第 X 题：集合的交、并和差运算

【问题描述】

编制一个能演示执行集合的交、并和差运算的程序。

【基本要求】

（1）集合的元素限定为小写字母字符['a'…'z']。

（2）演示程序以用户和计算机的对话方式执行。

【测试数据】

（1）输入 magazine 后， Set1 = [a, e, g, i, m, n, z]；

输入 paper 后， Set2 = [a, e, p, r]；

Set1∩Set2 = [a, e]，Set1∪Set2 = [a, e, g, i, m, n, p, r, z]，

Set1－Set2 = [g, i, m, n, z]。

（2）输入 012oper4a6tion89 后， Set1 = [a, e, i, n, o, p, r, t]；

输入 error data 后， Set2 = [a, d, e, o, r, t]；

Set1∩Set2 = [a, e, o, r, t]，Set1∪Set2 = [a, d, e, i, n, o, p, r, t]，

Set1－Set2 = [i, n, p]。

【实现提示】

以有序链表表示集合。

【选作题目】

（1）集合的元素判定和子集判定运算。

（2）求集合的补集。

（3）集合的混合运算表达式求值。

（4）集合的元素类型推广到其他类型，甚至任意类型。

实验报告示例 第 X 题：集合的交、并和差运算

实验报告

题目：编制一个演示集合的交、并和差运算的程序

班级：　　姓名：　　学号：　　完成日期：

一、需求分析

1．本演示程序中，集合的元素限定为小写字母字符['a'…'z']，集合的大小 n<27。集合输入的形式为一个以“回车键”为结束标志的字符串，串中字符顺序不限，且允许出现重复字符或非法字符，程序应能自动滤去。输出的运算结果字符串中将不含重复字符或者非法字符。

2．演示程序以用户和计算机的对话方式执行，即在计算机终端上显示合适的提示信息之后，由用户在键盘上输入演示程序中规定的运算命令；命令执行完后，显示相应的输入数据（滤去输入中的非法字符）和运算结果。

3．程序执行的命令包括：

1）构造集合 1；2）构造集合 2；3）求交集；4）求并集；5）求差集；6）结束。

“构造集合 1”和“构造集合 2”时，需以字符串的形式键入集合元素。

4. 测试数据

（1）输入 magazine 后，　Set1 = [a, e, g, i, m, n, z]；

输入 paper 后，　Set2 = [a, e, p, r]；

Set1∩Set2 = [a, e]，Set1∪Set2 = [a, e, g, i, m, n, p, r, z]，

Set1－Set2 = [g, i, m, n, z]。

（2）输入 012oper4a6tion89 后，　Set1 = [a, e, i, n, o, p, r, t]；

输入 error data 后，　Set2 = [a, d, e, o, r, t]；

Set1∩Set2 = [a, e, o, r, t]，Set1∪Set2 = [a, d, e, i, n, o, p, r, t]，

Set1－Set2 = [i, n, p]。

二、概要设计

集合是指具有某种特定性质的具体的或抽象的对象汇总成的集体，这些对象称为该集合的元素。若 x 是集合 A 的元素，则记作 x∈A。集合中的元素有三个特征：1）确定性（集合中的元素必须是确定的）。 2）互异性（集合中的元素互不相同）。例如：集合 A={1，a}，则 a 不能等于 1。3）无序性（集合中的元素没有先后之分），如集合{3,4,5}和{3,5,4}算作同一个集合。

题目要求实现集合的交、并和差算法，这些操作都需要检查集合中每一元素，为了提高算法的效率，选择以有序链表来存储集合。为此，需要两个定义两个类：有序链表类和集合类，其中有序链表类为集合类的基类。

1.　有序链表类 OrderedList

数据对象：$D = \{a_i | a_i \in CharSet, i=0,1,\ldots,n-1, n\geq 0\}$

数据关系：$R = \{<a_{i-1}, a_i> | a_{i-1}, a_i \in D, a_{i-1}<a_i, i=1,2,\ldots,n-1\}$

