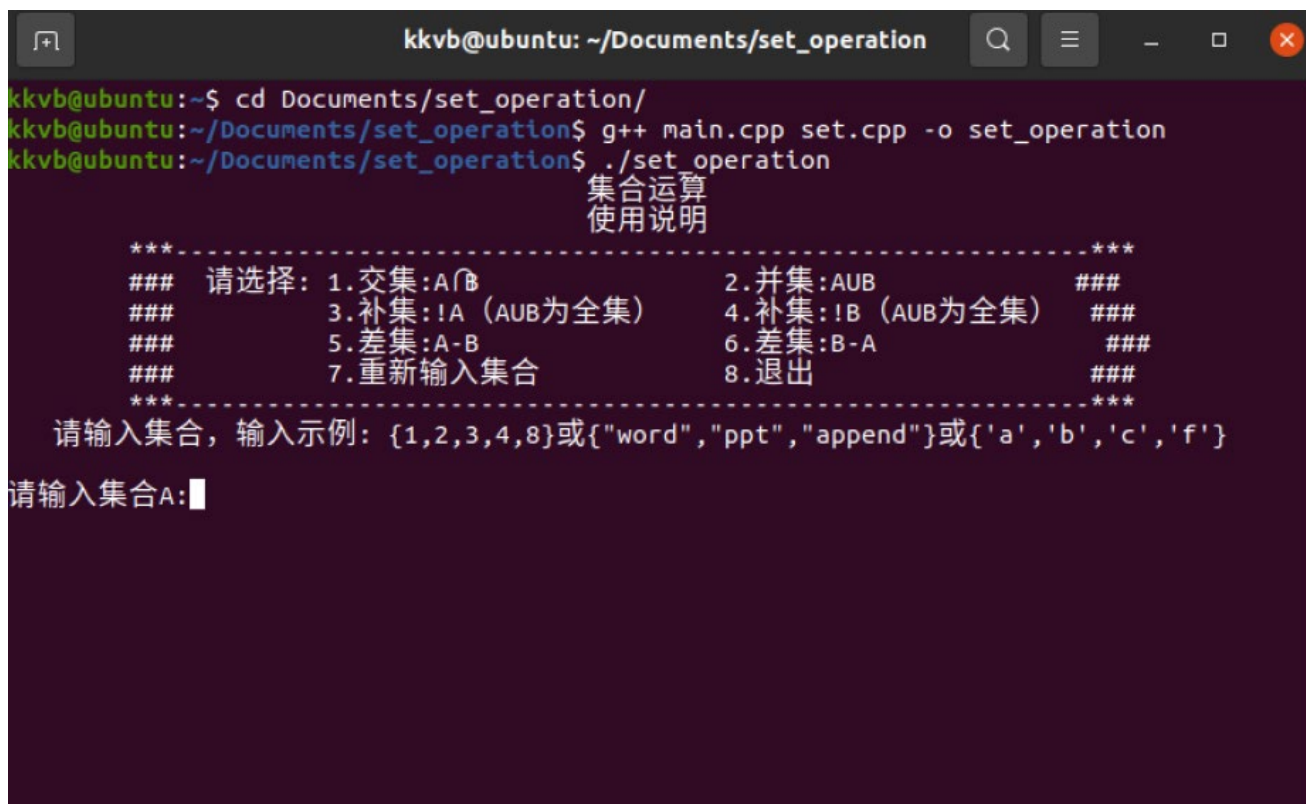
四 设计与调试分析

从上面的算法和调用关系可以看出, 这个程序的基本样子已经非常的清楚, 但是真正的程序中还要考虑各种限制条件。例如在查找的过程中, 可能不存在要查找的信息, 就要给出不存在此信息的提示等。

还有就是涉及到返回值得问题和程序中所要用到的变量的问题。在调试的过程中所遇到的问题很多, 其中最难的两个问题是给出的六个警告错误, 在 5.0 的版本下就是两个使得程序无法运行的错误。其中五个是因为在变量的声明中因为使用了浮点型和双精度型两种而造成在某些模块中两个变量相互赋值时类型不匹配。另一个是在查找的子函数中忘记了需要的返回语句, `return 1`。

