按位与运算符（&）

参加运算的两个数据，按二进制位进行“与”运算。

运算规则：0&0=0;　　0&1=0;　　1&0=0;　　1&1=1;

即：两位同时为“1”，结果才为“1”，否则为0

例如：3&5　即 0000 0011& 0000 0101 = 00000001　　因此，3&5的值得1。

另，负数按补码形式参加按位与运算。

“与运算”的特殊用途：

（1）清零。如果想将一个单元清零，即使其全部二进制位为0，只要与一个各位都为零的数值相与，结果为零。

（2）取一个数中指定位

方法：找一个数，对应X要取的位，该数的对应位为1，其余位为零，此数与X进行“与运算”可以得到X中的指定位。

例：设X=10101110，

取X的低4位，用 X & 0000 1111 = 00001110　即可得到；

还可用来取X的2、4、6位。

按位或运算符（|）

参加运算的两个对象，按二进制位进行“或”运算。

运算规则：0|0=0;　　0|1=1;　　1|0=1;　　1|1=1;

即 ：参加运算的两个对象只要有一个为1，其值为1。

例如:3|5　即 00000011　| 0000 0101 = 00000111　　因此，3|5的值得7。

另，负数按补码形式参加按位或运算。

“或运算”特殊作用：

（1）常用来对一个数据的某些位置1。

方法：找到一个数，对应X要置1的位，该数的对应位为1，其余位为零。此数与X相或可使X中的某些位置1。

例：将X=10100000的低4位置1 ，用X　| 0000 1111 = 1010 1111即可得到。

异或运算符（^）

参加运算的两个数据，按二进制位进行“异或”运算。

运算规则：0^0=0;　　0^1=1;　　1^0=1;　　1^1=0;

即：参加运算的两个对象，如果两个相应位为“异”（值不同），则该位结果为1，否则为0。

“异或运算”的特殊作用：

（1）使特定位翻转找一个数，对应X要翻转的各位，该数的对应位为1，其余位为零，此数与X对应位异或即可。

例：X=10101110，使X低4位翻转，用X ^0000 1111 = 1010 0001即可得到。

（2）与0相异或，保留原值 ，X ^ 00000000 = 1010 1110。

下面重点说一下按位异或,异或其实就是不进位加法,如1+1=0，,0+0=0,1+0=1。

异或的几条性质:

1、交换律

2、结合律（即(a^b)^c == a^(b^c)）

3、对于任何数x，都有x^x=0，x^0=x

4、自反性: a^b^b=a^0=a;

异或运算最常见于多项式除法，不过它最重要的性质还是自反性：A XOR B XOR B = A，即对给定的数A，用同样的运算因子（B）作两次异或运算后仍得到A本身。这是一个神奇的性质，利用这个性质，可以获得许多有趣的应用。 例如，所有的程序教科书都会向初学者指出，要交换两个变量的值，必须要引入一个中间变量。但如果使用异或，就可以节约一个变量的存储空间： 设有A,B两个变量，存储的值分别为a，b，则以下三行表达式将互换他们的值 表达式 （值）:

a=a^b;

b=b^a;

a=a^b;

应用举例1:

1-1000放在含有1001个元素的数组中，只有唯一的一个元素值重复，其它均只出现

一次。每个数组元素只能访问一次，设计一个算法，将它找出来；不用辅助存储空

间，能否设计一个算法实现？

解法一、显然已经有人提出了一个比较精彩的解法，将所有数加起来，减去1+2+...+1000的和。

这个算法已经足够完美了，相信出题者的标准答案也就是这个算法，唯一的问题是，如果数列过大，则可能会导致溢出。

解法二、异或就没有这个问题，并且性能更好。

将所有的数全部异或，得到的结果与1^2^3^...^1000的结果进行异或，得到的结果就是重复数。

应用举例2(综合&和^):(题目链接:http://gdutcode.sinaapp.com/problem.php?cid=1051&pid=7)

一系列数中,除两个数外其他数字都出现过两次,求这两个数字,并且按照从小到大的顺序输出.例如 2 2 1 1 3 4.最后输出的就是3 和4

```
1. #include
2. #include
3. #include
4. #include
5. usingnamespacestd;
6. #define N 1000010
7. int a[N];
8. intmain()
9. {
10. //freopen("why.in", "r", stdin);
11. //freopen("why.out", "w", stdout);
12. int t;
13. scanf("%d", &t);
14. while(t--) {
15. int n;
```

```
16. scanf("%d", &n);
17. int x = 0;
18. for(int i = 1; i <= n; i++) {
19. scanf("%d", &a[i]); x ^= a[i];
20.         }
21. int num1 = 0, num2 = 0;
22. int tmp = 1;
23. while(!(tmp & x)) tmp <<= 1;
24. cout<endl;
25. for(int i = 1; i <= n; i++) {
26. if(tmp & a[i]) num1 ^= a[i];
27. else num2 ^= a[i];
28.         }
29. printf("%d %d\n", min(num1, num2), max(num1, num2));
30.     }
31. return0;
32. }
```

这个题就是首先在输入的时候一直异或,就可以把这两个数异或的乘积找出来,就比如上例中x=3^4;

然后找一个变量tmp来分开这两个数.按位与的话可以发现会分开这两个数分别存在num1和num2中.然后就有结果了.

左移运算符（<<）

将一个运算对象的各二进制位全部左移若干位（左边的二进制位丢弃，右边补0）。

例：a = a<< 2将a的二进制位左移2位，右补0，

左移1位后a = a *2;

若左移时舍弃的高位不包含1，则每左移一位，相当于该数乘以2。

右移运算符（>>）

将一个数的各二进制位全部右移若干位，正数左补0，负数左补1，右边丢弃。

操作数每右移一位，相当于该数除以2。

例如：a = a>> 2 将a的二进制位右移2位，

左补0 or 补1得看被移数是正还是负。

复合赋值运算符

位运算符与赋值运算符结合，组成新的复合赋值运算符，它们是：

&= 例：a &=b 相当于a=a& b

|= 例：a |=b 相当于a=a |b

>>= 例：a >>=b 相当于a=a>> b

<<= 例：a<<=b 相当于a=a<< b

^= 例：a ^= b 相当于a=a^ b

运算规则：和前面讲的复合赋值运算符的运算规则相似。

不同长度的数据进行位运算

如果两个不同长度的数据进行位运算时，系统会将二者按右端对齐，然后进行位运算。

以“与”运算为例说明如下：我们知道在C语言中long型占4个字节，int型占2个字节，如果一个long型数据与一个int型数据进行“与”运算，右端对齐后，左

边不足的位依下面三种情况补足，

（1）如果整型数据为正数，左边补16个0。

（2）如果整型数据为负数，左边补16个1。

（3）如果整形数据为无符号数，左边也补16个0。

如：long a=123;int b=1;计算a& b。

如：long a=123;int b=-1;计算a& b。

如：long a=123;unsigned intb=1;计算a & b。