作业回顾：非编程题

1.想去的地方

拉萨，西安，武汉，南京，乌鲁木齐，各大省会城市...
都江堰，九寨沟，承德避暑山庄，漠河看极光...
欧洲，伦敦，纽约...
珠峰...
哪也不想去（就想在家写代码？）...

2.木有了...

作业回顾：问题总结

- **写代码的问题**
  - 不要使用goto，容易出bug
  - 不管问度娘还是什么，程序一定要亲手打一遍（语法最终都是**记在手上**而不是脑子里，这叫**肢体记忆**）
  - 注意代码的整体结构（参见下页）
  - 不经意的错误（俗称手滑了）
    - 变量类型写错了（比如实数写成int）
    - 忘记include必要的模块
    - =和==写反了
    - 拼写错误（如调用函数的时候函数名写错了）
    - 分号漏写了
    - 分号多写了
    - 函数声明没写返回类型
    - 数组越界
- 如果是竞赛里，不要做题目没要求的事

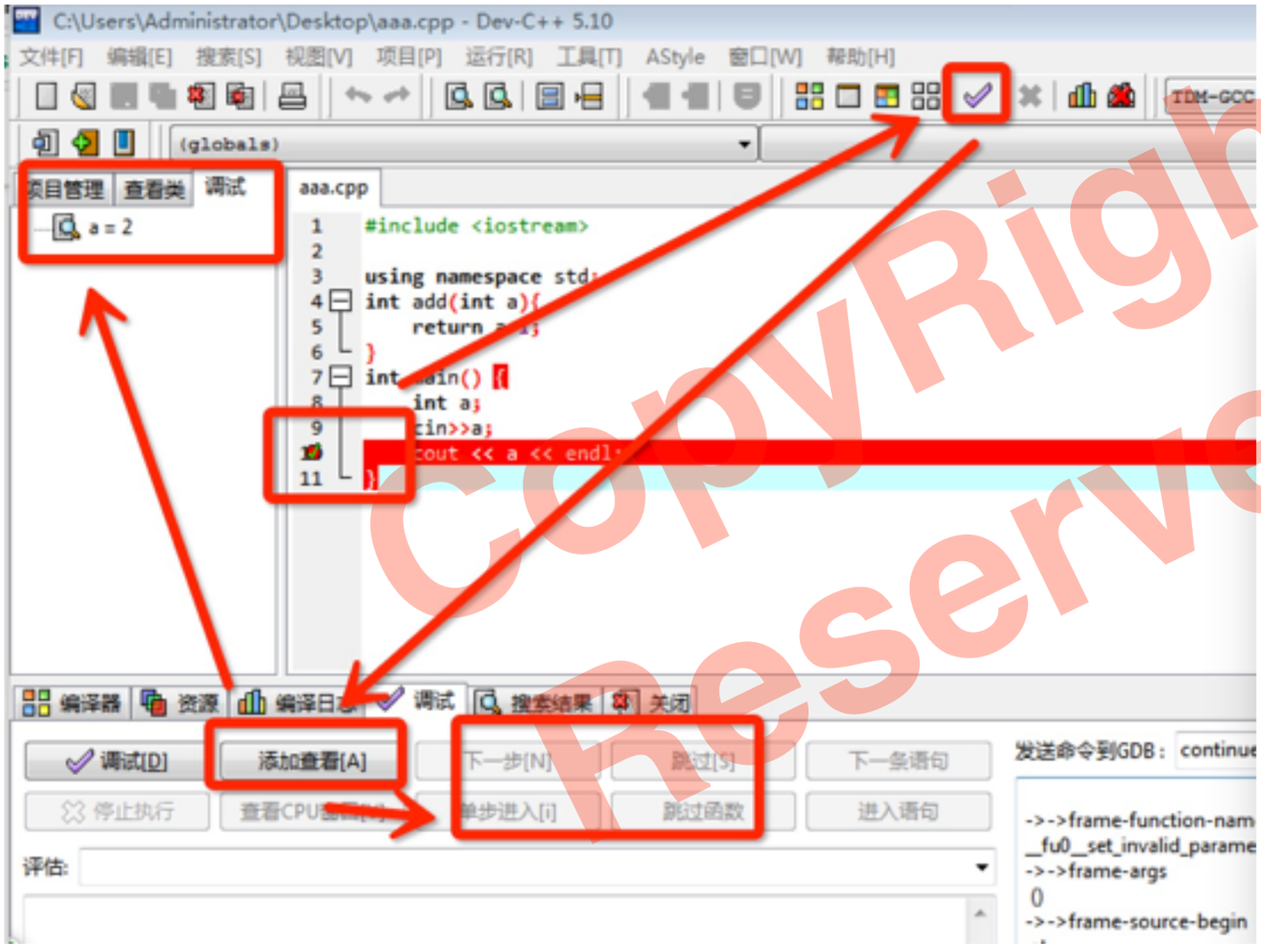
C++程序结构

```
#include….
using…

class 类 {
    变量声明...
    void 函数声明(){……}
    int 函数声明(参数…){……}
};

void 函数声明(){
    语句（声明变量/顺序/循环/分支/调用函数）;
    语句;
    ……
}

int 函数声明(参数…){
}

int main函数(){
    语句（声明变量/顺序/循环/分支/调用函数）;
    语句;
}
```

- → 从外头看程序主要就是类和函数组成的
- → 函数声明和变量声明的区别：后面带不带括号
- → 允许的嵌套：类套函数，函数套语句，语句套语句（if,while等复合语句）
- → 不允许的嵌套：函数套函数，函数套类，语句套函数
- → 编译不对的话，先Check一下程序结构是不是有问题

```
9   #include <iostream>
10  using namespace std;
11
12  void pointerDemo(){ ••• }
32
33  class Point { ••• };
43
44  void pointerWithClassDemo(){ ••• }
64
65  int main(int argc, const char * argv[]) { ••• }
70
```