五 用户手册

- 1 本程序可以在 VisualStudio2019 环境下运行。
- 2 在 Visual Studio 中打开该项目, 直接运行即可。
- 3 本项目所有源程序完全符合 c11/c++11 标准, 因此也可以使用支持该标准的其他编译器如 g++ 编译通过并生成可执行文件。
- 4 务必将 main.cpp 和 set.cpp 联合编译, 以下为 gcc9.3.0 示例, 运行开始界面如下:



```
kkvb@ubuntu: ~/Documents/set_operation
kkvb@ubuntu:~/Documents/set_operation$ g++ main.cpp set.cpp -o set_operation
kkvb@ubuntu:~/Documents/set_operation$ ./set_operation
集合运算
使用说明
***-----***
### 请选择: 1. 交集:  $A \cap B$                 2. 并集:  $A \cup B$                 ###
###          3. 补集:  $\neg A$  ( $A \cup B$  为全集)      4. 补集:  $\neg B$  ( $A \cup B$  为全集)  ###
###          5. 差集:  $A - B$                 6. 差集:  $B - A$                 ###
###          7. 重新输入集合                8. 退出                    ###
***-----***
请输入集合, 输入示例: {1,2,3,4,8}或{"word","ppt","append"}或{'a','b','c','f'}
请输入集合A: █
```

六 测试成果

程序各项功能测试:

```
kkvb@ubuntu: ~/Documents/set_operation
kkvb@ubuntu:~$ cd Documents/set_operation/
kkvb@ubuntu:~/Documents/set_operation$ g++ main.cpp set.cpp -o set_operation
kkvb@ubuntu:~/Documents/set_operation$ ./set_operation
集合运算
使用说明
***-----***
### 请选择: 1.交集:A∩B          2.并集:A∪B          ###
###          3.补集:!A (A∪B为全集)  4.补集:!B (A∪B为全集)  ###
###          5.差集:A-B            6.差集:B-A            ###
###          7.重新输入集合        8.退出                ###
***-----***
请输入集合, 输入示例: {1,2,3,4,8}或{"word","ppt","append"}或{'a','b','c','f'}

请输入集合A:{1,2,3,4,5,'a','b','c',"phone","apple","huawei"}
请输入集合B:{4,5,6,7,8,'c','x','y',"phone","xiaomi","samsung"}

集合输入完成, 请继续操作

请选择操作: 1
{"phone",'c',4,5}

请选择操作: 2
{"apple","huawei","phone","samsung","xiaomi",'a','b','c','x','y',1,2,3,4,5,6,7,8}

请选择操作: 3
{"samsung","xiaomi",'x','y',6,7,8}

请选择操作: 4
{"apple","huawei",'a','b',1,2,3}

请选择操作: 5
{"apple","huawei",'a','b',1,2,3}

请选择操作: 6
{"samsung","xiaomi",'x','y',6,7,8}

请选择操作: 7

    请输入集合, 输入示例: {1,2,3,4,8}或{"word","ppt","append"}或{'a','b','c','f'}

请输入集合A:8
输入有误:请重新输入:{1,2,3}
请输入集合B:{4,5,6}

集合输入完成, 请继续操作

请选择操作: 8
kkvb@ubuntu:~/Documents/set_operation$
```

