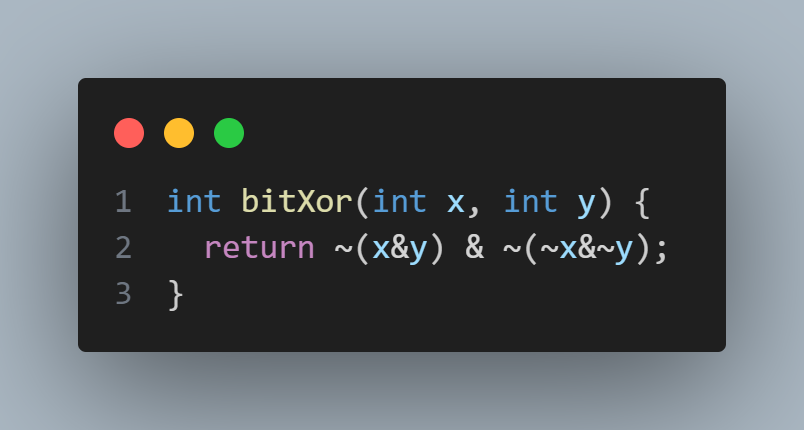
# DataLab实验报告

## BitXor

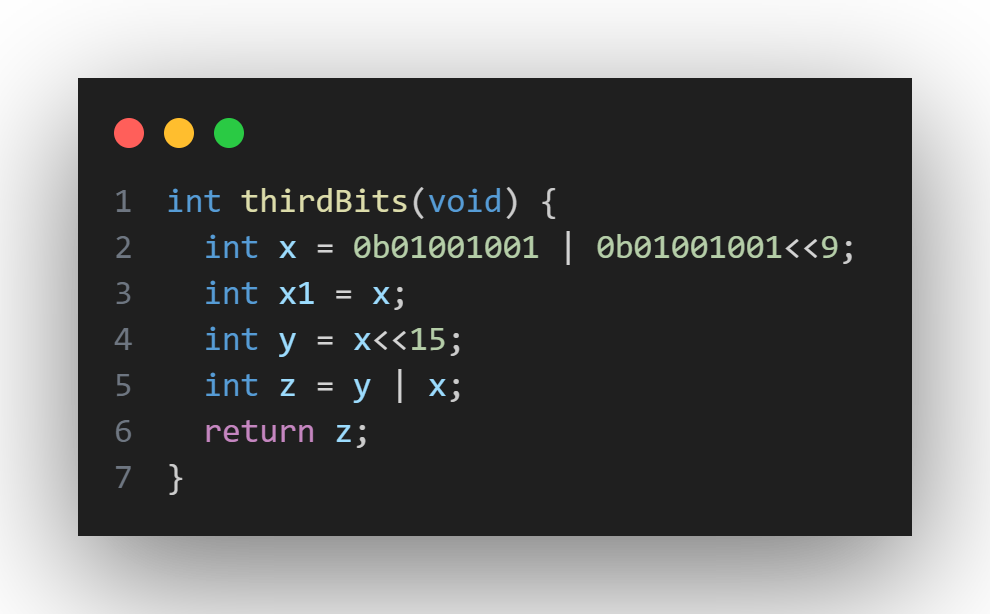


**由定义可知：a^b = (~a & b) | (a & ~b)**

**= (~a | a）&（~a | ~b）&（b | a）&（b | ~b ）**

**= ~( a & b）& ~(~a & ~b）**

## ThirdBits

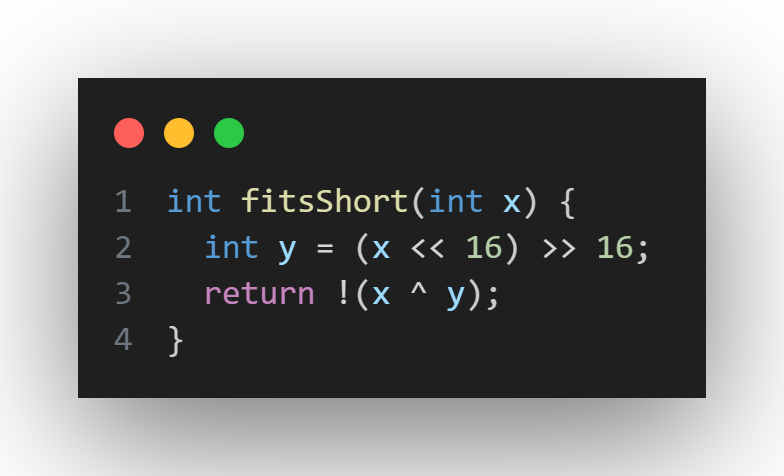


**0b01001001为最小8位的情况**

**x为后16位情况**

**z为32位的情况**

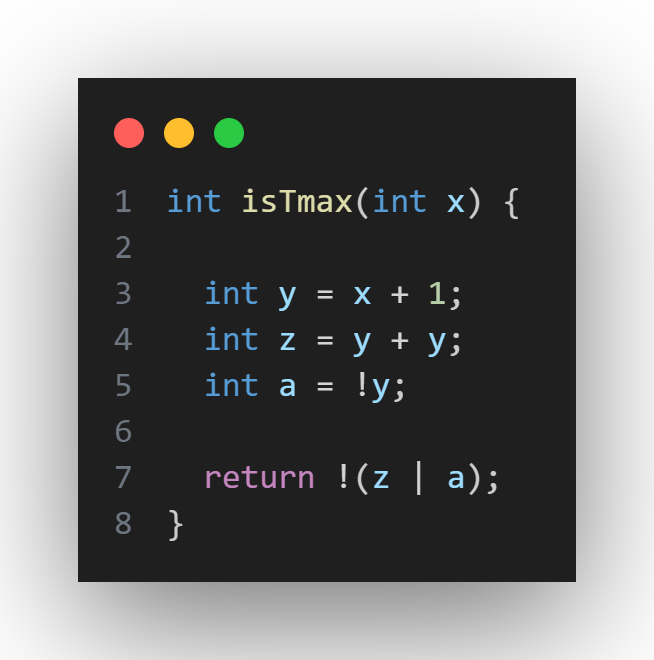
## fitsShort



**对x左移16位再右移16位得到y，**

**若x和y相等则x满足条件返回1，不然则返回0**

## IsTmax

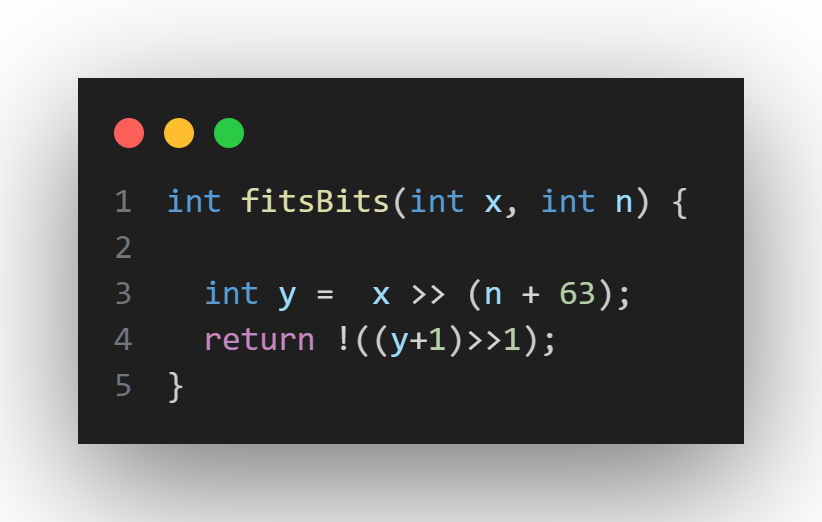


