实验二：约瑟夫环问题及集合

班级：计科一班 姓名：元雨暄 学号：1725111037 完成日期：2018.10.7

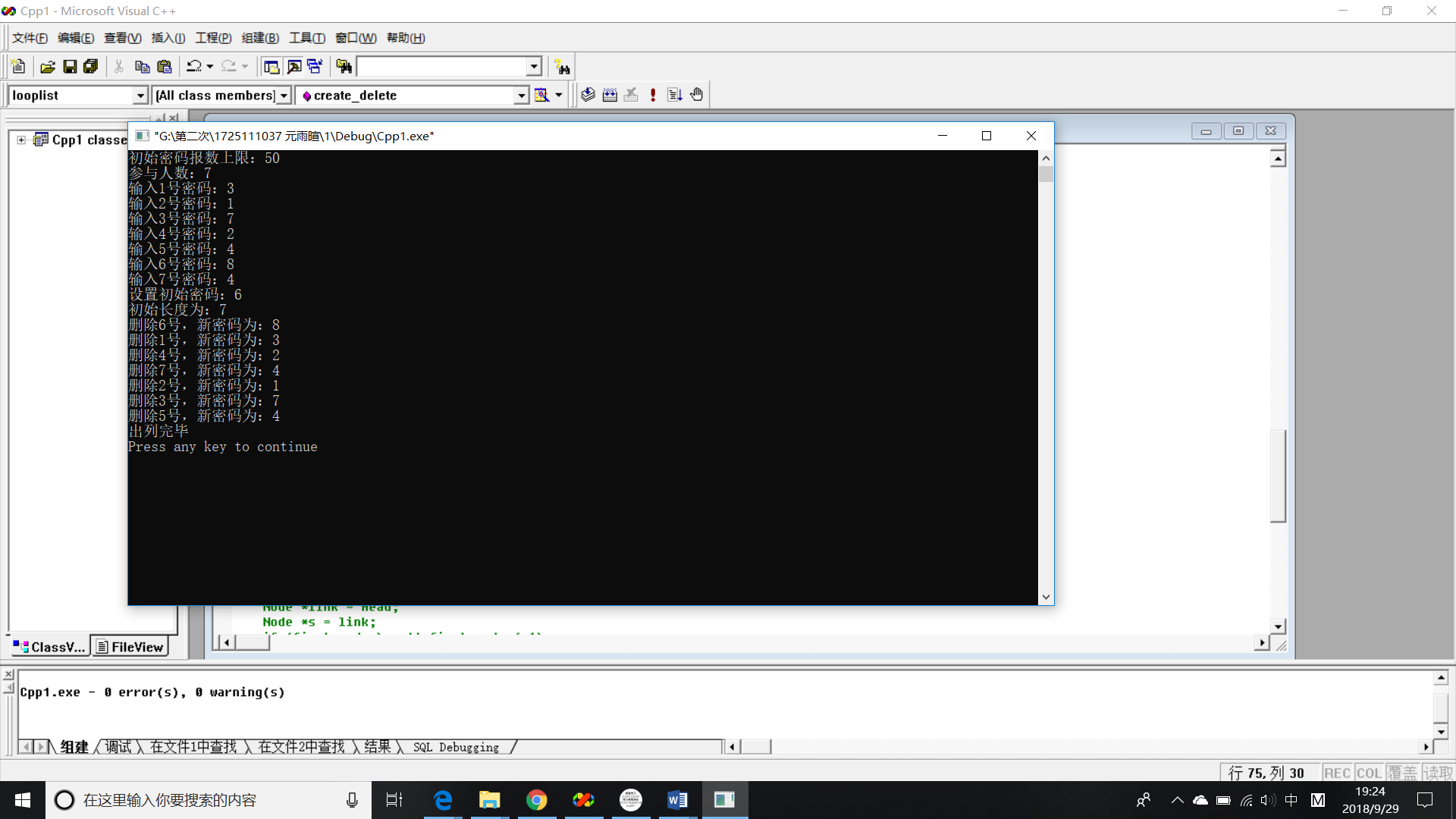
题目一：约瑟夫环

1. 需求分析
2. 用户以键盘输入的方式输入整数，例如初始密码报数上限m、参与人数、每个号次的密码以及初始密码等。初始密码报数上限输入后，初始密码的输入范围变为[1，m]，设置最大参与人数为Max\_num，数值为30，因此参与人数的输入范围变为[1,30]。
3. 当用户输入所需的内容后，按下回车键，程序开始运行，每次显示出出列的序号以及该序号的密码，作为新一轮循环的密码（格式：“删除xx号，新密码为：xx”）。
4. 编号为1,2,3,……n的n个人按顺时针方向围坐一圈，每人持有一个密码（正整数）。一开始任选一个正整数作为报数上限值m,从第一个人开始按顺时针方向自1开始顺序报数，报到m时停止报数。报m的人出列，将他的密码作为新的m的值，从他在顺时针方向上的下一个人开始重新从1报数，如此下去，直至所有的人全部出列为止。