DevC++调试步骤



- **编译通过结果不对怎么办**
  - 首先庆祝一下编译通过
  - 然后就是调试

1.设置断点（在你想要的地方）
2.启动调试
3.添加查看（你想要看的变量）
4.单步运行（一步步运行程序，观察变量值的变化）

作业回顾：编程题

2.inversepairwithdup.cpp

3.cubicequation.cpp（调试）

4.squreclass.cpp

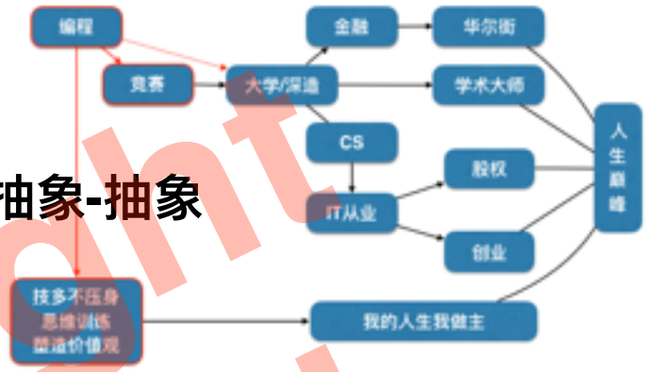
Welcome to Xcode

CS100 算法入门

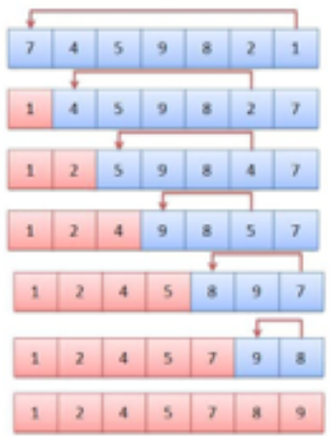
指针

先讲一点哲学：映射

抽象->具体



抽象-抽象

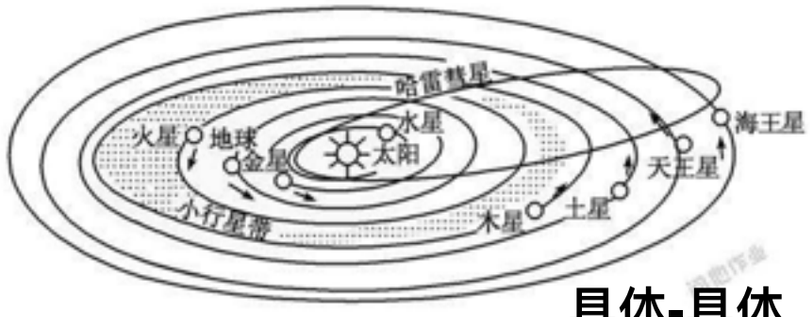


具体-抽象

很多事物都是对另一种事物的刻画/描述
这种关联关系称为映射
所有的事物通过各种映射互相关联/影响

完整信息

部分信息



具体-具体

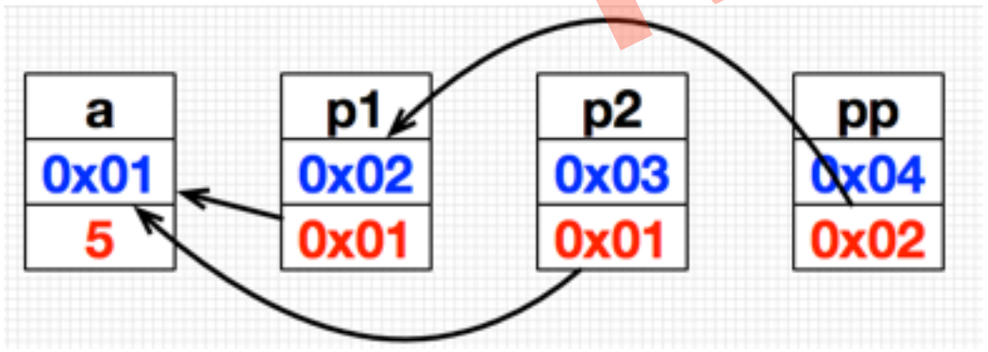
史上最牛的广告：指针

又是广告，有点崩溃?
这是最后一次讲新语法了

张衡路666号

C++技术牛不牛🐂
就看指针溜不溜

- → 所有变量都存储在内存中
- → 指针是一类特殊变量，它的值是另一个变量的内存地址
- → 通常说一个变量指向另一个变量，所以称为指针



怎样形象地理解指针

为了理解什么是指针，网上充满了形形色色的比喻，可见指针的重要性

我的理解

指针就像提线木偶的线

本身没有逻辑含义，却控制着其他变量

好好的为什么蛋疼要去用什么的指针？

→ 有时候程序需要动态申请内存空间，就需要用指针

→ 有了指针可以很自由地实现各种有趣的数据结构（后面会看到）

→ ……

指针运算（基本类型）

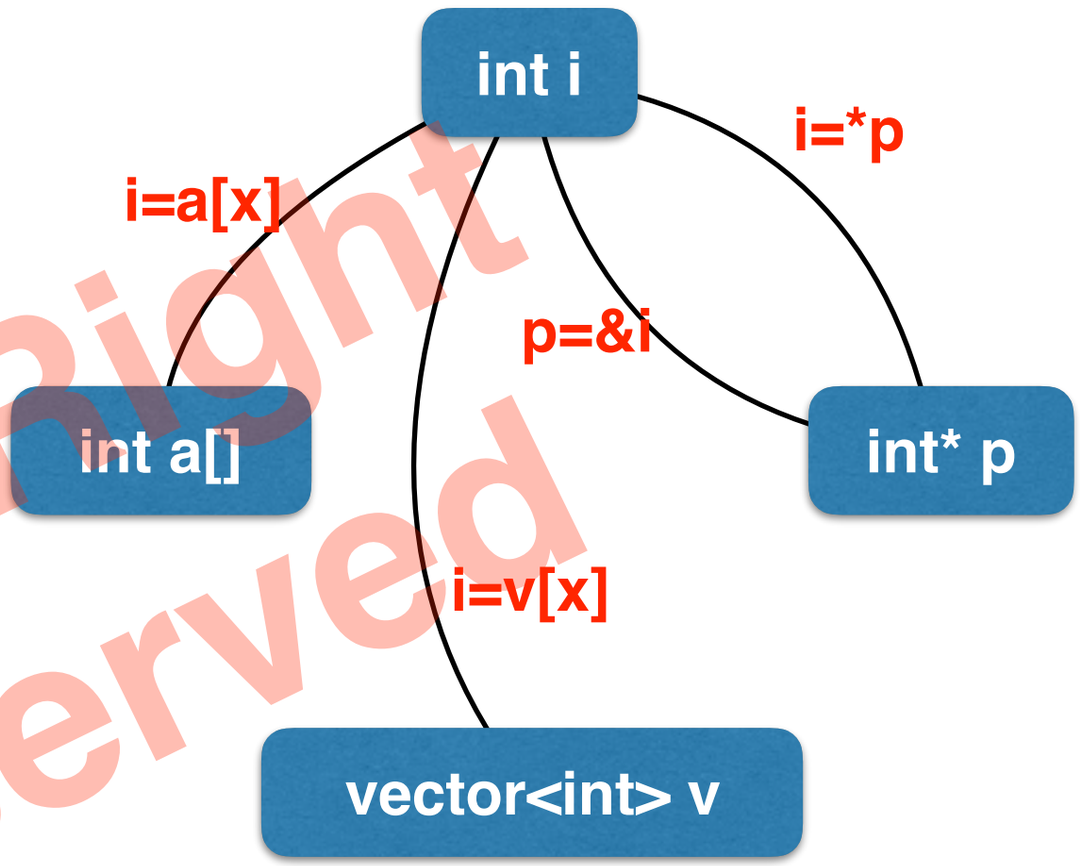
→ 怎样定义指针变量：

普通类型后加*，就表示指针类型

→ 指针与普通类型怎么转换

&：普通变量到指针

*：指针到普通变量



int	整数类型
int[]	整数数组类型
vector<int>	元素为整数的vector类型
int*	指向整数的指针类型

容易混淆的地方（设计确实有点蛋疼...）

&& 逻辑与算符

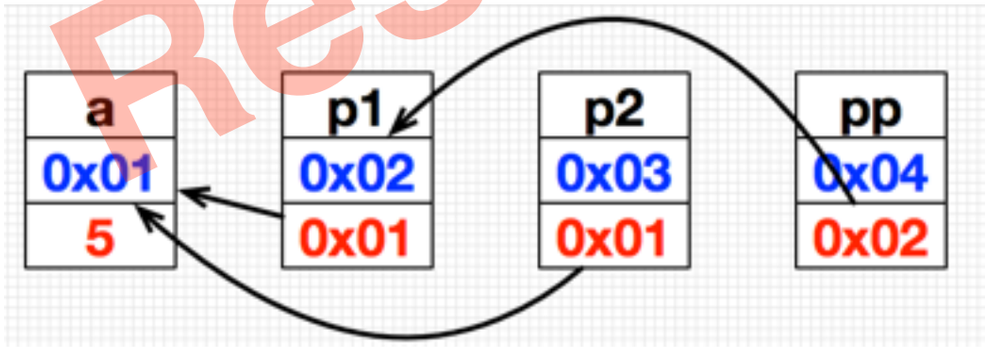
& 取地址算符

int* p; 声明一个变量p，**类型是整数的指针**

int i=*p; 声明一个变量i，类型是整数，值为**指针p指向的值**