**本题主要关注两个点，x=Tmax时和x=-1。当且仅当此时，z = 0。**

**又有x=-1时a=1； x=Tmax时a=0；**

**故可由z和a共同确定x是否Tmax**

## Fitsbits

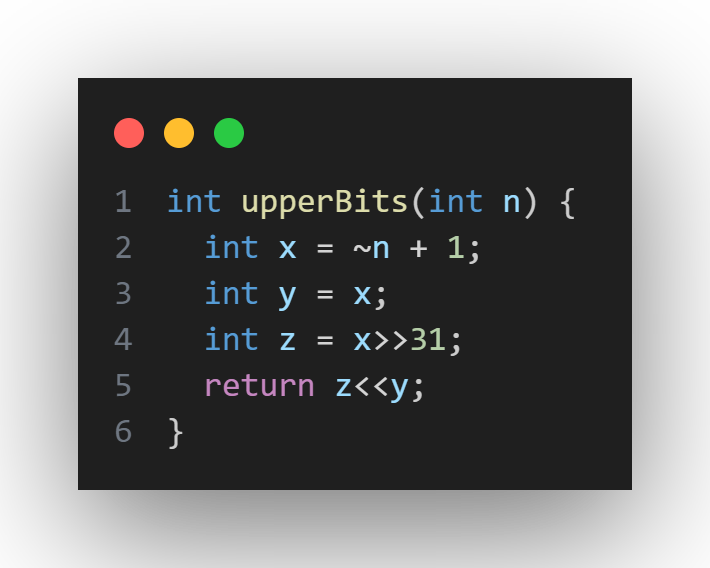


**利用ub(Undefined Behavior)获得y，即x右移n-1**

**此时若x满足条件则y为0(x>=0时)或0xFFFFFFFF(x<0)**

**然后y+1再右移1位，仅当x满足条件时结果为0。**

## UpperBits

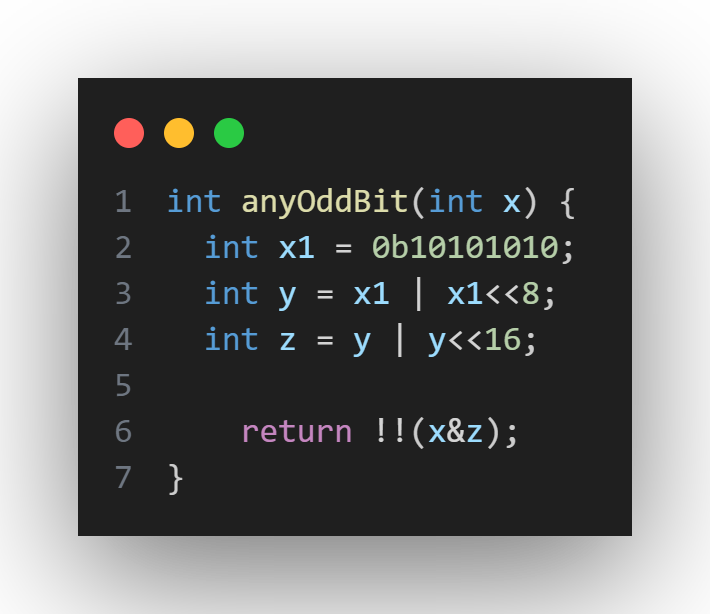


**x = -n;**

**z = 0xFFFFFFFF(x != 0时) 或 z = 0(x == 0时)**

**结果返回使用ub，即z<<(32-n)**

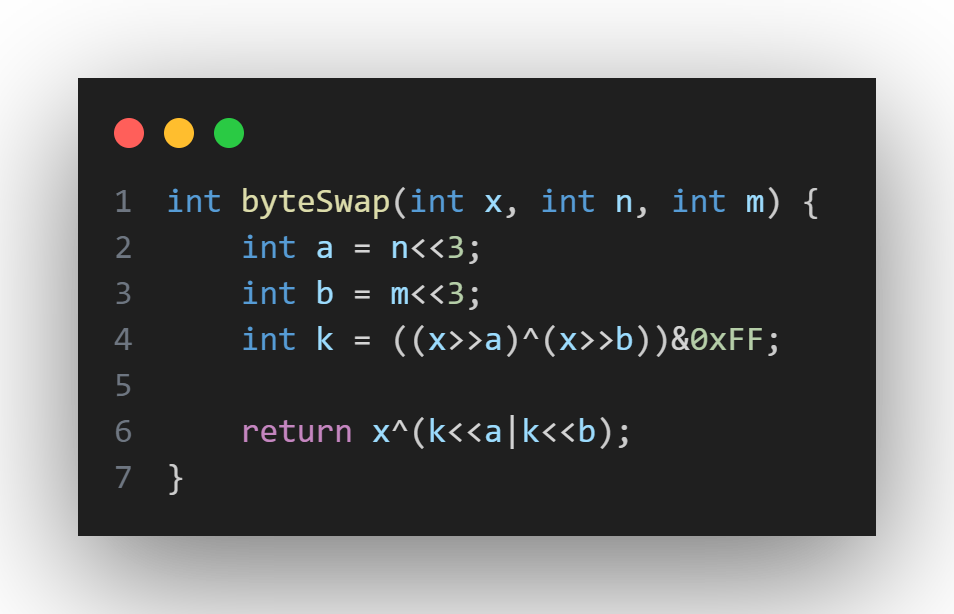