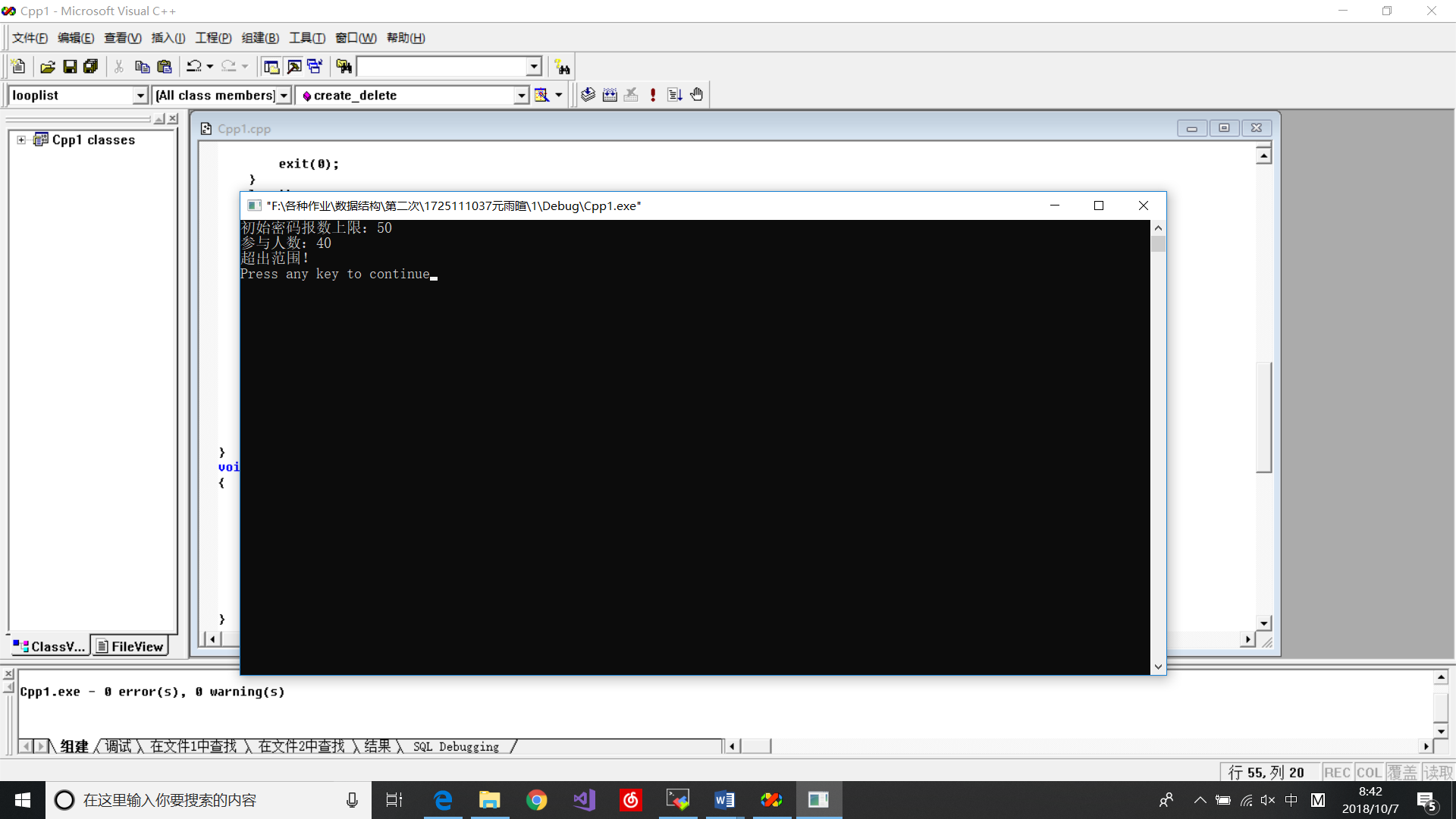
程序所能达到的功能：求出出列顺序。

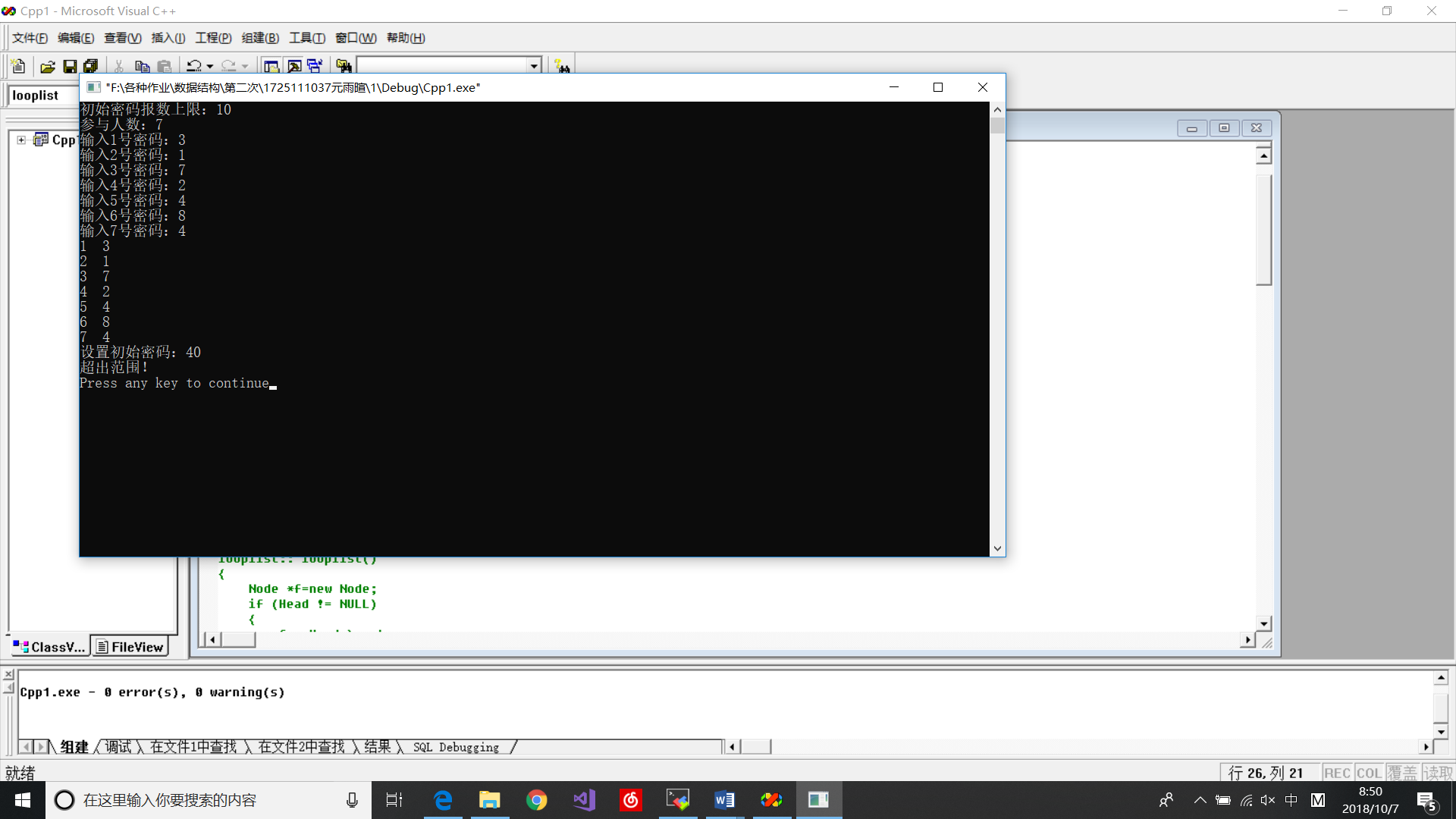
1. 测试数据：包括正确的输入及其输出结果和含有错误的输入及其输出结果

正确的输入以及输出结果如下：



错误的输入以及输出结果如下：





1. 概要设计

1、设定类和结构体的抽象数据类型定义：

struct Node

{

int data; //序号

int code; //密码

Node \*next;