在C++使用指针

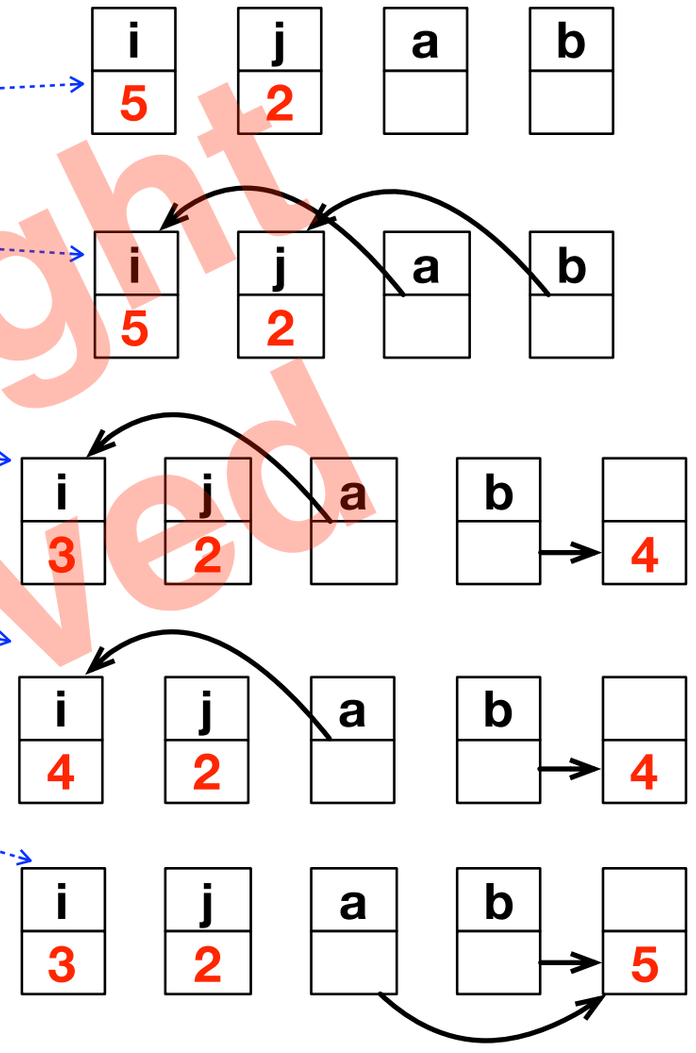
变量名	a	p1	p2	pp
地址	0x01	0x02	0x03	0x04
值	5	0x01	0x01	0x02
类型	int	int*	int*	int**
取地址运算	&a=0x01	&p1=0x02	&p2=0x03	&pp=0x04
取值运算	/	*p1=5	*p2=5	*pp=0x01



指针运算样例

```
187 void pointerDemo(){
188     int i=1,j=2;
189     int* a=0, *b=0;   // 空指针，不是数值0
190     a= &i;   // & : 取地址运算符，返回指针类型
191     b= &j;
192     cout<<"a="<<a<<" b="<<b<<endl;
193     cout<<"*a="<<*a<<" *b="<<*b<<endl; // * : 取值运算符
194
195     b=new int(4);    // 动态申请变量，返回对应指针类型
196     *a=3;    // 修改指向的值
197     std::cout<<"*a="<<*a<<" *b="<<*b<<endl;
198     *a=*b;   // 两个地址具有同样的数值
199     cout<<"a="<<a<<" b="<<b<<endl;
200     cout<<"*a="<<*a<<" *b="<<*b<<endl;
201
202     a=b;     // 两个指针指向同一内存地址
203     *a=5;
204     cout<<"a="<<a<<" b="<<b<<endl;
205     cout<<"*a="<<*a<<" *b="<<*b<<endl;
206 }
207
```

```
a=0x7fff5fbff65c b=0x7fff5fbff658
*a=1 *b=2
*a=3 *b=4
a=0x7fff5fbff65c b=0x100406cc0
*a=4 *b=4
a=0x100406cc0 b=0x100406cc0
*a=5 *b=5
```



链表：指针+类

报数问题

n个人围成一圈，编号依次为1..n（n<=1000000），从第一个人开始1-2报数，报到2的出列。输入n，输出最后剩下的一个人的编号

样例输入：

13

样例输出：

11

(2 4 6 8 10 12 1 5 9 13 7 11 13)