## anyOddBit



**x1，y都是为了得到z，即得到0bAAAAAAAA**

**若x的奇数位有1，则x&z一定不为0；反之一定为0。由此再用！！即得到最终结果**

## ByteSwap

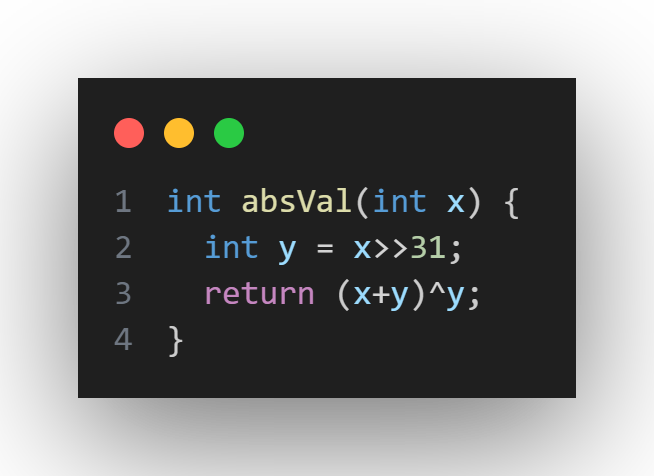


**记x为0xA3A2A1A0（Ai代表一个字节）**

**则k = 0xAn ^ 0xAm**

**由0^x = x; x^x^y = y;可得x^(k<<a) 为仅将n处的字节换成m处的，x^(k<<b)同理将m的字节换成n的**

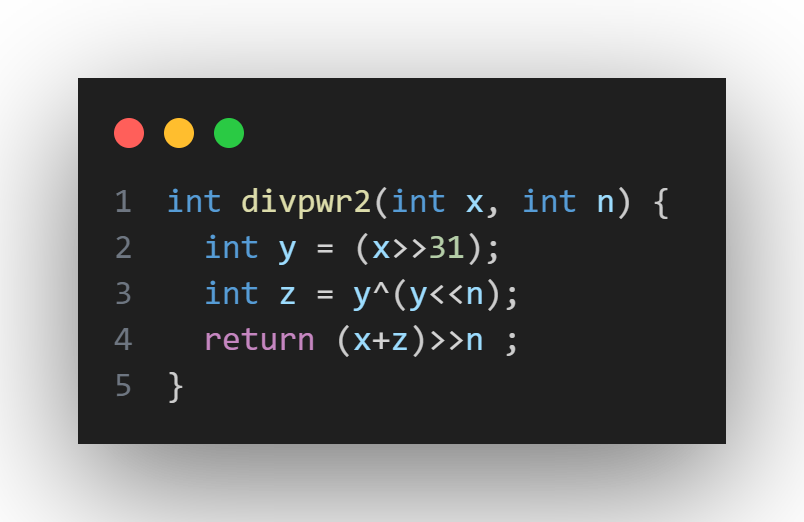
## AbsVal



**当x>=0时，y = 0 ，（x+y）^y = x**

**当x<0时，y = 0xFFFFFFFF ，（x+y）^y = ~(x-1) = ~x+1 = -x**

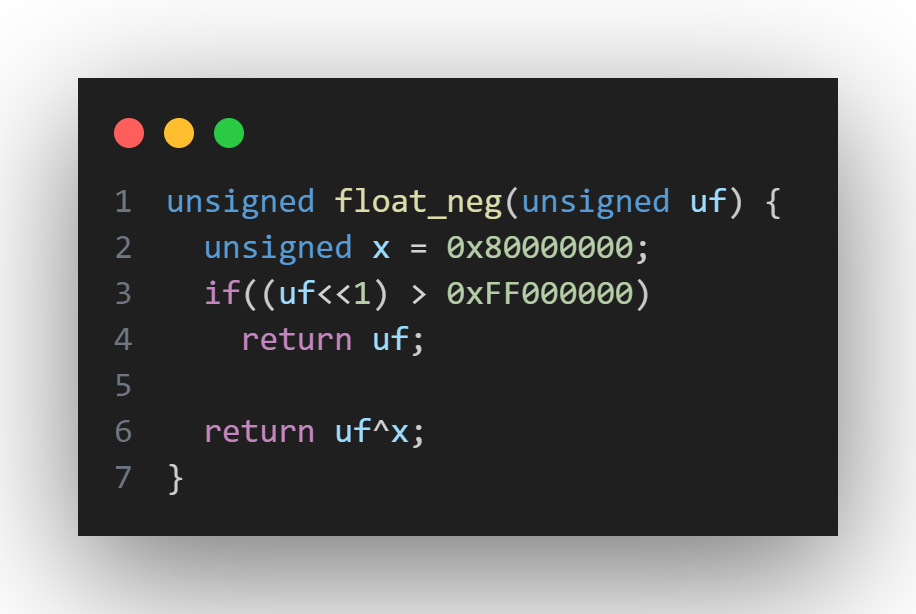
## Divpwr2



**当x>=0时，y=z=0，结果为x>>n**

**当x<0时，由于负数的移位取整方向和除法不同，需要加上特定的数之后再移位，即加上z = 。此时y = 0xFFFFFFFF，y<<n再和y异或正好为。**

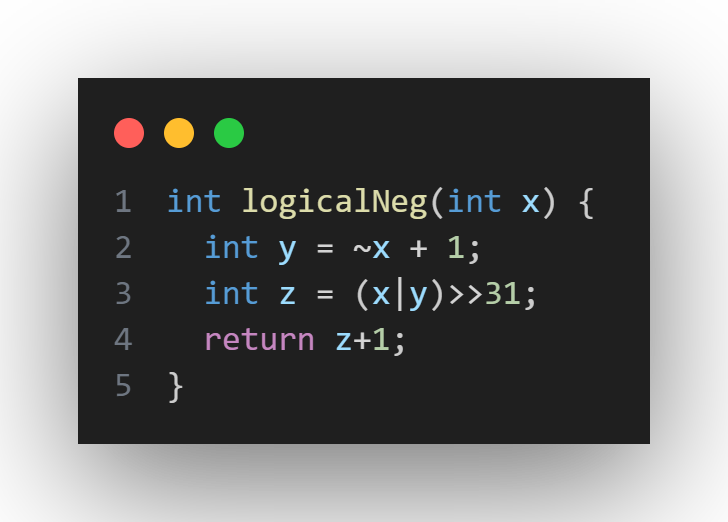