错误输入测试:

```

kkvb@ubuntu: ~/Documents/set_operation
kkvb@ubuntu:~$ cd Documents/set_operation/
kkvb@ubuntu:~/Documents/set_operation$ g++ main.cpp set.cpp -o set_operation
kkvb@ubuntu:~/Documents/set_operation$ ./set_operation
集合运算
使用说明
***-----***
### 请选择: 1.交集:A∩B          2.并集:A∪B          ###
###          3.补集:!A (A∪B为全集)  4.补集:!B (A∪B为全集)  ###
###          5.差集:A-B              6.差集:B-A          ###
###          7.重新输入集合          8.退出              ###
***-----***
请输入集合, 输入示例: {1,2,3,4,8}或{"word","ppt","append"}或{'a','b','c','f'}

请输入集合A:{1,2,3
输入有误:请重新输入:{1,2,3}
请输入集合B:4,5,6
输入有误:请重新输入:{4,5,6}

集合输入完成, 请继续操作

请选择操作: 8
kkvb@ubuntu:~/Documents/set_operation$

```

七 附录（源程序清单）

/*计算机科学与技术3班 段志超的课程设计题目, 选题是: 集合运算*/

set.h

```

1.  /* set.h -- 集合类型的头文件 */
2.  #ifndef SET_H_
3.  #define SET_H_
4.
5.  #include <string>
6.
7.  /* 特定程序的声明 */
8.  #define ERROR false
9.  #define OK true
10. #define TRUE true
11. #define FALSE false
12. using namespace std;
13.
14. /* 一般类型定义 */
15. typedef string ElemType;
16.

```

```
17. typedef struct Node {
18.     ElemType data; //数据元素
19.     struct Node* next; //下一个节点的地址
20. }Node, * PNode; //单个节点的结构体
21.
22. typedef struct List {
23.     PNode head; //集合的头节点
24.     PNode tail; //集合中的最后一个元素所在节点
25.     int length; //集合中元素的个数
26. }List, * PList;
27.
28.
29. /* 函数原型 */
30.
31. /* 创建集合 */
32. bool CreateSet(List& L);
33. /* 创建集合_重载 */
34. bool CreateSet(List& L, ElemType* s, int length);
35.
36. /* 销毁集合 */
37. bool DestroySet(List& L);
38.
39. /* 并集 */
40. bool Union(List L1, List L2, List& L3);
41.
42. /* 交集 */
43. bool Intersection(List L1, List L2, List& L3);
44.
45. /* 补集 */
46. bool Complement(List L1, List L2, List& L3);
47.
48. /* 差集 */
49. bool Difference(List L1, List L2, List& L3);
50.
51.
52.
53. /* 拷贝线性表 */
54. bool Copy(List L1, List& L2);
55.
56. /* 求线性表长度 */
57. bool ListLength(List L, int& index);
58.
59. /* 初始化线性表 */
60. bool InitList(List& L); //初始化线性表
61.
```



```
62. /* 销毁线性表 */
63. bool DestroyList(List& L); //销毁线性表
64.
65. /* 清空线性表 */
66. bool ClearList(List& L); //清空线性表
67.
68.
69. /* 操作:      获取线性表 index 位置的元素      */
70. /*      L 指向一个已初始化的链表      */
71. /*      index 为查找位置      */
72. /*      e 返回第 i 个位置元素的值      */
73. /* 后置条件:  查找成功返回 true, 查找失败返回 false */
74. bool GetElem(List L, int index, ElemType& e); //获取线性表 index 位置的元素
75.
76. /* 操作:      查找线性表中值为 e 的元素      */
77. /*      L 指向一个已初始化的链表      */
78. /*      e 为要查找的值      */
79. /* 后置条件:  查找成功返回元素位置, 查找失败返回 0 */
80. bool LocateElem(List L, ElemType e); //判断 e 是否在 L 中, 不在返回 false
81.
82. /* 操作:      添加 e 到线性表末尾      */
83. /*      L 指向一个已初始化的链表      */
84. /*      e 为要查找的值      */
85. /* 后置条件:  查找成功返回元素位置, 查找失败返回 0 */
86. bool Append(List& L, ElemType e); //添加 e 到线性表末尾
87.
88. /* 操作:      在线性表第 i 个位置插入元素      */
89. /*      L 指向一个已初始化的链表      */
90. /*      index 为要插入的位置      */
91. /*      e 为要插入的值      */
92. /* 后置条件:  插入成功返回 true, 插入失败返回 false */
93. bool InsertElem(List& L, int index, ElemType e); //插入 e 到 index 位置
94.
95. /* 操作:      删除线性表中值为 e 的元素      */
96. /*      L 指向一个已初始化的链表      */
97. /*      e 为要删除的元素值      */
98. /* 后置条件:  删除成功返回 true, 删除失败返回 false */
99. bool removeElem(List& L, ElemType e); //删除线性表中 e 元素
100.
101. /* 操作:      删除线性表中第 i 个元素      */
102. /*      L 指向一个已初始化的链表      */
103. /*      index 为要删除的位置      */
104. /* 后置条件:  删除成功返回 true, 删除失败返回 false */
105. bool removeIndex(List& L, int index); //删除线性表中 index 号元素
106.
```

```
107. /* 操作:      将集合元素排序      */
108. /*      L 指向一个已初始化的集合      */
109. /* 后置条件:    集合元素递增排列      */
110. bool sort(List& L); //给线性表排序, 默认升序
111.
112. /* 输出链表 */
113. bool printList(List L);
114.
115. /* 操作:      判断集合中元素是否是数字      */
116. /*      L 为需要判断的集合      */
117. /* 后置条件:    是数字, 返回 true; 否则返回 false      */
118. bool isNum(string str);
119.
120.
121. #endif
```