};

class looplist

{

数据对象：D={ai|ai∈CharSet,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1,ai>|ai-1,ai∈D,i=2,…,n}

public：

基本操作：

looplist()

操作结果：构造一个只有一个结点的链表，结点指向自身。

//~looplist(); //析构函数

void create\_list()

操作结果：根据输入的元素以及密码形成一个单向循环链表。

void show\_list()

操作结果：展示链表

void delete\_node()

操作结果：删除特定结点并且输出其序号以及密码

private:

Node \*Head;

int first\_code; //初始密码

int m; //初始密码报数上限

int length; //链表中元素个数

};

2、主程序的流程以及各程序模块之间的层次（调用）关系

定义一个循环链表，调用create\_list()函数形成链表， 调用show\_list()函数展示链表，调用delete\_node()函数进行约瑟夫环式的循环，删除结点。所有序号出列后，结束。

本程序包含两个模块：

1. 主函数模块
2. 循环链表模块

各模块之间的调用关系如下：

主函数模块

循环链表模块

三、 详细设计

1. 结点类型

struct Node

{

int data; //序号

int code; //密码

Node \*next;

};

2、约瑟夫环类型

class looplist

{

public:

looplist();

//~looplist(); //析构函数出错

void create\_list(); //创建相应链表

void show\_list(); //展示链表

void delete\_node(); //删除特定节点并输出相应信息

private:

Node \*Head;

int first\_code; //初始密码

int m; //初始密码报数上限

int length; //链表中元素个数

};

void create\_list();

//按照用户输入的初始密码报数上限确定m的值，用户根据屏幕出现的信息输入相应的内容，创建好一个单项循环链表。若输入人数超出范围则退出程序。

void show\_list();

//对于以及创建好的链表，顺序展示各序号以及密码。

void delete\_node();

//输入初始密码first\_code开始约瑟夫环循环，若初始密码大于初始密码报数上限m则退出，每一次删除相应的结点以及输出新一轮循环的密码，然后进入下一轮循环，直到所有序号都出列。

1. 主程序和其他函数的伪码算法

int main（）

{

//主程序

looplist L; //定义一个循环链表

L.create\_list(); //读入链表长度（即链表元素个数）以及密码

L.show\_list(); //输出链表各元素以及密码

L.delete\_node(); //删除相应的结点

return 0;

}

函数如下：

void looplist::create\_list() //尾插法

{

int n,i;

cout << "初始密码报数上限：";

cin >> m;

cout << "参与人数：";

cin >> n;

if (n > Max\_num || n < 1)

{

cout << "超出范围！" << endl;

exit(0);

}

length = n;

Node \*q,\*s;

cout<<"输入1号密码：";

cin>>Head->code;

Head->data=1;

q=Head;

for(i=1;i<length;i++)

{

s=new Node;

cout<<"输入"<<i+1<<"号密码：";

cin>>s->code;

s->data=i+1;

s->next=q->next;

q->next=s;

q=q->next;

}

}

void looplist::show\_list()

{

Node \*p = new Node;

p = Head;

for (int i = 0; i < length; i++)

{

cout << p->data << " " << p->code << endl;

p = p->next;

}

}

void looplist::delete\_node()

{

int order; //每一轮的密码

cout << "设置初始密码：";

cin >> first\_code;

order = first\_code;

Node \*link = Head;

Node \*s = link;

if (first\_code > m || first\_code < 1)

{

cout << "超出范围！" << endl;

exit(0);

}

cout<<"初始长度为："<<length<<endl;

while (length)

{

for (int i = 1; i < order; i++)

{

s = link;

link = link->next;

}

s->next = link->next;

cout << "删除" << link->data << "号，新密码为：" << link->code << endl;

order=link->code;

link = s->next;

length--;

}

cout<<"出列完毕"<<endl;

}

1. 函数的调用关系图反映了演示程序的层次结构：

主程序

delete\_node

show\_list

create\_list

四、调试分析

(1) 调试过程中经常遇到用new为一个结点开辟空间后总是用delete直接删除导致后续操作出现各种问题例如无法读取内存等。对于一些变量不使用delete删除反而会使程序顺畅。

还有就是设计的析构函数无法使用出现问题，使用默认的析构函数则可以。

(2) 算法的时空分析（包括基本操作和其他算法的时间复杂度和空间复杂度的分析）和 改进设想

时间复杂度：

create\_list() O(length)

show\_list() O(length)

delete\_node() O(length\*order)

空间复杂度：

create\_list() O(length)

show\_list() O(1)

delete\_node() O(2)