报数问题：使用数组

设一个数组v[n]，初始化为0（表示未出列）
设置一个当前位置current
for (i从1到n-1) {
    current循环+1直到v[current]等于0
    current循环+1（报数1）
    current循环+1直到v[current]等于0
    v[current]=1（报数2，出列）
}

n也不是特别大
直接弄个数组模拟也是可以的

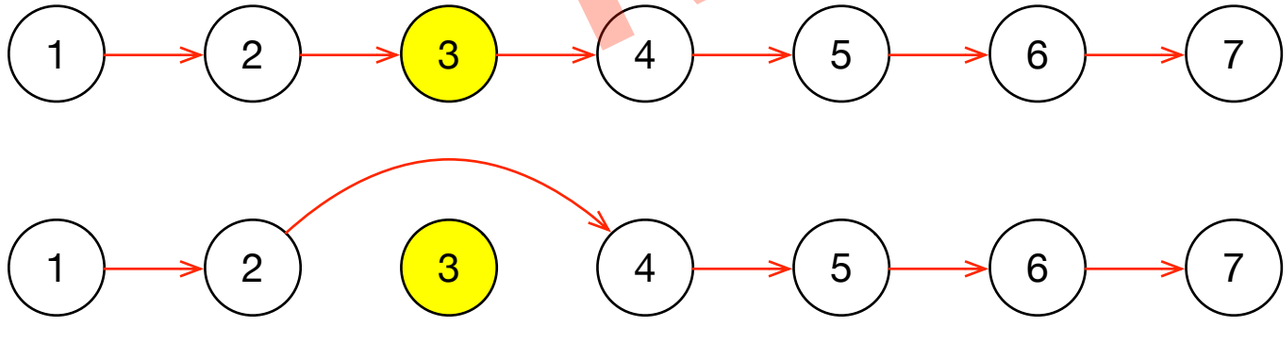
→ **复杂度是多少?**
每报一圈都要循环n次，并且人数减半
总共要报logn圈，所以复杂度就是O(nlogn)

```cpp
int countByArray(int n){
    vector<int> v=vector<int>(n); // 1表示已经出列，初始都是0
    int current=0;
    for (int i=1;i<=n-1;i++){
        for (;v[current]==1;) { // 找下一个0
            current=(current+1)%n;
        }
        current=(current+1)%n;  // 报数1，留在队列中
        for (;v[current]==1;) { // 再找下一个0
            current=(current+1)%n;
        }
        v[current]=1;   // 报数2，出列
        current=(current+1)%n;
    }
    for (;v[current];) {
        current=(current+1)%n;
    }
    return current+1;
}
```

报数问题：使用链表

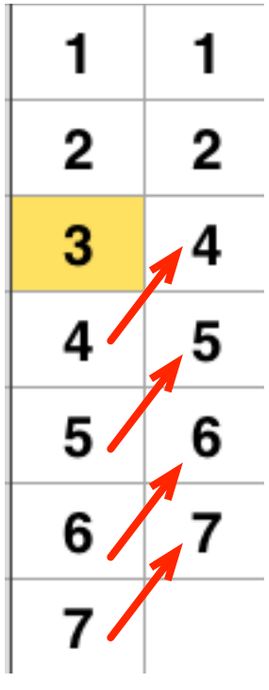
设一个数组v[n]，初始化为0（表示未出列）
设置一个当前位置current
for (i从1到n-1) {
　　current循环+1直到v[current]等于0
　　current循环+1（报数1）
　　current循环+1直到v[current]等于0
　　v[current]=1（报数2，出列）
}

→ 链表是没有下标，只记录后续值的结构，可以用O(1)时间删除和新增元素



主要浪费的时间是已经出列的还在被反复的循环
所以希望出列的就剔除掉

但是在数组里删一个元素（并且保持顺序）是O(n)的



报数问题：解答

```cpp
class Node {     // 链表的节点类
public:
    int id; // 编号
    Node* next=0;    // 指向下一个节点的指针
    Node(int id0){   // 初始化函数
        id=id0;
    }
};

// 使链表解法
int countByLink(int n){
    Node* head=new Node(1);
    Node* tail=head;

    for (int i=2;i<=n;i++){  // 创建链表
        tail->next=new Node(i);
        tail=tail->next;
    }
    tail->next=head;     // 连接成环

    Node* current=head; // 开始报数（当前节点报1）
    for (int i=1;i<=n-1;i++){
        current->next=current->next->next;  // 删除下一个节点（报2）
        current=current->next;  // 挪到下一个节点（报1，注意报2的已经删掉了）
    }
    return current->id;
}
```

又一种新符号，真的崩溃啦！！！
安啦，这是最后一个新运算符了

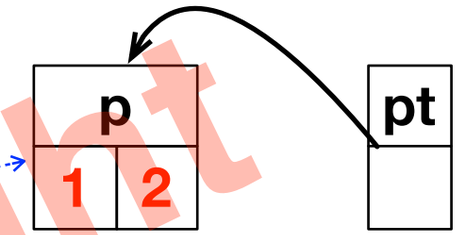
→ new是什么鬼？这是一种直接创建指针变量的写法
特别适合于自定义的类型
比如new Node(1)表示创建一个指针变量，指向一个Node变量

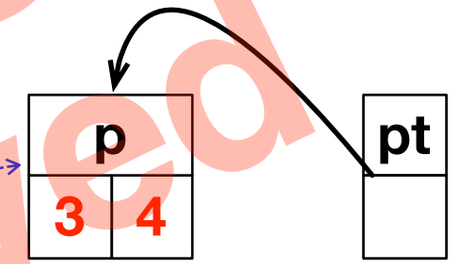
→ ->是class指针特有的运算符（挺形象的），等价于先取指针的值再取成员
比如current->next等价于(*current).next

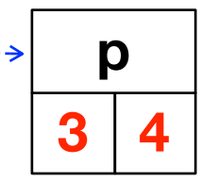
代码参见countoff.cpp

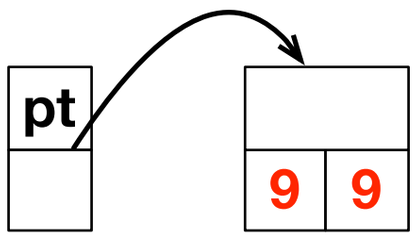
自定义类型的指针运算

```
33  class Point {
34  public:
35      int x,y;
36      Point(int x0,int y0){
37          x=x0; y=y0;
38      }
39      void print(){
40          cout<<"("<<x<<","<<y<<")"<<endl;
41      }
42  };
43
44  void pointerWithClassDemo(){
45      Point p=Point(1,2);
46      Point* pt=&p;    // 指针p指向变量pt的地址
47
48      cout<<"pt="<<pt<<endl;     // 输出指针变量（值是地址）
49      p.print();      // 这三行效果是一样的
50      (*pt).print();
51      pt->print();
52      cout<<endl;
53
54      pt->x=2;      // 改变p指向的值，pt也跟着
55      pt->y=3;
56      p.print();
57      pt->print();
58      cout<<endl;
59
60      pt=new Point(9,9);    // p指向另一个新的地址
61      p.print();      // 这时p指向的值与pt的值无关
62      pt->print();
63  }
```