set.cpp

```
1. #include <sstream>
2. #include <iostream>
3. #include "set.h"
4.
5.
6.
7. bool Copy(List L1, List& L2) {
8.     if (!L1.head) {
9.         return ERROR;
10.    }
11.    PNode current = L1.head->next;
12.    while (current) {
13.        Append(L2, current->data);
14.        current = current->next;
15.    }
16.
17.    return OK;
18. }
19.
20. // 求两集合的补集
21. bool Complement(List L1, List L2, List& L3) {
22.     PNode node_l2 = L2.head;
23.
24.     for (int i = 1; i <= L2.length; i++) {
25.         node_l2 = node_l2->next;
26.         if (!LocateElem(L1, node_l2->data)) {
```



```
27.         Append(L3, node_l2->data);
28.     }
29. }
30. sort(L3);
31. return OK;
32. }
33.
34. //求两集合的差集
35. bool Difference(List L1, List L2, List& L3) {
36.     PNode node_l1 = L1.head;
37.
38.     for (int i = 1; i <= L1.length; i++) {
39.         node_l1 = node_l1->next;
40.         if (!LocateElem(L2, node_l1->data)) {
41.             Append(L3, node_l1->data);
42.         }
43.     }
44.     sort(L3);
45.     return OK;
46. }
47.
48. //求两集合的交集
49. bool Intersection(List L1, List L2, List& L3) {
50.     PNode node_b = L2.head;
51.
52.     for (int i = 1; i <= L2.length; i++) {
53.         node_b = node_b->next;
54.         if (LocateElem(L1, node_b->data)) {
55.             Append(L3, node_b->data);
56.         }
57.     }
58.     sort(L3);
59.     return OK;
60. }
61.
62. //求两集合并集
63. bool Union(List L1, List L2, List& L3) {
64.     PNode node_l2 = L2.head;
65.     Copy(L1, L3);
66.
67.     for (int i = 1; i <= L2.length; i++) {
68.         node_l2 = node_l2->next;
69.         if (!LocateElem(L3, node_l2->data)) {
70.             Append(L3, node_l2->data);
71.         }
```

```
72.     }
73.     sort(L3);
74.     return OK;
75.
76. }
77.
78. //创建新的集合
79. bool CreateSet(List& L, ElemType* s, int length) {
80.     if (InitList(L)) {
81.         for (int i = 0; i < length; i++) {
82.             if (!LocateElem(L, s[i])) {
83.                 Append(L, s[i]);
84.             }
85.         }
86.         sort(L); //排序
87.         return OK;
88.     }
89.
90.     return ERROR;
91. }
92.
93. bool CreateSet(List& L) {
94.     if (InitList(L)) {
95.         return OK;
96.     }
97.
98.     return ERROR;
99. }
100.
101. //销毁集合
102. bool DestroySet(List& L) {
103.     DestroyList(L);
104.     return OK;
105. }
106.
107. double stringToNum(const string& str)
108. {
109.     istringstream iss(str);
110.     double num;
111.     iss >> num;
112.     return num;
113. }
114.
115. bool sort(List& L) {
116.     int flag = 1;
```

```
117.     ElemType temp;
118.     PNode current;
119.
120.     current = L.head->next;
121.     for (int i = 1; i < L.length; i++) {
122.         if (!isNum(current->data)) {
123.             flag = 0;
124.             break;
125.         }
126.         current = current->next;
127.     }
128.
129.     if (flag) { //数字排序
130.         for (int i = L.length; i >= 2; i--) {
131.             current = L.head->next;
132.             for (int j = 1; j < i; j++) {
133.                 if (stringToNum(current->data) > stringToNum(current->next->data)) {
134.                     temp = current->data;