(3) 经验和体会等

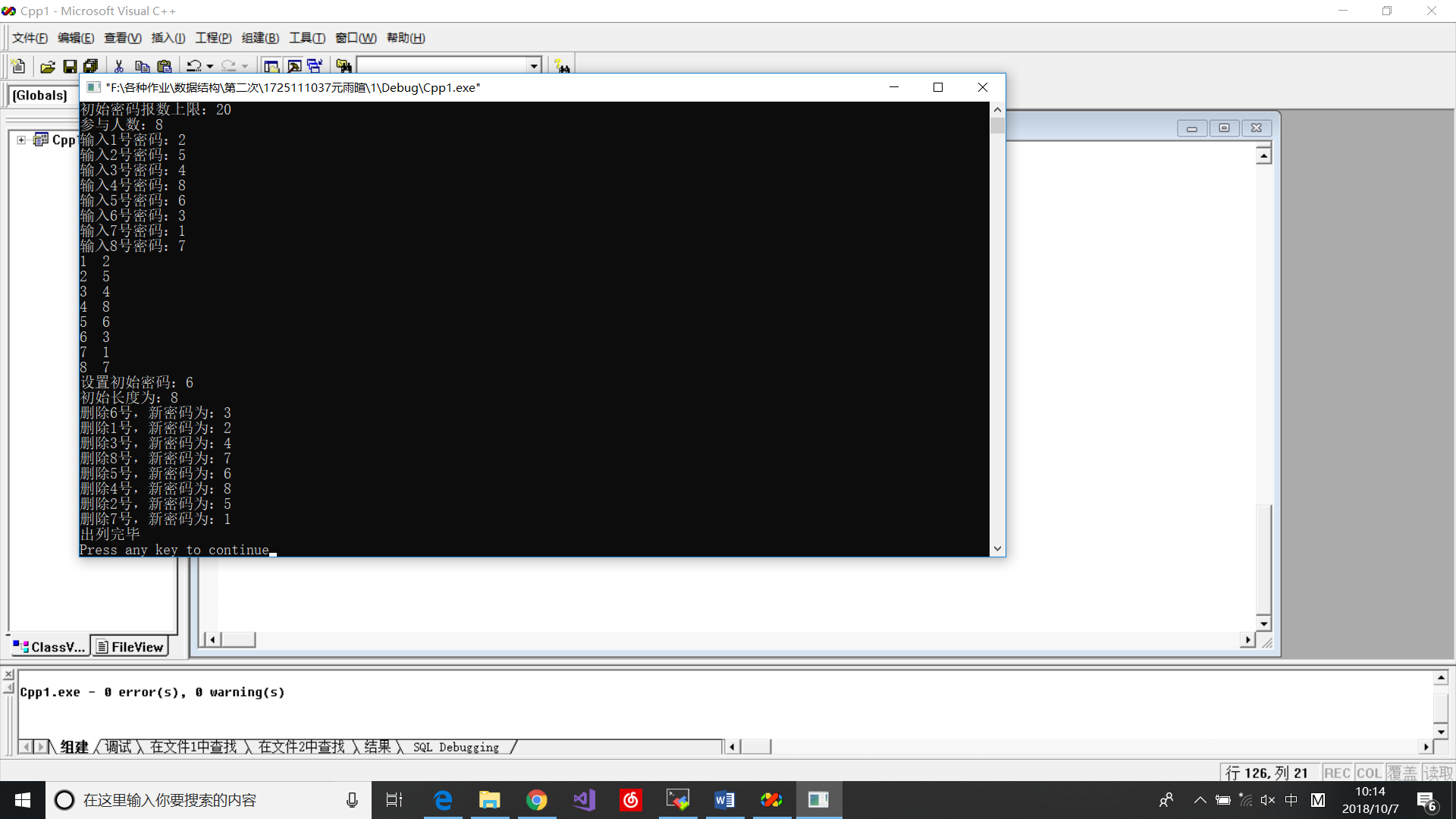
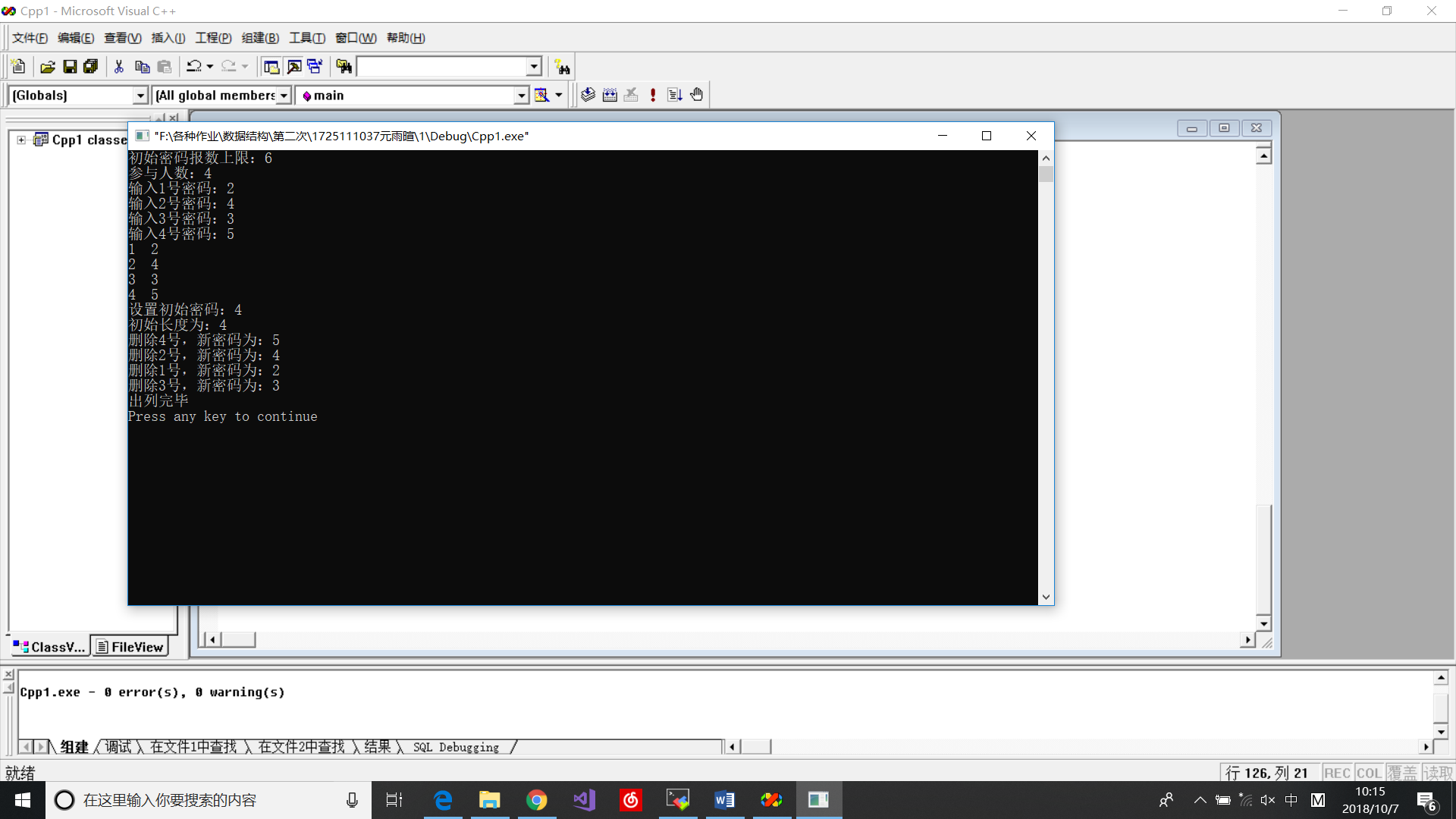
熟悉了循环链表结点创建以及删除的方法，意识对于一个问题的各种考虑是会体现在算法中的，所以考虑问题需要更加全面。