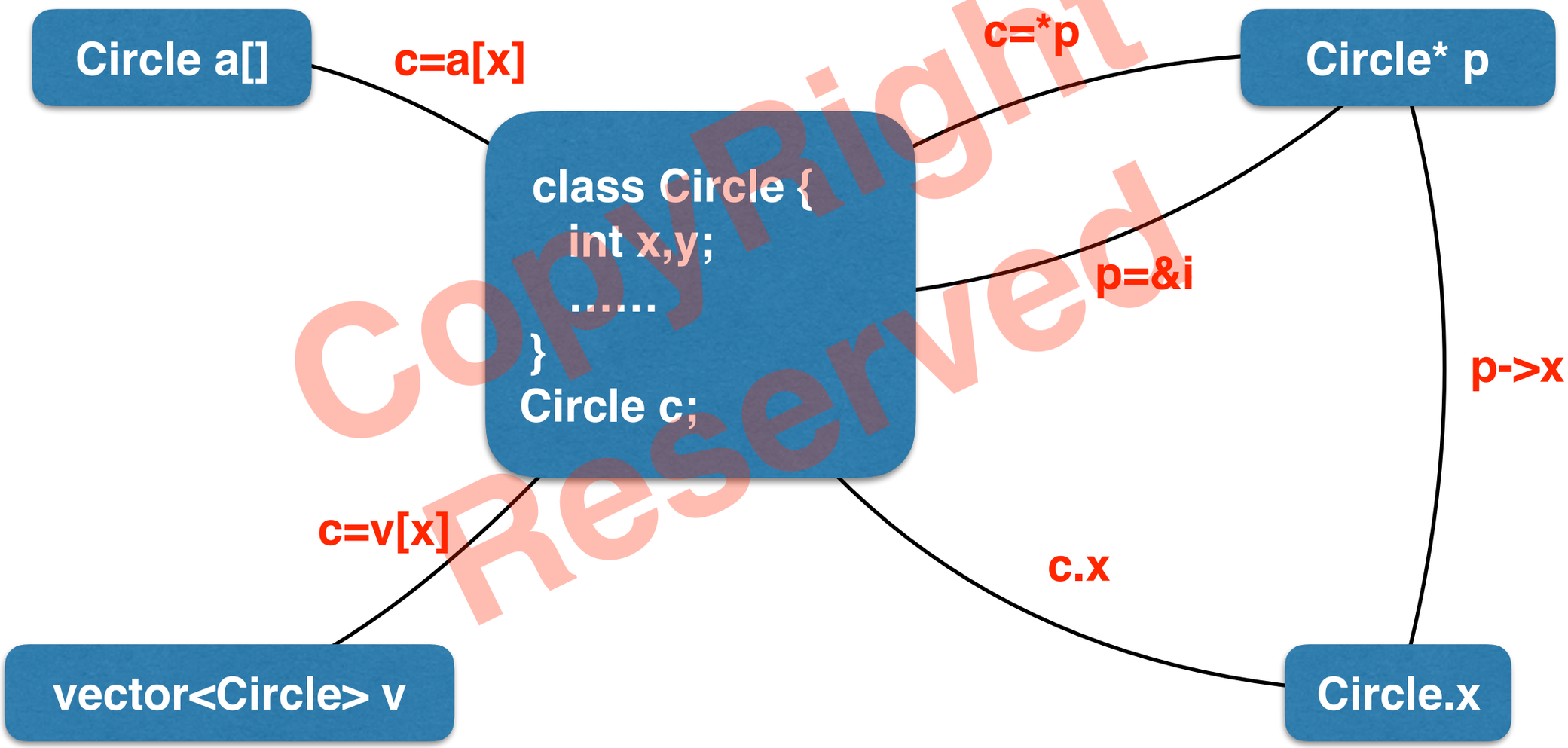







```
pt=0x7fff5fbff748
(1,2)
(1,2)
(1,2)

(2,3)
(2,3)

(2,3)
(9,9)
```

指针运算（自定义类型）



指针：突破线性结构

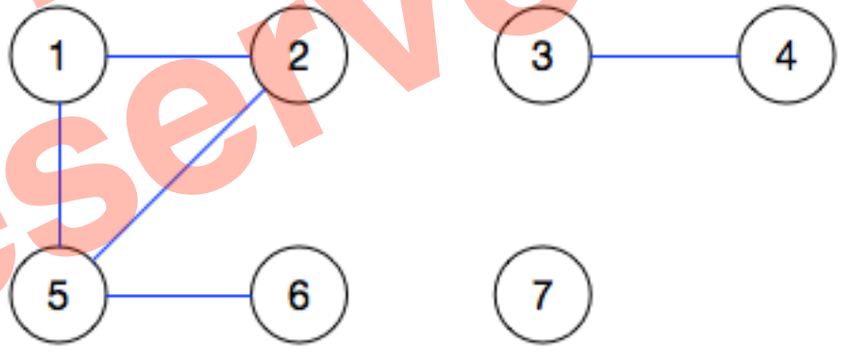
人物关系问题

有n个人物，编号为1..n（n<=1000000）。有m组数据（xi,yi），i=1..m，表示xi,yi两人是同组的，同组关系具有传递性。另外有k组查询（xj,yj），j=1..k，表示查询xi,yi两人是否同组