基本操作：

OrderedList()

操作结果：构造一个空的有序表。

~OrderedList()

初始条件：有序表已存在。

操作结果：销毁有序表。

length()

初始条件：有序表已存在。

操作结果：返回有序表的长度。

getElemPos(pos)

初始条件：有序表已存在。

操作结果：获得指向第 pos 个数据元素的指针。

locateElem(e,&p)

初始条件：有序表已存在。

操作结果：若有序表中存在元素 e，则 p 指示第一个值为 e 的元素的位置，并返回函数值 true；否则 p 指示第一个大于 e 的元素的前驱的位置，并返回函数值 false。

append(e)

初始条件：有序表已存在。

操作结果：在有序表的末尾插入元素 e。

insertAfter(p,e)

初始条件：有序表已存在，p 指向表中一个元素。

操作结果：在有序表中 p 指示的元素之后插入元素为 e 的结点。

traverse()

初始条件：有序表已存在。

操作结果：依次访问有序表的每个元素。

2. 集合类 Set

数据对象：D = {a_i|a_i 为小写字母且互不相同, i=1,2,…,n, 0<=n<=26}

数据关系：R = { }

基本操作：

Set(Str)

初始条件：Str 为字符串。

操作结果：生成一个由 Str 中小写字母构成的集合。

~Set()

初始条件：集合已存在。

操作结果：销毁集合的结构。

intersectionSets(S1,S2)

初始条件：集合 S1 和 S2 已存在。

操作结果：生成一个由 S1 和 S2 的交集构成的集合。

unionSets(S1,S2)

初始条件：集合 S1 和 S2 已存在。

操作结果：生成一个由 S1 和 S2 的并集构成的集合。

differenceSets(S1,S2)

初始条件：集合 S1 和 S2 已存在。

操作结果：生成一个由 S1 和 S2 的差集构成的集合。

3. 本程序包含四个模块：

1） 主程序模块：

```
int main()
{
    初始化；
    do {
        接受命令；
        处理命令；
    } while(“命令” = “退出”)；
}
```

2） 结点结构单元模块——定义有序表的结点结构。

3） 有序链表单元模块——实现有序链表类。

4） 集合单元模块——实现集合类。

三、详细设计

1. 元素类型、结点类型

```
typedef char elemType;   // 元素类型

struct node {              // 结点类
    elemType data;     // 数据部分
    node *next;          // 指针部分

    node():next(NULL){}     // 结点类的默认构造函数
    node(const elemType &e, node *n = NULL){data = e;next = n;}
};
```

2. 有序链表类

1）有序链表定义

```
// 链表设头、尾两个指针和表长数据域，并设头结点，头结点的数据域没有实际意义。
class OrderedList
{
// 输出运算符重载
friend ostream& operator<<(ostream &os, const OrderedList &list);

private:
    node *head,*tail;     // 分别指向链表的头结点和尾结点
    int size;                  // 链表当前长度，即集合元素个数
protected:
    node *getElemPos(int pos) const;   // 获得指向第 pos 个元素的指针
    bool isValidElem(elemType e);        //判断 e 是否是集合有效元素
public:
    OrderedList();   // 默认构造函数，构造一个只有头结点的空链表
    OrderedList(elemType *s,int sz);   // 构造函数，用 s 所存储的数组元素初始化链表
```

```
    virtual ~OrderedList();       // 析构函数

    // 重置链表元素
    void reset(elemType *s,int sz);

    void remove(int pos);     // 删除第 pos 位置的结点

    int length();   // 返回链表长度

    void clear();   // 清空链表

    //若链表中存在元素 e，则 p 指示第一个值为 e 的元素的位置，并返回函数值 true；
    //否则 p 指示第一个大于 e 的元素的前驱的位置，并返回函数值 false。
    bool locateElem(elemType e, node* &p);

    //在链表中 p 指示的元素之后插入指针 s 所指向的结点。
    //void insertAfter(node* p, node* s);

    //在链表中 p 指示的元素之后插入元素值为 e 的结点。
    void insertAfter(node* p, elemType e);

    // 在链表的末尾插入指针 s 所指向的结点。
    //void append(node *s);

    // 在链表的末尾插入元素值为 e 的结点。。
    void append(elemType e);

    // 遍历算法
    void traverse()const;
};
```