135.                     current->data = current->next->data;
136.                     current->next->data = temp;
137.                 }
138.                 current = current->next;
139.             }
140.         }
141.     }
142.     else { //字符串排序
143.         for (int i = L.length; i >= 2; i--) {
144.             current = L.head->next;
145.             for (int j = 1; j < i; j++) {
146.                 if (current->data > current->next->data) {
147.                     temp = current->data;
148.                     current->data = current->next->data;
149.                     current->next->data = temp;
150.                 }
151.                 current = current->next;
152.             }
153.         }
154.     }
155.     return OK;
156. }
157.
158. bool isNum(string str)
159. {
160.     stringstream sin(str);
161.     double d;
```

```
162.     char c;
163.     if (!(sin >> d))
164.         return false;
165.     if (sin >> c)
166.         return false;
167.     return true;
168. }
169.
170. /* 以下为对线性表的操作 */
171.
172. //删去重复元素
173. bool removeReated(List& L) {
174.     if (!L.head) {
175.         return ERROR;
176.     }
177.
178.     PNode current = L.head->next;
179.
180.     for (int i = 1; i <= L.length; i++) {
181.         PNode temp = current;
182.         for (int j = i + 1; j < L.length; j++) {
183.             if (current->data == temp->next->data) {
184.                 PNode f = temp->next;
185.                 temp->next = f->next;
186.                 delete f;
187.                 L.length--;
188.             }
189.
190.             temp = temp->next;
191.         }
192.
193.         current = current->next;
194.     }
195.
196.     return OK;
197. }
198.
199. //返回线性表长度
200. bool ListLength(List L, int& index) {
201.     if (!L.head) {
202.         return ERROR;
203.     }
204.     return L.length;
205. }
206. bool InitList(List& L) {
```

```
207.     L.head = L.head = new Node;//L.head = (PNode)malloc(sizeof(Node));
208.     if (!L.head) {
209.         return ERROR;
210.     }
211.
212.     L.head->next = NULL;
213.     L.tail = L.head;
214.     L.length = 0;
215.
216.     return OK;
217. }
218. bool DestroyList(List& L) {
219.     if (!L.head) {
220.         return ERROR;
221.     }
222.     PNode current = L.head;
223.     while (current) {
224.         PNode f = current;
225.         current = current->next;
226.         delete f;
227.     }
228.     // PNode p;
229.     // while(L.head != L.tail){
230.     //     p = L.head;
231.     //     L.head = L.head->next;
232.     //     free(p);
233.     // }
234.
235.     // free(L.head);
236.     //printf("销毁成功");
237.     return OK;
238. }
239. bool ClearList(List& L) {
240.     if (!L.head) {
241.         return ERROR;
242.     }
243.     PNode current = L.head;
244.     while (current != L.tail) {
245.         PNode f = current;
246.         current = current->next;
247.         delete f;
248.     }
249.     L.head = L.tail;
250.     L.length = 0;
251.     return OK;
```

```
252. }
253. bool ListEmpty(List& L) {
254.     if (!L.head) {
255.         return TRUE;
256.     }
257.
258.     return FALSE;
259. }
260. bool GetElem(List L, int index, ElemType& e) {