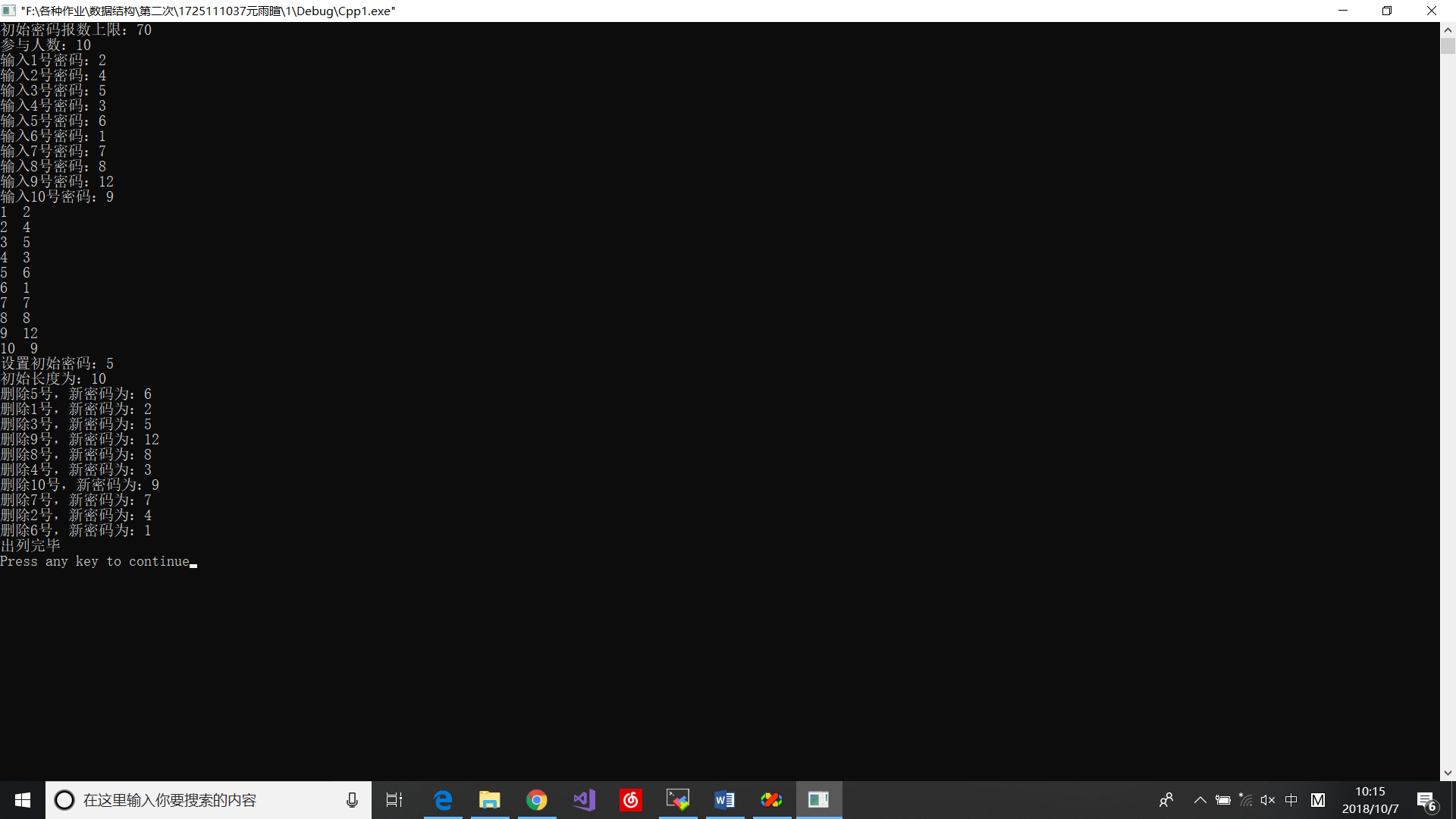
五、用户使用说明

进入调试界面后，根据所给出的提示一步步通过键盘输入即可

首先输入初始密码报数上限m，再输入参与人数，根据所给的人数获得每个序号，然后依次输入每个序号所持有的密码，最后设置初始密码，按下回车键开始进行循环。之后屏幕上会列出出列的序号以及新的密码直到循环结束