## Float\_neg

****

**当(uf<<1) > 0xFF000000时，判定为NaN，return原数**

**不然则与0x80000000做异或，使第一位符号位取反，然后return**

## logicalNeg

****

**y= -x**

**当x不为0时，x与y必有一个负数，则z必定等于0xFFFFFFFF**

**而当x=0时，z = 0**

**最后返回z+1正好为结果**

## bitMask

****

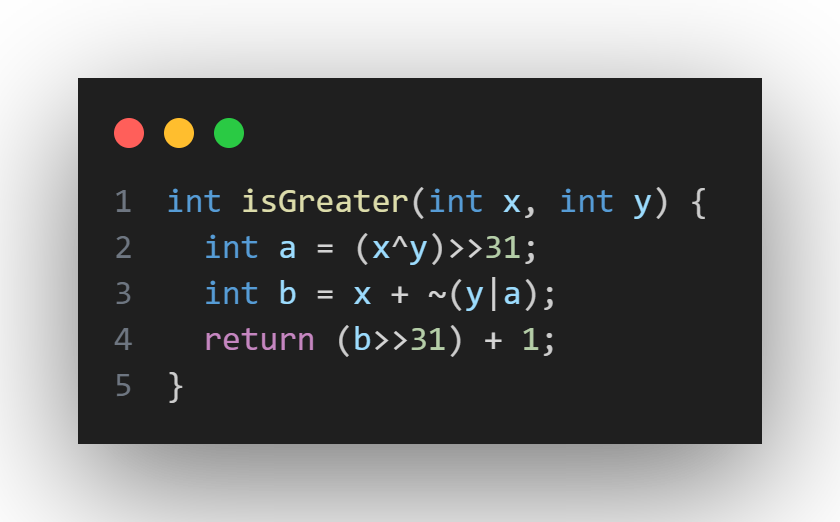
**记最后结果为0b 000…111…000…**

**则 y = 0b 000…111111…**

**z = 0b11111111…000**

**两者与运算即为结果，并且当high<low时正好返回0**

## isGreater

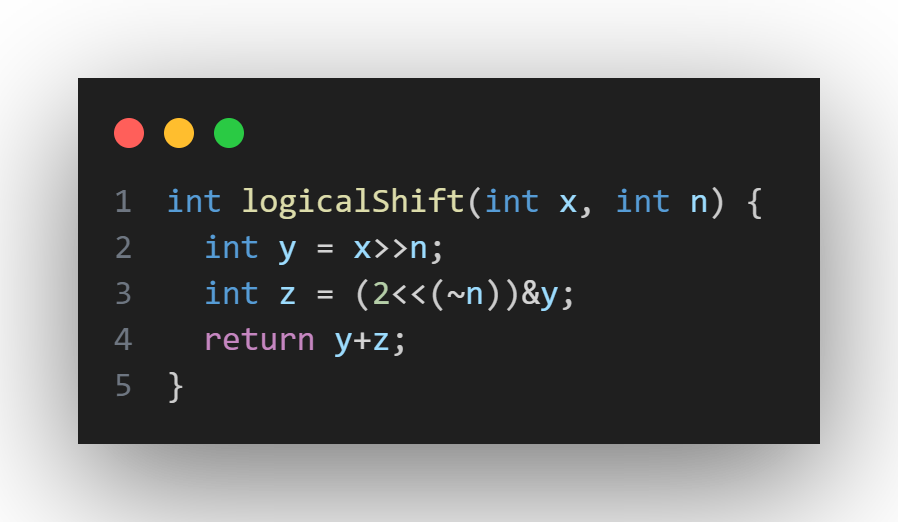