261.     if (!L.head || index < 1 || index > L.length) {
262.         return ERROR;
263.     }
264.     PNode current = L.head;
265.     int i = 0;
266.     while (i != index) {
267.         current = current->next;
268.         i++;
269.     }
270.
271.     e = current->data;
272.
273.     return OK;
274. }
275. bool LocateElem(List L, ElemType e) {
276.     if (!L.head) {
277.         return FALSE;
278.     }
279.
280.     PNode current = L.head->next;
281.     while (current) {
282.         if (current->data == e) {
283.             return TRUE;
284.         }
285.         current = current->next;
286.     }
287.
288.     return FALSE;
289. }
290.
291. bool Append(List& L, ElemType e) {
292.     if (!L.head) {
293.         return ERROR;
294.     }
295.
296.     PNode newNode = new Node;
```

```
297.     newNode->data = e;
298.     newNode->next = NULL;
299.
300.     L.tail->next = newNode;
301.     L.tail = newNode;
302.     L.length++;
303.     return OK;
304. }
305. bool InsertElem(List& L, int index, ElemType e) {
306.     if (!L.head) {
307.         return ERROR;
308.     }
309.
310.     if (index < 1 || index > L.length + 1) {
311.         return ERROR;
312.     }
313.     int i = 1;
314.
315.     PNode current = L.head;
316.     while (i != index) {
317.         current = current->next;
318.         i++;
319.     }
320.
321.     PNode newNode = new Node;
322.
323.     newNode->data = e;
324.     newNode->next = current->next;
325.     current->next = newNode;
326.
327.     if (index == L.length + 1) {
328.         L.tail = newNode;
329.     }
330.
331.     L.length++;
332.     return OK;
333. }
334. bool removeIndex(List& L, int index)
335. {
336.     if (!L.head || L.length == 0) {
337.         return ERROR;
338.     }
339.
340.     if (index < 1 || index > L.length) {
341.         return ERROR;
```

```
342.     }
343.     PNode current = L.head;
344.     int i = 0;
345.     while (++i != index) {
346.         current = current->next;
347.     }
348.
349.     PNode f = current->next;
350.     current->next = f->next;
351.     delete f;
352.     L.length--;
353.
354.     return OK;
355. }
356.
357. bool removeElem(List& L, ElemType e) {
358.     if (!L.head || L.length == 0) {
359.         return ERROR;
360.     }
361.
362.     PNode current = L.head;
363.
364.     while (current) {
365.         current = current->next;
366.
367.         if (current->data == e) {
368.             PNode f = current->next;
369.             current = f;
370.
371.             delete f;
372.             L.length--;
373.             return OK;
374.         }
375.     }
376.
377.     return ERROR;
378.
379. }
380. bool printList(List L) {
381.     if (!L.head) {
382.         return ERROR;
383.     }
384.
385.     std::cout << "{";
386.     if (!L.length) {
```



```
387.         std::cout << "}" << endl;
388.         return OK;
389.     }
390.
391.     PNode current = L.head->next;
392.
393.     for (int i = 1; i < L.length; i++) {
394.         std::cout << current->data << ",";
395.         current = current->next;
396.     }
397.
398.     std::cout << current->data << "}" << endl;
399.     return OK;
400. }
```