2）有序链表实现

```
// 判断是否是链表的有效元素，如果是返回 true，否则返回 false
// 本演示示例有效元素是小写英文字母
bool OrderedList::isValidElem(elemType e)
{
    if(islower(e)) return true;
    else return false;
};

// 默认构造函数，构造一个只有头结点的空链表
OrderedList::OrderedList()
{
    size = 0;
```

```
    head = tail = new node;
}

// 构造函数，用 s 所存储的数组元素初始化链表
OrderedList::OrderedList(elemType *s,int sz)
{
    size = 0;
    head = tail = new node;   // 产生头节点
    node *p;
    // 生成链表
    for(int i = 0; i < sz; ++i){
        // 过滤重复元素并按字母次序大小插入
        if((isValidElem(s[i]))&&(!locateElem(s[i],p)))
            insertAfter(p,s[i]);
    }
}

// 析构函数
OrderedList::~OrderedList()
{
    clear();
    delete head;
}

// 获得指向第 pos 个元素的指针,pos 有效取值[-1,size-1];
// -1 表示获得头结点;
// 如果 pos 不在有效范围内，则返回 NULL
node* OrderedList::getElemPos(int pos) const
{
    if((pos < -1) || (pos >= size)) // 如果结点为空，则返回 NULL
        return NULL;
    else
        if(pos == -1) return head;   // -1 表示获取头节点
    node* p = head->next;
    for(int j = 0; j < pos; ++j) p = p->next;

    return p;
}

// 删除第 pos 位置的结点,pos 有效取值[0,size-1];
void OrderedList::remove(int pos)
{
    if((pos < 0)|| (pos >= size)) return; // 如果结点为空，不作任何操作
    node* p = getElemPos(pos-1);    // 找到第 pos-1 个结点
```

```
    node *q = p->next;                  // 找到需要删除的结点
    p->next = q->next;                  // 将第 pos-1 个结点的 next 域指向第 pos+1 个结点
    delete q;                           // 删除第 pos 个结点
    --size;                             // 链表长度减 1
    if(size == 0) tail = head; //   空链表
}

// 返回链表长度
int OrderedList::length()
{
    return size;
}

// 清空链表
void OrderedList::clear()
{
    while(length() > 0) remove(0);
}

//若链表中存在元素 e，则 p 指示第一个值为 e 的元素的位置，并返回函数值 true；
//否则 p 指示第一个大于 e 的元素的前驱的位置，并返回函数值 false。
bool OrderedList::locateElem(elemType e, node* &p)
{
    node *pre;
    pre = head;
    p = pre->next;
    while((p) && (p->data < e)) {
        pre = p; p = p->next;
    }
    if((p) && (p->data == e))
        return true;
    else {
        p = pre; return false;
    }
}

//在链表中 p 指示的元素之后插入元素值为 e 的结点。
void OrderedList::insertAfter(node* p, elemType e)
{
    node *s;
    if(p){
        s = new node(e,p->next);
        //s->next = p->next;
        p->next = s;
```

```
            if(tail == p) tail = s;
            ++size;
        }
}

// 在链表的末尾插入元素值为 e 的结点
void OrderedList::append(elemType e)
{
        node *s = new node(e,NULL);
        if(s) {
            if(tail != head) tail->next = s;
            else head->next = s;
            tail = s;
            ++size;
        }
}

// 遍历算法
void OrderedList::traverse()const
{
        node* p = head->next;
        cout << "[";
        while(p) {

            cout << p->data;
            if(p->next) cout <<',';
            p = p->next;
        }
        cout << "]";
}

// 重置链表元素
void OrderedList::reset(elemType *s,int sz)
{
        clear();     // 清除链表中原有的元素

        node *p;
        // 生成链表
        for(int i = 0; i < sz; ++i){
            // 过滤重复元素并按字母次序大小插入
            if((isValidElem(s[i]))&&(!locateElem(s[i],p)))
                insertAfter(p,s[i]);
        }
}
```

```
// 输出运算符重载
ostream& operator<<(ostream &os, const OrderedList &list)
{
	node* p = list.head->next;
	cout << "[";
	while(p) {

		cout << p->data;
		if(p->next) cout <<',';
		p = p->next;
	}
	cout << "]";

	return os;
}
```