输入n,m,k,xi,yi,xj,yj（共m+k+1行），输出每组查询的结果（共k行）

样例输入：

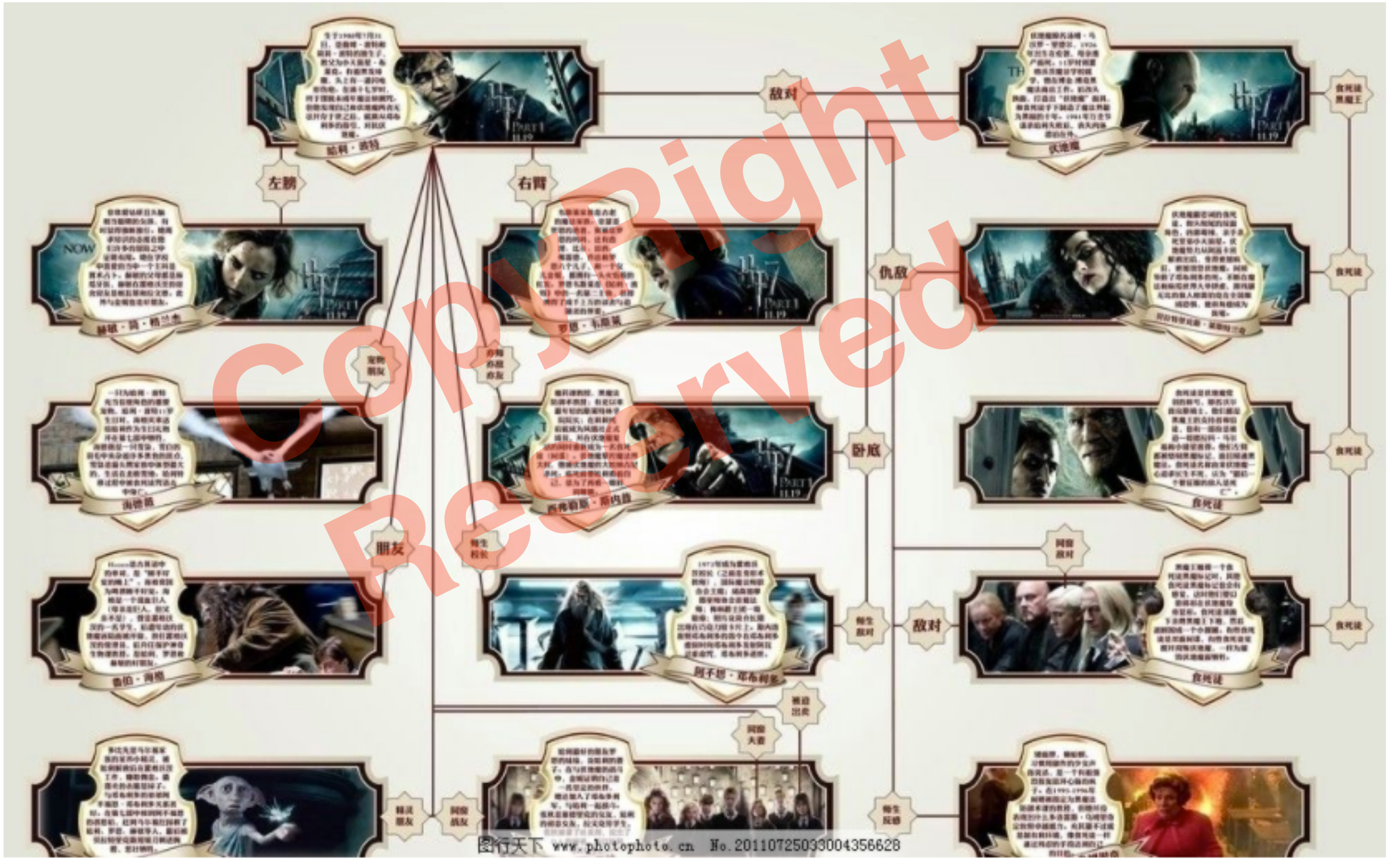
7 5 3
1 2
3 4
5 6
2 5
1 6
1 7
2 6
3 5



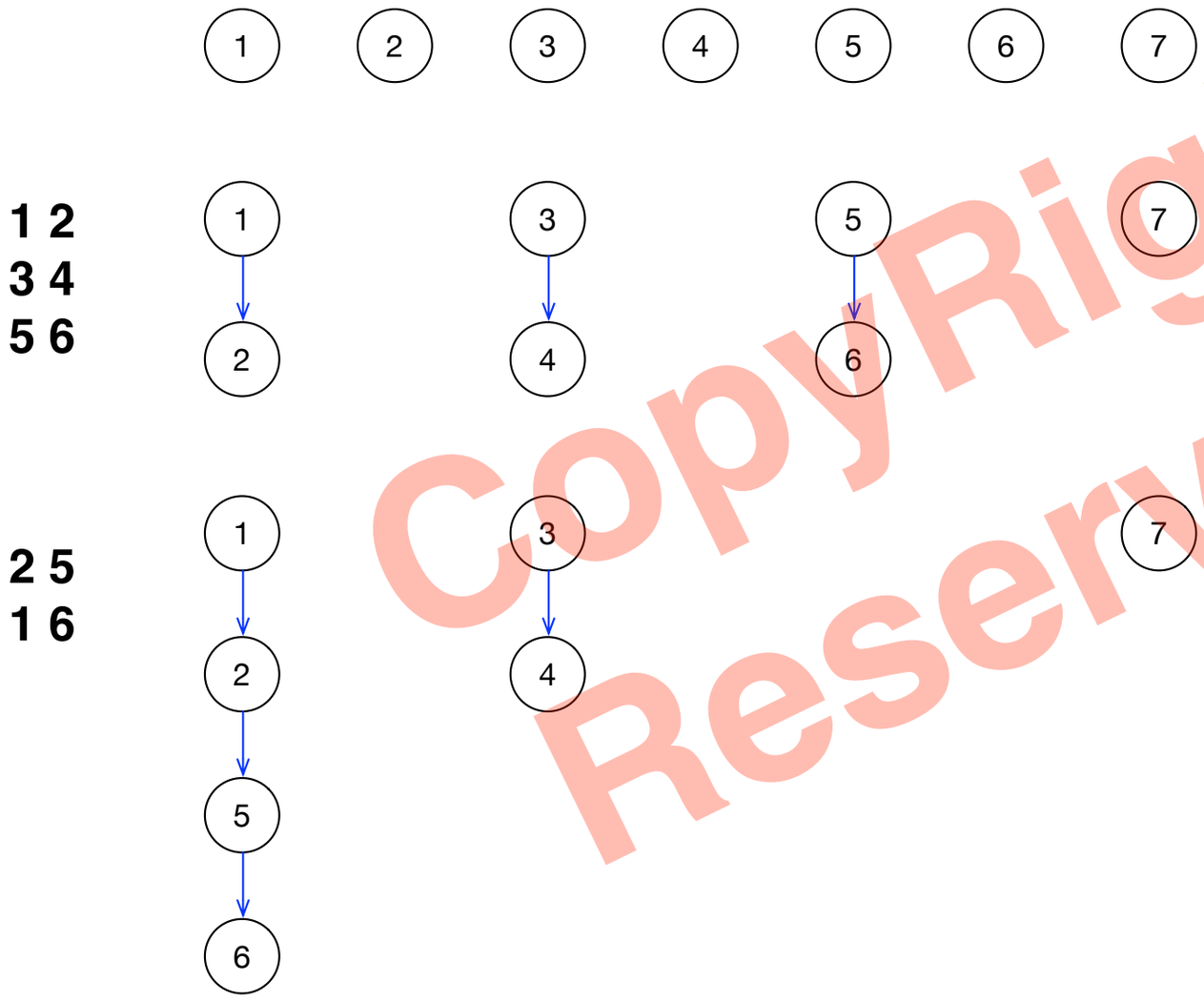
样例输出：

0
1
0

人物关系问题



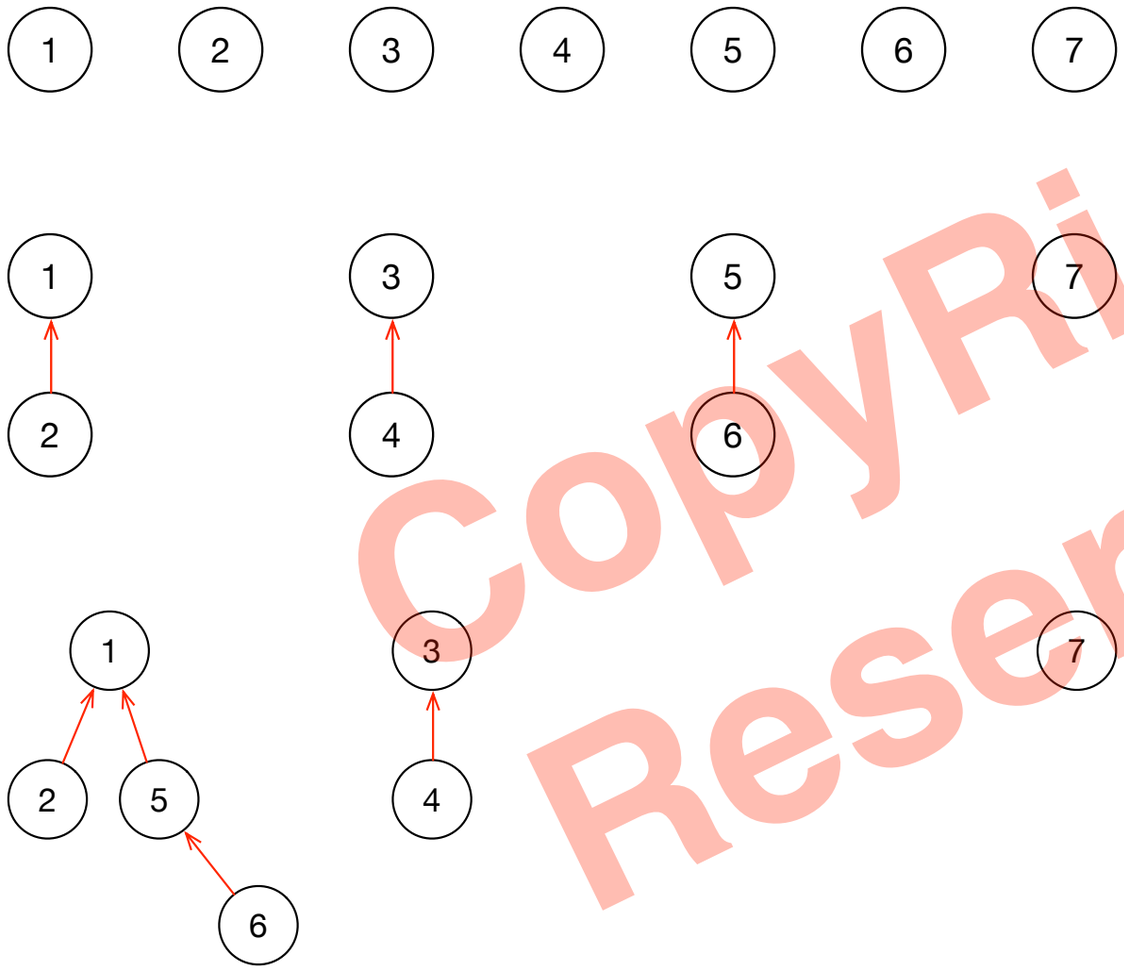
人物关系问题：思路



本质上就是一些集合的合并过程
也许有人想到桶?

怎么实现一大堆桶?
vector数组：vector<int>[]
链表数组：Node*[]

人物关系问题：思路优化



每个组(桶)设置一个组长

合并：一个组长挂到另一个下面

判断同组：组长相同

有了指针就可以很轻松
地玩转各种数据关系

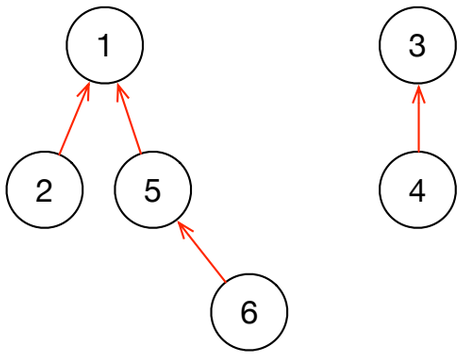
人物关系问题：解答

```cpp
class Person {  // 人物类型
public:
    int id; // 编号
    Person* parent=0;    // 上级，0表示此人为组长
    Person(int id0){     // 初始化函数
        id=id0;
    }

    Person* findLeader(){    // 返回此人所在组的组长
        // 如果此人就是组长，返回自己，否则返回上级的组长（本人与上级是同组的）
        Person* leader= parent==0 ? this : parent->findLeader();
        return leader;
    }
};

int main(int argc, const char * argv[]) {
    int n,m,k;
    cin>>n>>m>>k;
    vector<Person*> people;
    for (int i=0;i<=n;i++){ // 初始化，所有人都自成一组
        people.push_back(new Person(i));
    }

    int x,y;
    for (int i=0;i<m;i++){
        cin>>x>>y;
        Person* leaderX=people[x]->findLeader();
        Person* leaderY=people[y]->findLeader();
        // 如果原先不是一组，则合并
        if (leaderX->id != leaderY->id){
            leaderX->parent=leaderY;
        }
    }

    vector<int> result;
    for (int i=0;i<k;i++){
        cin>>x>>y;  // 判断x,y是否同组
        Person* leaderX=people[x]->findLeader();
        Person* leaderY=people[y]->findLeader();
        result.push_back(leaderX->id==leaderY->id); // 记录结果
    }
    for (int i=0;i<k;i++){  // 输出
        cout<<result[i]<<endl;
    }
    return 0;
}
```

