And 运算

and运算通常用于二进制的取[位操作](https://baike.baidu.com/item/%E4%BD%8D%E6%93%8D%E4%BD%9C)，例如一个数 and 1的结果就是取[二进制](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6)的最末位。这可以用来判断一个整数的奇偶，二进制的最末位为0表示该数为[偶数](https://baike.baidu.com/item/%E5%81%B6%E6%95%B0)，最末位为1表示该数为奇数

or 运算

or运算通常用于二进制特定位上的无条件[赋值](https://baike.baidu.com/item/%E8%B5%8B%E5%80%BC)，例如一个数or 1的结果就是把二进制最末位强行变成1。如果需要把二进制最末位变成0，对这个数or 1之后再减一就可以了，其实际意义就是把这个数强行变成最[接近](https://baike.baidu.com/item/%E6%8E%A5%E8%BF%91/1356208)的偶数

 xor运算

xor运算的逆运算是它本身，也就是说两次异或同一个数最后结果不变，即（a xor b) xor b = a。xor运算可以用于简单的加密

加法和减法互为逆运算，并且加法满足交换律。把#换成+，把@换成-，我们可以写出一个不需要临时变量的swap过程（Pascal）。

procedure swap(var a,b:longint);

begin

a:=a + b;

b:=a - b;

a:=a - b;

end;

procedure swap(var a,b:longint);

begin

a:=a xor b;

b:=a xor b;

a:=a xor b;

end;

 not运算 ~ ===

not运算的定义是把内存中的0和1全部取反。使用not运算时要格外小心，你需要注意整数类型有没有符号。如果not的对象是[无符号整数](https://baike.baidu.com/item/%E6%97%A0%E7%AC%A6%E5%8F%B7%E6%95%B4%E6%95%B0" \t "_blank)（不能表示负数），那么得到的值就是它与该类型[上界](https://baike.baidu.com/item/%E4%B8%8A%E7%95%8C" \t "_blank)的差，因为无符号类型的数是用00到$FFFF依次表示的。下面的两个程序（仅语言不同）均返回65435。

shl运算 << ===

a shl b就表示把a转为二进制后左移b位（在后面添b个0）。例如100的二进制为1100100，而110010000转成十进制是400，那么100 shl 2 = 400。可以看出，a shl b的值实际上就是a乘以2的b次方，因为在二进制数后添一个0就相当于该数乘以2。

通常认为a shl 1比a \* 2更快，因为前者是更底层一些的操作。因此程序中乘以2的操作请尽量用左移一位来代替

shr运算 >> ===

和shl相似，a shr b表示二进制右移b位（去掉末b位），相当于a除以2的b次方（取整）。我们也经常用shr 1来代替div 2，比如[二分查找](https://baike.baidu.com/item/%E4%BA%8C%E5%88%86%E6%9F%A5%E6%89%BE" \t "_blank)、堆的插入操作等等

The following two code samples, written in the programming language [C++](https://en.wikipedia.org/wiki/C%2B%2B), both determine if the given unsigned integer **x** is a [power of two](https://en.wikipedia.org/wiki/Power_of_two).

*// The obvious method*

unsigned int x = ...;

bool isPowerOfTwo;

**if** (x > 0) {

*/\* Divide by two as long as the next division is an integer,*

*and if it isn't, check if the number is 1 (meaning the number is*

*some power of two) \*/*

**while** ((x % 2) == 0) {

x = x / 2;

}

isPowerOfTwo = (x==1);

}

**else** { *// zero is never a power of two*

isPowerOfTwo = false;

}

*// A method using bit manipulation*

bool isPowerOfTwo = x && !(x & (x - 1));

The second method uses the fact that powers of two have one and only one bit set in their binary representation:

x == 0...010...0

x-1 == 0...001...1

x & (x-1) == 0...000...0

If the number is neither zero nor a power of two, it will have '1' in more than one place:

x == 0...1...010...0

x-1 == 0...1...001...1

x & (x-1) == 0...1...000...0