3．集合类

集合类继承有序链表类，数据成员不需要增加，增加了交、并和差集三种操作。

1）集合类的定义

```
 class Set:public OrderedList
 {
 public:
    Set();   // 默认构造函数，构造一个只有头结点的空集合
    Set(elemType *s,int sz);   // 构造函数，用 s 所存储的数组元素初始化集合
    virtual ~Set();       // 析构函数

    // 交集运算，求 set1 和 set2 的交集
    void intersectionSets(const Set &set1,const Set &set2);

    // 并集运算，求 set1 和 set2 的并集
    void unionSets(const Set &set1,const Set &set2);

    // 差集运算，求 set1 和 set2 的差集
    void differenceSets(const Set &set1,const Set &set2);
 };
```

2）集合类的实现

```
#include "Set.h"

// 默认构造函数，构造一个只有头结点的空集合
// 直接使用基本的默认构造函数
Set::Set()
{
```

```
}

// 构造函数，用 s 所存储的数组元素初始化集合
Set::Set(elemType *s,int sz):OrderedList(s,sz)
{

}

// 析构函数，不需要作任何工作
Set::~Set()
{

}

// 并集运算，求 set1 和 set2 的并集
void Set::unionSets(const Set &set1,const Set &set2)
{
    node *p1,*p2;
    elemType e1,e2;

    clear();    // 删除集合原有元素
    p1 = set1.getElemPos(0);
    p2 = set2.getElemPos(0);

    while(p1&&p2) {         // 依次比较两个集合中的元素
        e1 = p1->data;
        e2 = p2->data;
        if(e1 <= e2) {    // e1 属于并集中的元素
            append(e1);
            p1 = p1->next;
            if( e1 == e2) p2 = p2->next; // 避免出现重复值
        }
        else {              // e2 属于并集中的元素
            append(e2);
            p2 = p2->next;
        }
    }
    while(p1) {          // set1 剩余元素都属于差集的元素
        e1 = p1->data;
        append(e1);
        p1 = p1->next;
    }
    while(p2) {          // set2 剩余元素都属于差集的元素
        e2 = p2->data;
```

```
            append(e2);
            p2 = p2->next;
        }
}

// 交集运算，求 set1 和 set2 的交集
void Set::intersectionSets(const Set &set1,const Set &set2)
{
        node *p1,*p2;
        elemType e1,e2;

        clear();    // 删除集合原有元素
        p1 = set1.getElemPos(0);
        p2 = set2.getElemPos(0);

        while(p1&&p2) {         // 依次比较两个集合中的元素
            e1 = p1->data;
            e2 = p2->data;
            if(e1 < e2) p1 = p1->next;
            else
                if(e1 > e2) p2 = p2->next;
                else {                  // e1==e2,属于交集的元素
                    append(e1);
                    p1 = p1->next;
                    p2 = p2->next;
                }
        }
}

// 差集运算，求 set1 和 set2 的差集
void Set::differenceSets(const Set &set1,const Set &set2)
{
        node *p1,*p2;
        elemType e1,e2;

        clear();    // 删除集合原有元素
        p1 = set1.getElemPos(0);
        p2 = set2.getElemPos(0);

        while(p1&&p2) {
            e1 = p1->data;
            e2 = p2->data;
            if(e1 < e2) {       // e1 属于差集的元素
                append(e1);
```

```
            p1 = p1->next;
        }
        else
            if(e1 > e2) p2 = p2->next;
            else {    // e1 == e2,不属于差集的元素
                p1 = p1->next;
                p2 = p2->next;
            }
    }
    while(p1) {            // set1 剩余元素都属于差集的元素
        e1 = p1->data;
        append(e1);
        p1 = p1->next;
    }
}
```

