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;

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