****

**a用来判断x，y是否异号（因为异号相减需考虑溢出，同号不需）**

**若x y同号，b = x-y-1，然后通过符号位区分结果**

**若x y异号，b = x, 此时若x为正则x>y，x为负则x<y**

## logicalShift

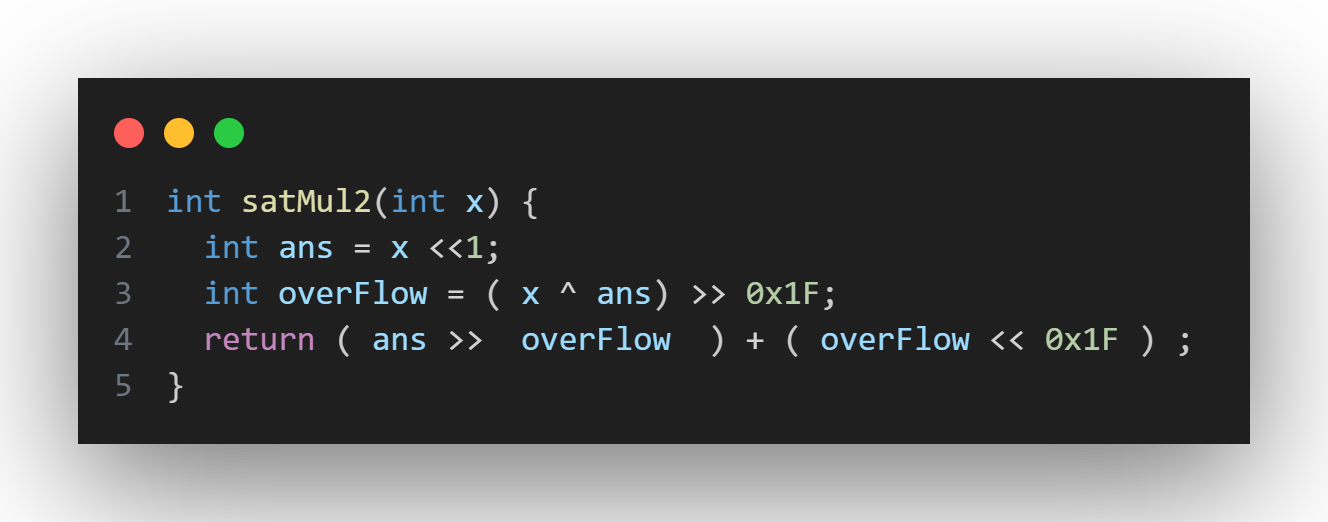
****

**2<<~n即2右移31-n位**

**若y为正数，则z = 0，返回y**

**若y为负数，则z = 2<<~n。y+z使y因为移动产生的1全变成0**

## satMul2

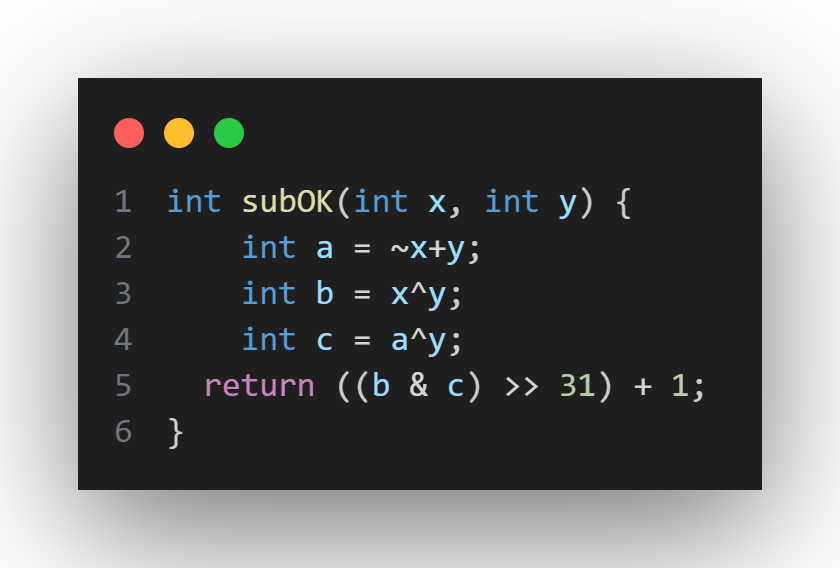
****

**ans为2\*x，若无溢出，则直接return ans**

**若有溢出，则overflow为0xFFFFFFFF,**

**此时，若x>0，ans右移overflow(即31位)为0xFFFFFFFF，加上0x80000000即为Tmax。同理若x<0，ans右移31位为0，加上0x80000000即为Tmin**

## subOK

****

**a = y - x -1。b用来判断x y是否异号。