4. 主程序模块

主程序包含 main 函数和三个其他函数。

```
// 主函数
int main()
{
    char cmd='\0';

    Set set1;          // 集合 1
    Set set2;          // 集合 2
    Set result;       // 存放集合 1 和集合 2 运算结果

    do {
        display(cmd,set1,set2,result);     // 刷新屏幕显示
        cmd = readCommand();                    // 读取有效命令
        interpret(cmd,set1,set2,result); // 解释执行命令
    }while (cmd !='q');                          // cmd='q'时结束程序执行

    return 0;
}

// 刷新屏幕显示
void display(char cmd,const Set &set1,const Set &set2,const Set &result)
{
    system("cls");    // 清屏
    cout                                                                    <<
"*****************************************************************\n";
    cout << "* MakeSet1-1 MakeSet2-2 Intersection-i Union-u Difference-d Quit-q *\n";
```

```
    cout                                                                    <<
"*************************************************************************\n";
    cout << "\n\n";
    cout << "                Operation:";
    cout << cmd;
    cout << "\n\n\n";
    cout << "                    Set1:" << set1 << "\n\n";
    cout << "                    Set2:" << set2 << "\n\n";
    cout << "                  Result:" << result << "\n\n";
    cout                                                                    <<
"*************************************************************************\n";
}

// 读入操作命令符
char readCommand()
{
    const char commands[] = "12iIuUdDqQ";    // 有效命令集
    char cmd;
    do {
        cout << "* Enter a operation code [1,2,i,u,d OR q] :";
        cmd = cin.get();
        while(cin.get()!='\n');   // 清除多余的输入
    } while(!strchr(commands,cmd));

    return tolower(cmd);
}

// 解释执行命令 cmd
void interpret(char cmd,Set &set1,Set &set2,Set &result)
{
    elemType str[MAXSIZE];

    switch(cmd) {
        case '1':       // 重置 set1
            cout << "Please input a string to construct set1:";
            cin.getline(str,MAXSIZE);
            set1.reset(str,strlen(str));
            break;
        case '2':      //   重置 set2
            cout << "Please input a string to construct set2:";
            cin.getline(str,MAXSIZE);
            set2.reset(str,strlen(str));
            break;
        case 'i':      // 求 set1 和 set2 交集
```

```
                result.intersectionSets(set1,set2);
                break;
            case 'u':     // 求 set1 和 set2 并集
                result.unionSets(set1,set2);
                break;
            case 'd':     // 求 set1 和 set2 差集
                result.differenceSets(set1,set2);
                break;
            default:
                break;
        }
    }
```

四、调试分析

1. 由于对集合的三种运算的算法设计方法不足，在早期版本中对有序链表未设置尾指针和 append 操作，导致算法效率低。

2. 一些函数参数传递类型设计的不恰当，例如：对于并集运算，最初设计时函数原型为 void unionSets(Set set1,Set set2)，函数参数采用的值传递方式，set1 和 set2 作为函数的局部变量，在调用该函数时，需要用到拷贝构造函数进行构造；函数调用结束时，还需用到析构函数。这一方面造成效率低下，另一方面程序中没有专门写拷贝构造函数，使用的是默认拷贝构造函数，造成错误。调试程序时花费了不少时间，后修改为 void unionSets(const Set &set1,const Set &set2)。