六、测试结果



题目二：集合的交、差、并运算

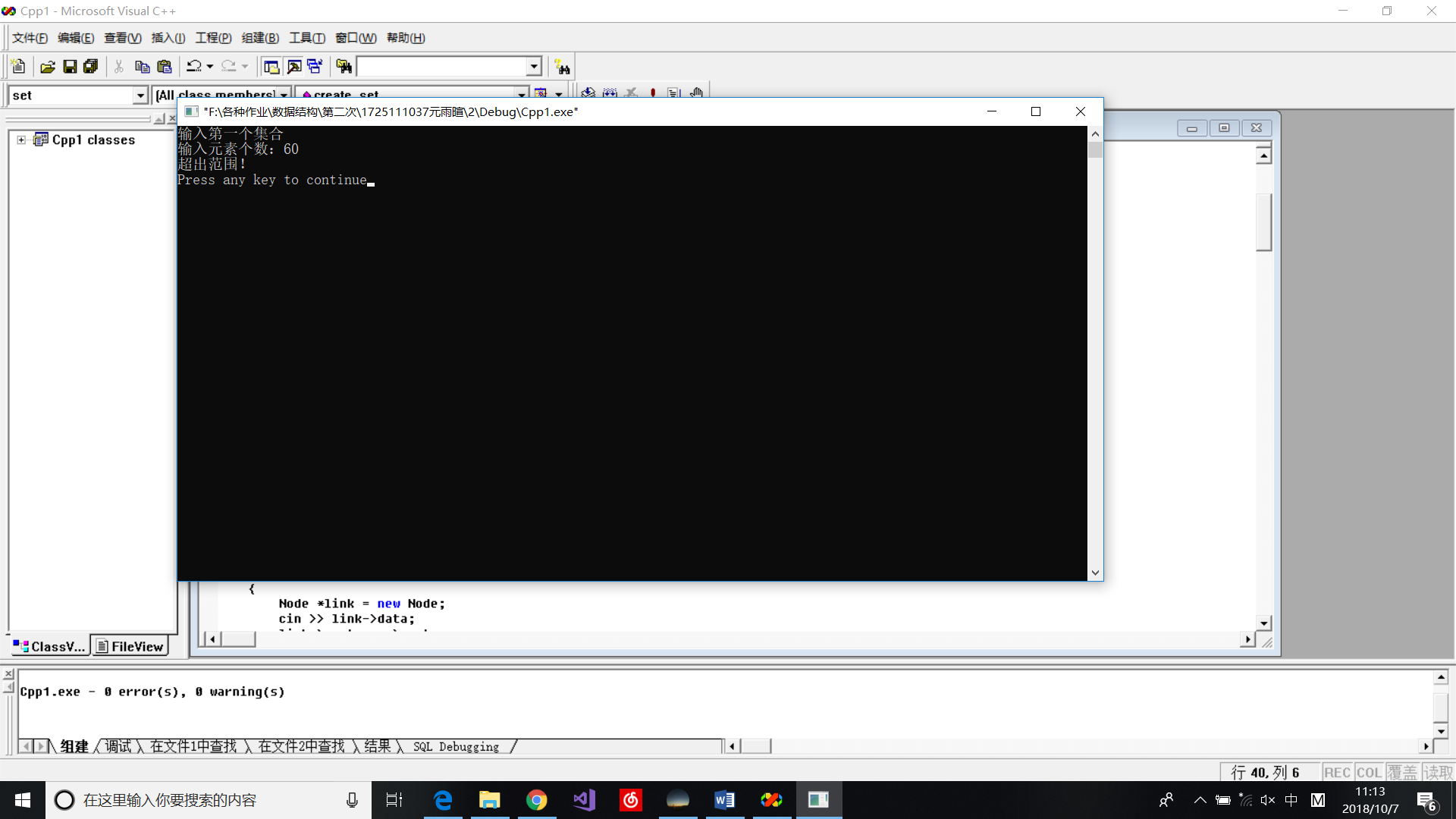
一、需求分析

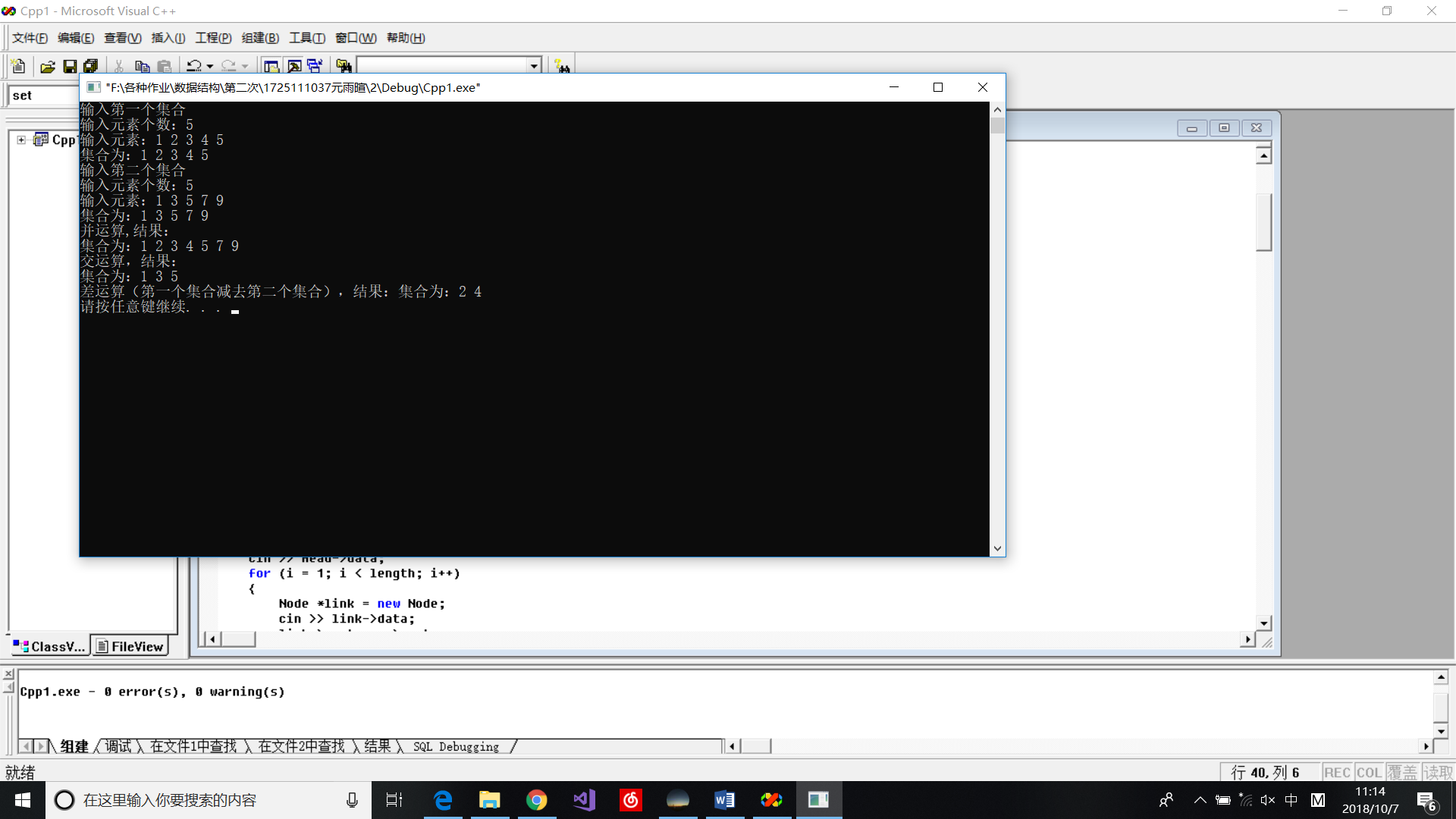
1）从键盘读入集合元素的个数以及每个集合的元素，输入值形式为整数，且输入的集合个数不得超过50个也不能小于1。

