一般情况： a>>b 的值就是 a 乘以 2的b次方分之一

a<<b 的值就是 a乘以 2的b次方

# 转载：https://baike.1688.com/doc/view-d1750791.html

先说左移,左移就是把一个数的所有位都向左移动若干位,在C中用<<运算符.例如:

　　int i = 1;

　　i = i << 2; //把i里的值左移2位

　　也就是说,1的2进制是000...0001(这里1前面0的个数和int的位数有关,32位机器,gcc里有31个0),左移2位之后变成 000...0100,也就是10进制的4,所以说左移1位相当于乘以2,那么左移n位就是乘以2的n次方了(有符号数不完全适用,因为左移有可能导致符号变化,下面解释原因)

　　需要注意的一个问题是int类型最左端的符号位和移位移出去的情况.我们知道,int是有符号的整形数,最左端的1位是符号位,即0正1负,那么移位的时候就会出现溢出,例如:

　　int i = 0x40000000; //16进制的40000000,为2进制的01000000...0000

　　i = i << 1;

　　那么,i在左移1位之后就会变成0x80000000,也就是2进制的100000...0000,符号位被置1,其他位全是0,变成了int类型所能表示的最小值,32位的int这个值是-2147483648,溢出.如果再接着把i左移1位会出现什么情况呢?在C语言中采用了丢弃最高位的处理方法,丢弃了1之后,i的值变成了0.

　　左移里一个比较特殊的情况是当左移的位数超过该数值类型的最大位数时,编译器会用左移的位数去模类型的最大位数,然后按余数进行移位,如:

　　int i = 1, j = 0x80000000; //设int为32位

　　i = i << 33; // 33 % 32 = 1 左移1位,i变成2

　　j = j << 33; // 33 % 32 = 1 左移1位,j变成0,最高位被丢弃

　　在用gcc编译这段程序的时候编译器会给出一个warning,说左移位数>=类型长度.那么实际上i,j移动的就是1位,也就是33%32后的余数.在gcc下是这个规则,别的编译器是不是都一样现在还不清楚.

　　总之左移就是: 丢弃最高位,0补最低位

　　再说右移,明白了左移的道理,那么右移就比较好理解了.

　　右移的概念和左移相反,就是往右边挪动若干位,运算符是>>.

　　右移对符号位的处理和左移不同,对于有符号整数来说,比如int类型,右移会保持符号位不变,例如:

　　int i = 0x80000000;

　　i = i >> 1; //i的值不会变成0x40000000,而会变成0xc0000000

　　就是说,符号位向右移动后,正数的话补0,负数补1,也就是汇编语言中的算术右移.同样当移动的位数超过类型的长度时,会取余数,然后移动余数个位.

　　负数10100110 >>5(假设字长为8位)，则得到的是 11111101

　　总之,在C中,左移是逻辑/算术左移(两者完全相同),右移是算术右移,会保持符号位不变.实际应用中可以根据情况用左/右移做快速的乘/除运算,这样会比循环效率高很多.

　　例:C语言中左移<<表示乘以2，右移>>表示除以2，这是由计算机工作原理导致的！但是要是7，二进制数为0111，右移一位得3.5，但是右移之后二进制数变成0011，是3。不一样啊，怎模解释呢？？

　　移位操作符的两个操作数必须是整型的。整个移位表达式的值的类型也是整型的，而且，左移位操作符与右移位操作符的运算并不对称。0111右移一位是把最后一位的1去掉，左边补个0，得0011，转换为十进制是3，这是正确的。并不等同于除以2 。