3. 算法的复杂度分析

1）时间复杂度

由于采用带头结点的有序单链表结构，并增设了尾指针和表的长度两个标识，各种操作的算法时间复杂度比较合理。

length,insertAfter 和 append 函数的时间复杂度都是 O(1)。

locateElem、traverse、clear、输出运算符重载和析构函数的时间复杂度为 O(n)。

构造 Set 和重新构造 reset 有序集算法需要读入 n 个元素，逐个用 locateElem 判断不在当前集合中及确定插入位置后，才用 insertAfter 插入到有序集中，所有时间复杂度为 $O(n^2)$。

假设 set1 的元素个数为 m，set2 的元素个数为 n，求并集运算利用集合的“有序性”将两个集合的 m+n 个元素不重复地依次比较利用 append 插入到当前并集的尾部，故可在 O(m+n)时间内完成。可对并集和差集的运算作类似分析，它们的时间复杂度也是 O(m+n)。

2）空间复杂度

Set 类中的成员函数实现算法使用的辅助空间与元素个数无关，即是 O(1)。

在主程序中，interpret 函数需要用一个字符串变量读入 n 个元素，需要辅助空间为 O(n)。

五、用户手册

1. 本程序使用的 Code::Blocks 10.05 IDE，程序以项目（project）方式组织，如图 1 所示：

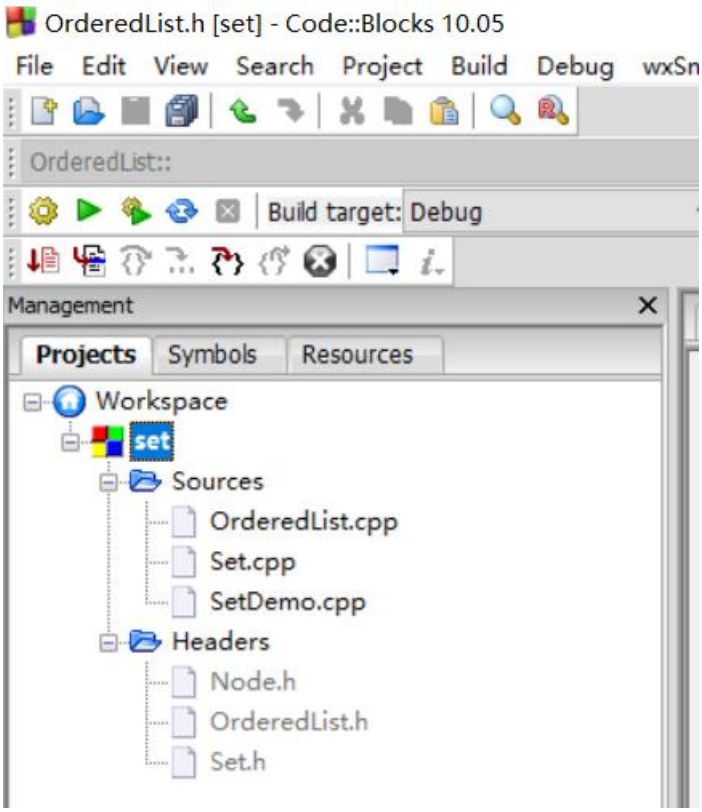

图 1 项目结构

2．依次点击菜单“Build”-> “Build and run”，显示文本方式的用户界面，如图 2 所示：

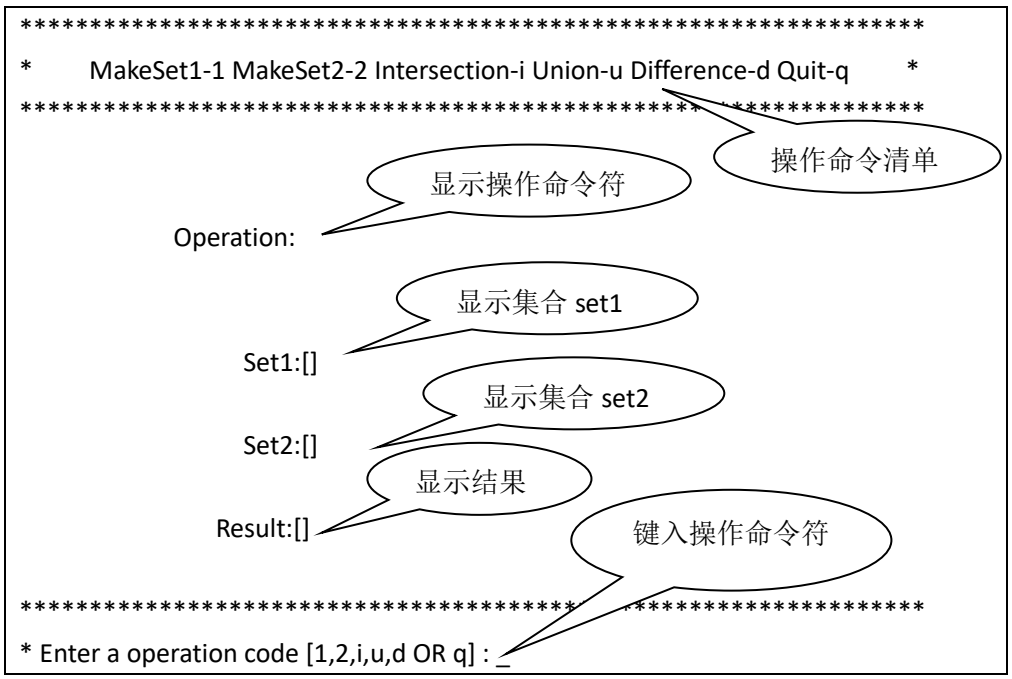

图 2 程序主界面

3．键入操作命令符后，接着输入“回车键”，程序就执行相应的命令。有效的操作命令符为 1、2、i、I、u、U、q 或 Q，不区分大小写。如果输入操作命令符无效，提示并要求重新输入操作命令符。