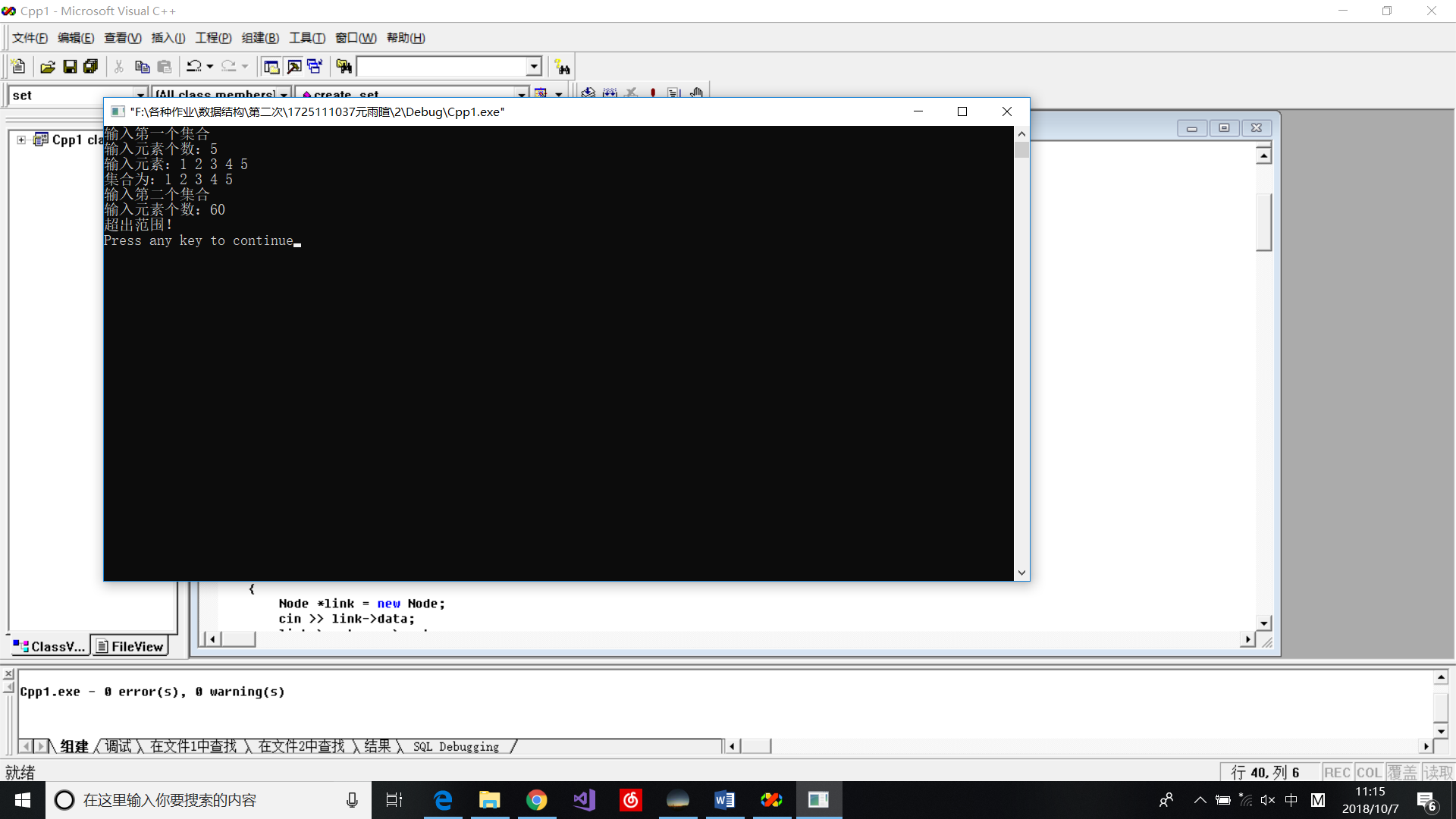
2）以集合形式输出结果。

3）程序能够对两个集合进行交、并、差运算并且显示运算结果。

4）测试数据：







二、概要设计

说明本程序中用到的所有抽象数据类型的定义、主程序的流程以及各程序模块之间的层次（调用）关系。

struct Node

{

Node \*next;

int data;

};

class set

{

数据对象：D={ai|ai∈CharSet,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1,ai>|ai-1,ai∈D,i=2,…,n}

public:

基本操作：

set();

操作结果：构造函数创建一个循环链表；

//~set();

操作结果：析构函数

void create\_set(); //输入创造集合

操作结果：根据用户输入创建集合

void show\_set(); //展示集合

操作结果：展示输入的集合

void data\_deduplication();

操作结果：去除集合中重复的元素

set union\_set(set &s);

操作结果：求集合的并

set intersection(set &s);

操作结果：求集合的交

set difference\_set(set &s);

操作结果：求两个集合的差

private:

Node \*Head;

int length;

};

2、主程序的流程以及各程序模块之间的层次（调用）关系

本程序包含两个模块：

1. 主函数模块
2. 循环链表模块

各模块之间的调用关系如下：

主函数模块

循环链表模块

1. 详细设计

1、结点类型

struct Node

{

Node \*next;

int data;

};

2、集合类型

class set

{

数据对象：D={ai|ai∈CharSet,i=1,2,…,n,n>=0}

数据关系：R1={<ai-1,ai>|ai-1,ai∈D,i=2,…,n}

public:

基本操作：

set();

操作结果：构造函数创建一个循环链表；

//~set();

操作结果：析构函数

void create\_set(); //输入创造集合

操作结果：根据用户输入创建集合

void show\_set(); //展示集合

操作结果：展示输入的集合

void data\_deduplication();

操作结果：去除集合中重复的元素

set union\_set(set &s);

操作结果：求集合的并

set intersection(set &s);

操作结果：求集合的交

set difference\_set(set &s);

操作结果：求两个集合的差

private:

Node \*Head;

int length;

};

1. 主程序和其他函数的伪码算法

int main()

{

set s1, s2, s3,s4,s5;

cout << "输入第一个集合" << endl;

s1.create\_set();

s1.show\_set();

cout << "输入第二个集合" << endl;

s2.create\_set();

s2.show\_set();

cout << "并运算,结果："<<endl;

s3=s1.union\_set(s2);

s3.data\_deduplication();

s3.show\_set();

cout << "交运算，结果："<<endl;

s4 = s1.intersection(s2);

s4.data\_deduplication();

s4.show\_set();

cout << "差运算（第一个集合减去第二个集合），结果：";

s5 = s1.difference\_set(s4);

s5.show\_set();

cout << "差运算（第二个集合减去第一个集合），结果：";

s5 = s2.difference\_set(s1);

s5.show\_set();

return 0;

}

#include<iostream>

using namespace std;

#define max\_num 50

struct Node

{

Node \*next;

int data;

};

class set

{

public:

set();

//~set();

set set\_copy();

void create\_set(); //输入创造集合

void show\_set(); //展示集合

void data\_deduplication(); //去除集合中重复的元素

set union\_set(set &s); //求集合的并

set intersection(set &s); //求集合的交

set difference\_set(set &s); //求两个集合的差

private:

Node \*Head;

int length;

};

set::set()

{

Head = new Node;

Head->data = NULL;

Head->next = Head;

length = 1;

}

set set::set\_copy()