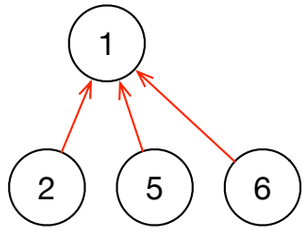
→ 这有一个特殊指针称为this，表示指向自己

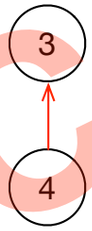
代码参见peoplerelation.cpp

人物关系问题：路径压缩



→ 我们的目的是尽快找到组长
寻找过程中，可以顺便把路径上的成员直接挂到组长下面去。这种技巧称为路径压缩



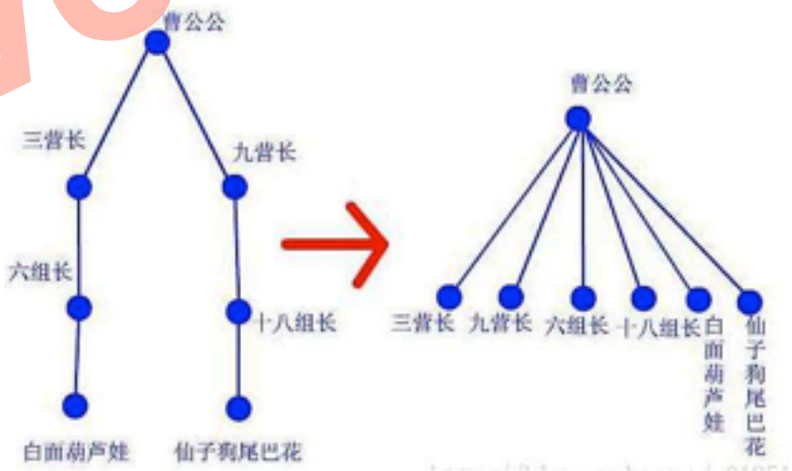



```
22  Person* findLeader(){    // 返回此人所在组的组长
23      // 如果此人就是组长，返回自己，否则返回上级的组长（本人与上级是同组的）
24      Person* leader= parent==0 ? this : parent->findLeader();
25      if (leader->id!=id){  // 路径压缩优化
26          parent=leader;
27      }
28      return leader;
29  }
```

立体式压缩袋
加了一个宽度为40cm
的底部。
棉被收纳能力更加出众
更适合衣橱收纳

并查集

→ 上面用的这种数据结构称为并查集
这是一种专门用于集合合并和查询的牛逼哄哄的数据结构



→ 并查集的平均复杂度是O(ark(n))，ark(n)是Arckmann函数的反函数（参加扩展阅读）
你不需要知道怎么证明（其实我也不知道）
只需要知道O(ark(n))是一种比O(logn)还要低的复杂度，
在人类可想象的范围内ark(n)不会超过4

作业

1.哲学问题

你怎么理解这句话：**世界万物都是相互联系的**

2.说出以下每句语句的含义

```
int* i;          // 样例：声明一个变量i，类型是指向整数的指针
double* d;
Point* p;
*i=3;
p->x=4;
```

（Point是上节课定义的点类型）

NOTE：

1.如果C++实在有困难，可以用Python写；

2.Python也是在有困难，可以写伪代码，但要写的详细

3.上交代码都要求带正确缩进（不论C++、Python、伪代码）

3. 1-3报数 (countoff3.cpp)

与例题一样，只是改为1至3报数，报到3的出列。使用链表

4.（选做）还是1-2报数，要求复杂度O(logn)，你试试？(countoff2better.cpp)

（正确写出的有红包）

5.（选做）说出以下每句语句的含义（晕了吧？+_+）

```
int *p,*q,**r;
int* a[10];
vector<int*> v;
vector<int>* v2;
int i=*p**q;
*p+=i;
q=&(i+*p);
r=&p;
```

（答对6题有红包）

扩展阅读：复合类型的存储结构

```
13  class Point {    // 平面上的点
14  public:
15      int x=0,y=0;
16      Point(int x0,int y0){ //构造函数（初始化）
17          x=x0; y=y0;
18      }
19
20      void print(){    // 打印点的信息（坐标）
21          cout<<"("<<x<<","<<y<<")"<<endl;
22      }
23  };
24
25  int main(int argc, const char * argv[]) {
26      int a[10];
27      cout<<setw(10)<<"a="<<setw(20)<<a<<endl;     // 数组就是指针
28      cout<<setw(10)<<"a+1="<<setw(20)<<a+1<<endl;    // 数组变量+1表示指针后移一个变量位置
29      cout<<setw(10)<<"&a[0]="<<setw(20)<<&a[0]<<endl;     // 首元素取地址，与第一行一样
30      cout<<setw(10)<<"&a[1]="<<setw(20)<<&a[1]<<endl; // 为什么差4？因为一个int占4字节
31      cout<<endl;
32
33      Point p(2,3);
34      cout<<setw(10)<<"&p.x="<<setw(20)<<&p.x<<endl;
35      cout<<setw(10)<<"&p.y="<<setw(20)<<&p.y<<endl;     // 为什么差4？因为一个int占4字节
36      return 0;
37  }
```

- → 简单的类型（如int，float）都存放在一个内存地址，复杂类型呢?
- → 数组是在内存中连续存储的，数组变量其实就是指向首地址的指针
- → 自定义类型中，各成员变量（也称为属性Field）也是依次连续存储的

```
        a=       0x7fff5fbff770
      a+1=       0x7fff5fbff774
    &a[0]=       0x7fff5fbff770
    &a[1]=       0x7fff5fbff774

     &p.x=       0x7fff5fbff2e8
     &p.y=       0x7fff5fbff2ec
Program ended with exit code: 0
```

扩展阅读：Ackermann函数

Ackermann函数的完整形式是一个二元函数，定义如下：

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1,A(m,n-1)) & \text{sinon} \end{cases}$$

之前提到的ark(n)是A(n,n)的反函数

m/n	1	2	3	4	5	通项
0	2	3	4	5	6	n+1
1	3	4	5	6	7	n+2
2	5	7	9	11	13	2n+3
3	13	29	61	125	253	2^(n+3)-3
4	65533	19729位数天文数字	2^(19729位数)位数都是天文数字	位数的位数都是天文数字	位数的位数的位数都是天文数字	2^(2^(....2^2))-3，共n+3个2
5	大到非人类					语言难以形容