c用来判断a与y是否异号当x与y同号时不可能溢出，则b的符号位为0，返回1**

**当x与y异号时，b的符号位为1：**

**若x为正y为负：**

**a若溢出为正，c的符号位为1，最终返回0；**

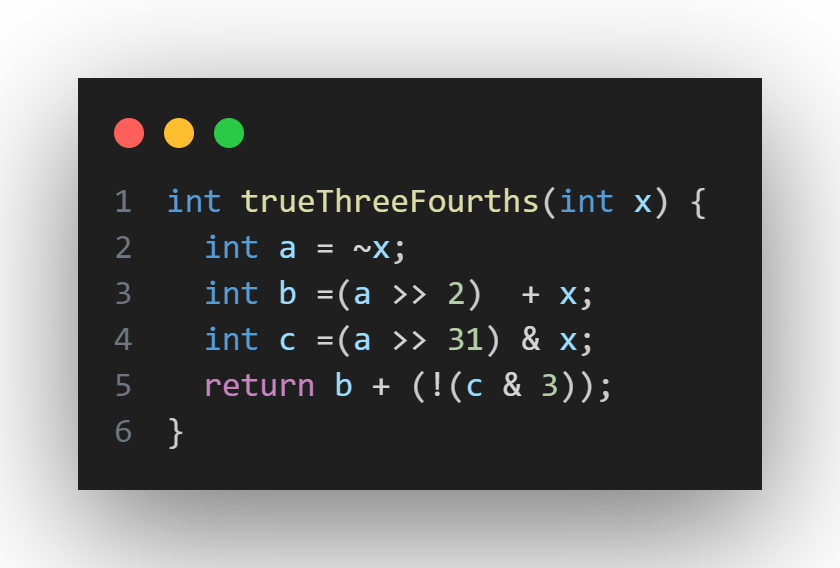
**a若未溢出则c符号位为0，最终返回1**

**若y为正x为负：**

**a若溢出为负，c的符号位为1，最终返回0；**

**a若未溢出则c符号位为0，最终返回1**

## trueThreeFourths

****

**a = -x-1**

**b = a/4 + x =**

**但此时若x为正且第三位不为1时或x为负会造成误差，故构造c**

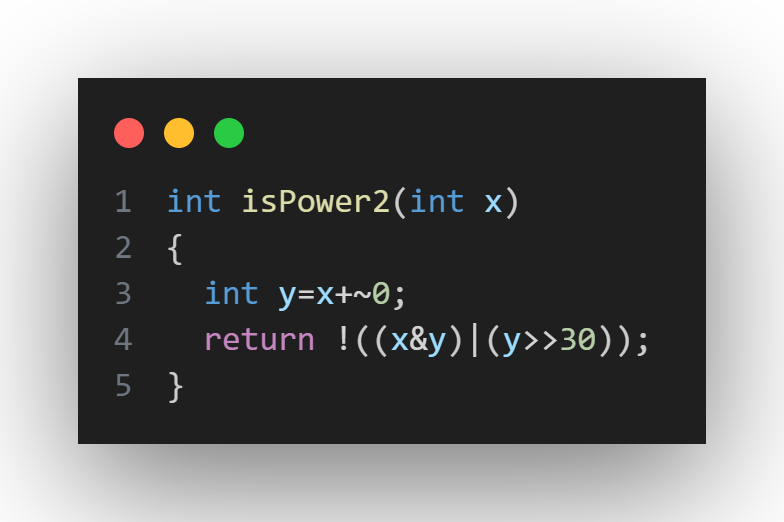
**当x<0时，c = 0，return b+1;**

**当x>=0时，c = x**

**当x第三位不为1时，return b+1**

**当x第三位为1时，return b**

## isPower2

****

**y = x - 1**