{

int i;

set set\_link;

set\_link.length = length;

Node \*node\_2\_Head = new Node;

node\_2\_Head->next = set\_link.Head;

set\_link.Head->data = Head->data;

Head = Head->next;

for (i = 1; i < length; i++)

{

Node \*k = new Node;

k->data = Head->data;

k->next = set\_link.Head->next;

set\_link.Head->next = k;

set\_link.Head = k;

Head = Head->next;

}

set\_link.Head = node\_2\_Head->next;

return set\_link;

}

void set::create\_set()

{

int i;

cout << "输入元素个数：";

cin >> length;

if(length > max\_num || length<1)

{

cout<<"超出范围！"<<endl;

exit(0);

}

Node \*p = new Node;

p = Head;

cout << "输入元素：";

cin >> Head->data;

for (i = 1; i < length; i++)

{

Node \*link = new Node;

cin >> link->data;

link->next = p->next;

p->next = link;

p = p->next;

}

}

void set::show\_set()

{

int i;

Node \*q = Head;

cout << "集合为：";

for (i = 0; i < length; i++)

{

cout << q->data << " ";

q = q->next;

}

cout << endl;

}

void set::data\_deduplication() //去掉重复的元素

{

int i,j,count;

count = length;

Node \*s1 = new Node;

Node \*s2 = new Node;

s1 = Head;

Node \*s1\_link = new Node;

s1\_link->next = Head;

s2 = s1;

for (i = 0; i < length; i++)

{

for (j = 0; j < length; j++)

{

if (s2->next->data == s1->data)

{

if (s2->next != s1)

{

Node \*d = new Node;

d = s2->next;

s2->next = d->next;

s2 = s2->next;

length--;

break;

}

else

{

break;

}

/\*

else if(s2->next->next==s1\_link->next)

{

Node \*d2 = new Node;

d2 = s2->next;

s2->next = d2->next;

s2 = s2->next;

length--;

}

\*/

}

else

{

s2 = s2->next;

}

}

s1 = s1->next;

s2 = s1;

}

}

set set::union\_set(set &s) //并

{

set s3;

int i;

s3.length = length + s.length;

Node \*m = Head;

Node \*h = new Node; //h是原链表的“第一个”元素

h->data = NULL;

h->next = Head;

for (i = 0; i < length - 1; i++)

{

m = m->next;

} //此时m变成原链表的“最后”一个元素

Node \*n = new Node;

n->data = NULL;

n->next = s.Head;

m->next = s.Head;

for (i = 0; i < s.length; i++)

{

m = m->next; //m变成链表s的最后一个元素

}

Node \*n2 = new Node;

n2->data = NULL;

n2->next = m;

m->next = h->next;

s3.Head=h->next;

m=m->next;

s.Head = n->next;

n2->next->next = s.Head;

return s3;

}

set set::intersection(set &s) //交，建立第三个集合， 把有相同的元素放到第三个集合中；

{

int i, j;

set L;

Node \*L1, \*L2, \*L3;

Node \*temp = new Node;

Node \*t\_L2 = new Node;

L1 = Head;

L2 = s.Head;

L3 = L.Head;

temp->next = L3; //temp->next指向L3 保留L3

temp->data = NULL;

t\_L2->data = NULL; //t\_L2指向L2 保留L2

t\_L2->next = L2;

for (i = 0; i < length; i++)

{

for (j = 0; j < s.length; j++)

{

if (L2->data != L1->data)L2 = L2->next;

else

{

if (L3->data == NULL)

{

L3->data = L2->data;

break;

}

else

{

L.length++;

Node \*L\_link = new Node;

L\_link->data = L2->data;

L\_link->next = L3->next;

L3->next= L\_link;

L3 = L3->next;

break;

}

}

}

L1 = L1->next;

L2 = t\_L2->next;

}

L3 = temp->next;

return L;

}

set set::difference\_set(set &s) //求差，set本身减去set s；求差的函数可以用一个集合减去之前求得的交集，A-(A交B) s就是交集，那么只需要删除A中有的s的元素即可

{

int i, j,count; //count用来记录有没有遍历整个交集

set L;

Node \*q1, \*q2, \*q3;

q1 = Head;

q2 = s.Head;

q3 = L.Head;

Node \*q2\_link = new Node; //用来保存结点q2

Node \*q3\_link = new Node; //用来保存结点q3

Node \*q\_last = new Node;

for (i = 0; i < s.length-1; i++)

{

q2 = q2->next;

}

q\_last->next = q2;

q2 = q2->next;

q3\_link->data = NULL;

q3\_link->next = q3;

q2\_link->data = NULL;

q2\_link->next = q2;

L.length = length - s.length;

for (i = 0; i < length; i++)

{

for (j = 0; j < s.length; j++)

{

if (q1->data == q2->data)

{

break;

}

else

{

if (q2->data == q\_last->next->data)

{

if (q3->data == NULL)

{

q3->data = q1->data;

}

else

{

Node \*n = new Node;

n->data = q1->data;

n->next = q3->next;

q3->next = n;

q3 = q3->next;

}

}

q2 = q2->next;

}

}

q1 = q1->next;

q2 = q2\_link->next;

}

q3=q3\_link->next;

return L;

}

1. 函数的调用关系图反映了演示程序的层次结构：

主程序

第一个集合

create\_set();

第二个集合

create\_set();

difference\_set()

union\_set()

data\_deduplication();

intersection();

data\_deduplication();

Show\_set()

Show\_set()

四、调试分析

(1)有了第一题的基础之后做这一题难度便降低了，重要的是对优化算法的思考。

(2) 算法的时空分析（包括基本操作和其他算法的时间复杂度和空间复杂度的分析）和 改进设想

时间复杂度：

create\_set() O(length)

show\_set() O(length)

union\_set() O(length)

data\_deduplication() O(length\*length)

intersection() O(s1.length\*s2.length)

difference\_set() O(s1.length\*s3.length) (s3为s1与s2的交集)

空间复杂度：

create\_set() O(length)

show\_set() O(1)

union\_set() O(1)

data\_deduplication() O(length)

intersection() O(length)

difference\_set() O(s1.length-s3.length) (s3为s1与s2的交集)

五、用户使用说明

进入调试界面后，根据所给出的提示一步步通过键盘输入即可

首先输入一个集合元素个数，再输入集合各元素，再同样方法输入第二个集合，按下回车键开始进行集合的并、交、差运算。之后屏幕上会列出集合的并、交、差运算结果

六、测试结果