4．进入“MakeSet1”和“MakeSet2”的命令后，即提示键入集合元素串，结束符为“回车键”。

5．接受其他命令后即执行相应运算并显示相应的结果。

六、测试结果

执行命令 ’1’ ： 键入 magazine 后，构建集合 set1：[a,e,g,i,m,n,z]
执行命令 ’2’ ： 键入 paper 后，构建集合 set2：[a,e,p,r]
执行命令 ’i’ ： 构建集合 set1 和集合 set2 的交集：[a,e]
执行命令 ’u’ ： 构建集合 set1 和集合 set2 的并集：[a,e,g,i,m,n,p,r,z]
执行命令 ’d’ ： 构建集合 set1 和集合 set2 的差集：[g,i,m,n,z]
执行命令 ’1’ ： 键入 012oper4a6tion89 后，构建集合 set1：[a,e,i,n,o,p,r,t]
执行命令 ’2’ ： 键入 error data 后，构建集合 set2：[a,d,e,o,r,t]
执行命令 ’i’ ： 构建集合 set1 和集合 set2 的交集：[a,e,o,r,t]
执行命令 ’u’ ： 构建集合 set1 和集合 set2 的并集：[a,d,e,i,n,o,p,r,t]
执行命令 ’d’ ： 构建集合 set1 和集合 set2 的差集：[i,n,p]

七、附录

源程序文件名清单：

```
Node.h          // 结点结构单元
OrderedList.h   // 有序链表定义单元
OrderedList.cpp // 有序链表实现单元
Set.h           // 集合类定义单元
Set.cpp         // 集合类实现单元
Setdemo.cpp     // 主程序
```