main.cpp

```
1. #include <iostream>
2. #include <algorithm>
3.
4. #include "set.h"
5. using namespace std;
6.
7.
8. // 重载集合的输出运算符
9. ostream& operator<<(ostream& output, List& L) {
10.     printList(L);
11.     return output;
12. }
13. void inputSet(List& set);
14. void select();
15.
16.
17. int main() {
18.     printf("\t          集合运算          \n");
19.     printf("\t          使用说明          \n");
20.     select();
21.
22.     while (true) {
23.         Lab:    List set1;
24.                List set2;
25.                List set3;
26.                CreateSet(set3);
27.                cout << "    请输入集合, 输入示例: {1,2,3,4,8}或{\"word\", \"ppt\", \"append\"}或
```

```
{'a','b','c','f'}\n\n";
28.     cout << "请输入集合 A:";
29.     inputSet(set1);
30.     cout << "请输入集合 B:";
31.     inputSet(set2);
32.     cout << "\n 集合输入完成, 请继续操作\n";
33.
34.     //select();
35.     while (true) {
36.         cout << "\n 请选择操作: ";
37.         int ch;
38.         cin >> ch;
39.         switch (ch) {
40.             case 1:
41.                 Intersection(set1, set2, set3);
42.                 cout << set3; ClearList(set3);
43.                 break;//交集
44.             case 2:
45.                 Union(set1, set2, set3);
46.                 cout << set3; ClearList(set3);
47.                 break;//并集
48.             case 3:
49.                 Complement(set1, set2, set3);
50.                 cout << set3; ClearList(set3);
51.                 break;//!A (AUB 为全集)
52.             case 4:
53.                 Complement(set2, set1, set3);
54.                 cout << set3; ClearList(set3);
55.                 break;//!B (AUB 为全集)
56.             case 5:
57.                 Difference(set1, set2, set3);
58.                 cout << set3; ClearList(set3);
59.                 break;//A-B
60.             case 6:
61.                 Difference(set2, set1, set3);
62.                 cout << set3; ClearList(set3);
63.                 break;//B-A
64.             case 7:
65.                 DestroySet(set1);
66.                 DestroySet(set2);
67.                 DestroySet(set3);
68.                 cout << endl;
69.                 goto Lab;//重新输入集合
70.             case 8:
71.                 DestroySet(set1);
```

```
72.         DestroySet(set2);
73.         DestroySet(set3);
74.         return 0;//退出
75.     default:
76.         cout << "操作有误!";
77.         break;
78.     }
79. }
80. }
81.
82. return 0;
83. }
84.
85.
86. void select() {
87.     printf("\t***-----***\n");
88.     printf("\t###   请选择: 1.交集:AnB           2.并集:AUB           ###\n");
89.     printf("\t###           3.补集:!A (AUB 为全集)   4.补集:!B (AUB 为全集)   ###\n");
90.     printf("\t###           5.差集:A-B           6.差集:B-A           ###\n");
91.     printf("\t###           7.重新输入集合           8.退出           ###\n");
92.     printf("\t***-----***\n");
93. }
94.
95. void inputSet(List& set) {
96.     ElemType* deal;
97.     int length;
98.     while (1) {
99.         string str;
100.        cin >> str;
101.
102.        if (str.length() == 2) {
103.            deal = new string[0];
104.            length = 0;
105.            break;
106.        }
107.
108.        int former = str.find("{");
109.        int later = str.find("}");
110.        int count_former = count(str.begin(), str.end(), '{');
111.        int count_later = count(str.begin(), str.end(), '}');
112.        int count_Elem = count(str.begin(), str.end(), ',') + 1;
113.        //检查输入合法性
114.        if (former != 0 || later != str.length() - 1 || count_former != 1 || count_later != 1) {
115.            cout << "输入有误:请重新输入:";
```

```
116.     }
117.     else {
118.         deal = new string[count_Elem];
119.
120.         str[0] = ',';
121.         str[str.length() - 1] = ',';
122.         //提取元素
123.         for (unsigned current = 0, next, i = 0; current < str.length() && i < count_El
            em; i++) {
124.             next = str.find_first_of(',', current + 1);
125.             deal[i] = str.substr(current + 1, next - current - 1);
126.             current = next;
127.         }
128.         length = count_Elem;
129.
130.         break;
131.     }
132. }
133.
134. CreateSet(set, deal, length);
135. delete[] deal; //
136. }
```