**当x是2的幂时，x形同0b000…10000…**

**y形同0b000…01111…**

**故x&y = 0，但是我们还要排除x是负数和x是0的干扰，**

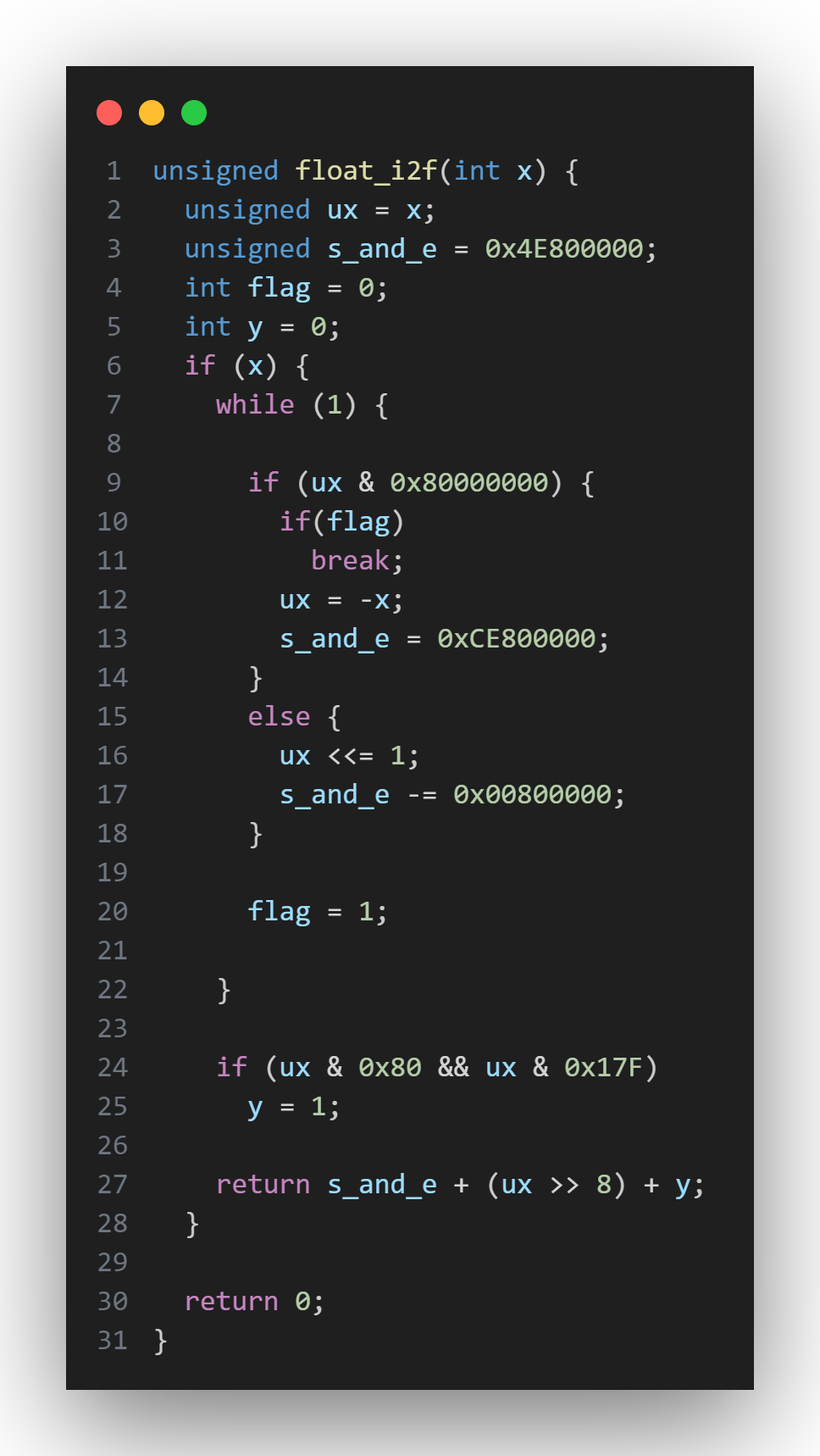
**注意到当x是负数(不是Tmin)或x是0时，y<0，(x&y)|y可以排除干扰**

**注意到当x>0时，x若是幂，y最大为0x1FFFFFFF，则y应至少右移30位才能消除对正确结果的影响**

**注意到当x = Tmin时，y = 0x7FFFFFFF，则y右移最多30位才能排除干扰**

**综上应该返回!((x&y)|(y>>30))**

## Float\_i2f

****

**易知把x变成浮点数，E的取值在0~30。则把s\_and\_e初始化为0x4E800000，代表符号位为0，exp为10011101(即E为30)。**

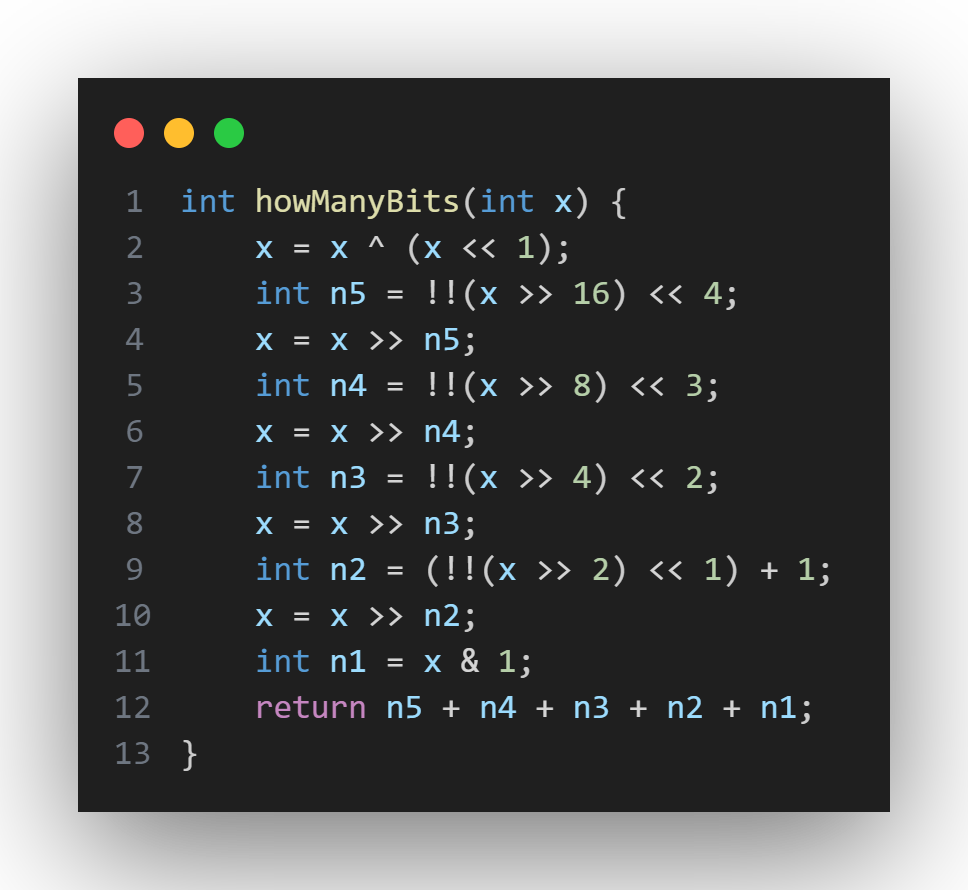
**然后进入循环，flag仅仅用来辅助第一次循环，若第一次循环发现x<0，则把x取正然后把s\_and\_e的符号位赋值为1**

**之后的循环就是不断让ux左移，每成功左移一次就让exp-1，直到ux的最左位为1，但是注意到这样会使exp多减一次1**

**所以让ux右移8位空出符号位和exp的左7位，然后加上s\_and\_e正好让exp补上多减的1，同时也满足frac省略第一个1的特性**

**但是这样做会造成精度的损失，寻找规律发现当ux&0x80与ux&0x17F同时满足时，结果需要加1**

## howManyBits

****

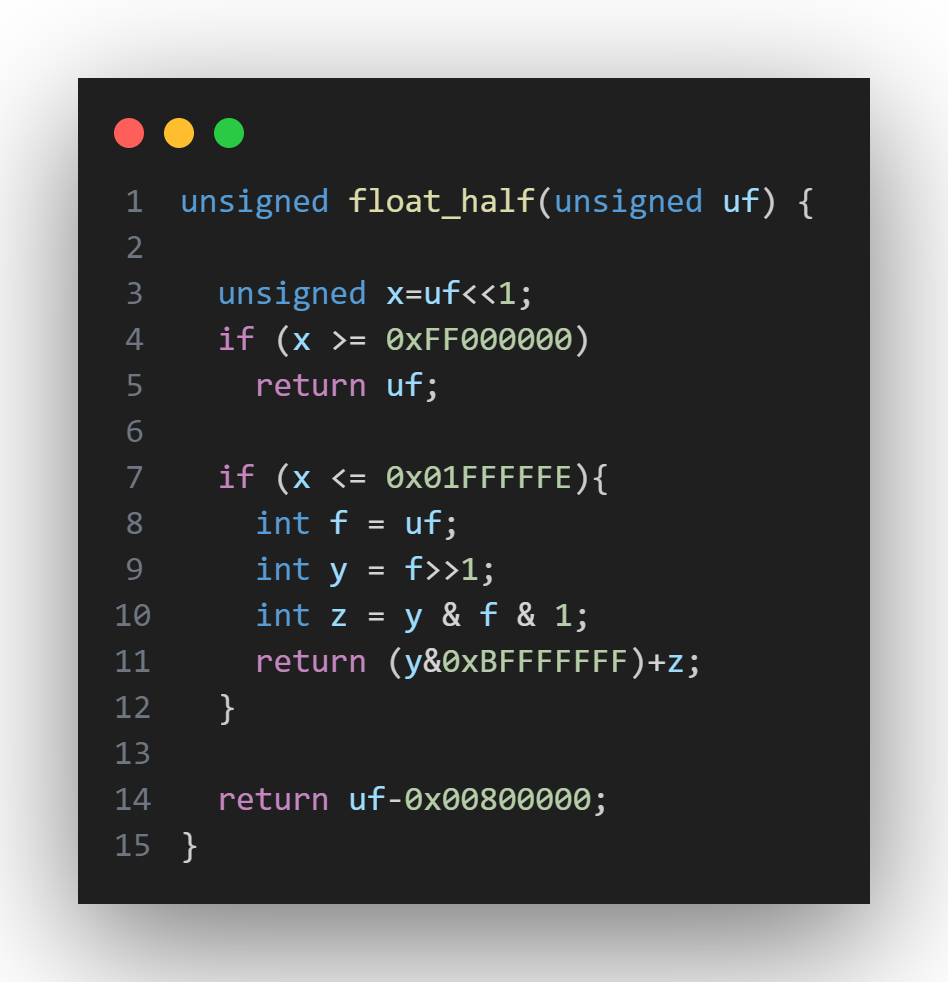
**由于返回数 1<= n <= 32,所以n-1用二进制表示有5位，n1到n5就分别是最低位到最高位，最后再加1就是n**

**第一步异或是为了消除负数左边全是1的影响**

**接下来的操作相似，即检查x的左位是否有数字，有则ni赋值成再乘，同时移动x以进行下一步计算**

**由于最后的返回值要＋1，我们发现如果把这个1加在n2上并使x接下来多移动一位，就可以直接通过x&1来判断n1，节省了操作符**

## Float\_half

****

**x为uf去掉符号位的结果，用来判断不同情况**

**x>=0xFF000000时，exp为FF，uf即为无穷或NaN，返回uf。**

**X<=0x01FFFFFE时，exp为1或0，此时uf除以2后exp为0，frac为原来的右移1位。因此我们让y=f>>1，由于f是int类型，y会自动保留符号位，我们只需让y&0xBFFFFFFF以消除符号位右移进入exp(即从左向右第二位)的影响。然后，考虑舍入会有1的误差，令变量z，当f的右边第一位和第二位都为1时，我们让结果加上1以平衡误差**

**除了上述两种情况外，其他时候我们只需让exp-1即可，即uf-0x00800000**