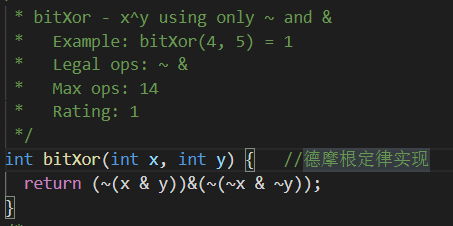
**实验报告**

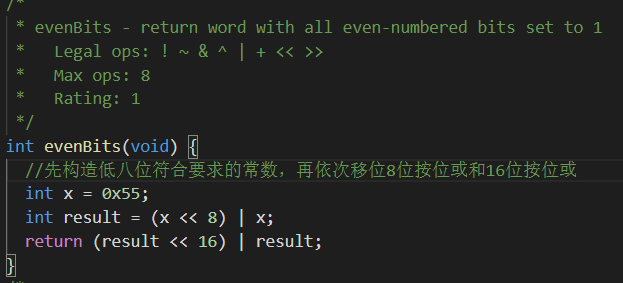
（代码中的注释比较简洁(字少)，文字分析更详细）

1. bitXor



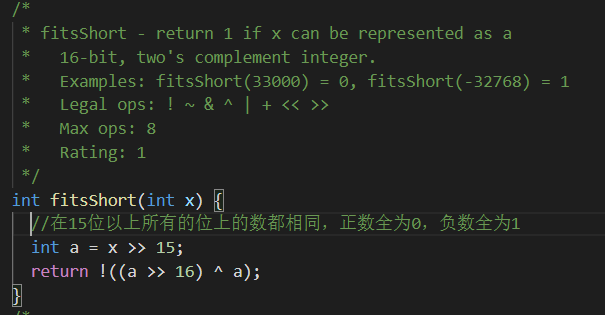
本题用取反和位与实现异或，采用德摩根定律即可实现，注意到异或取反是同或，可以节省一个操作符

2.evenBits



由于只能构造八位的常数，先构造低八位符合要求的常数0x55，然后将其左移8位取或，再左移16位取或，即可得到满足要求的数

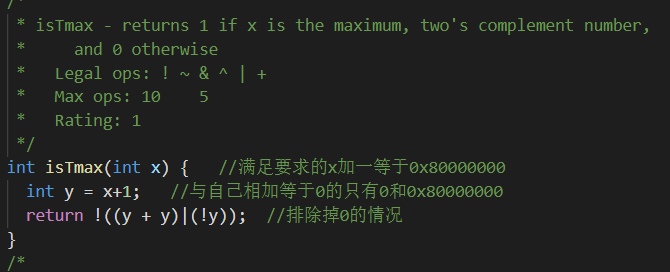
3.fitsShort



如果x能被表示为16位的二进制数，那么x的15位以上的所有位上的数字应该相同，与其符号位一致，即正数全为0，负数全为1。

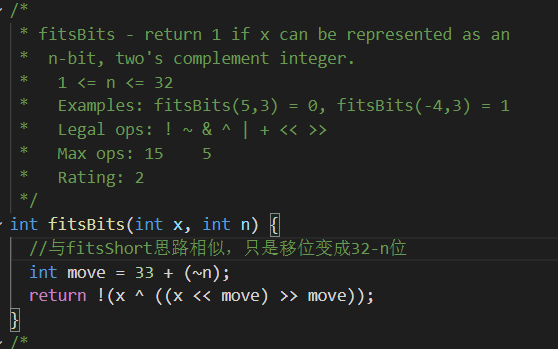
利用右移按符号位补1或0的性质，所以只要创建变量a等于x右移15位，则a的高16位一定统一为0或1，再据此检测a是否全为1或0即可

4.isTmax



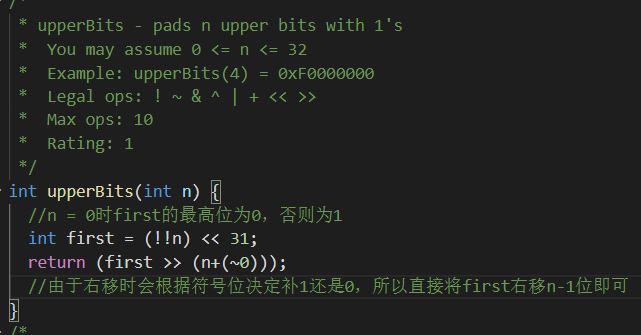
二进制有符号数的最大值为0x7fffffff，加1后会等于0x80000000。在int范围内的所有数中，只有0和0x80000000与自身相加后等于0，即让y+y等于0的只有0和0x80000000，所以只需要再与！y按位或，排除掉0的情况再取非即可。

5.fitsBits



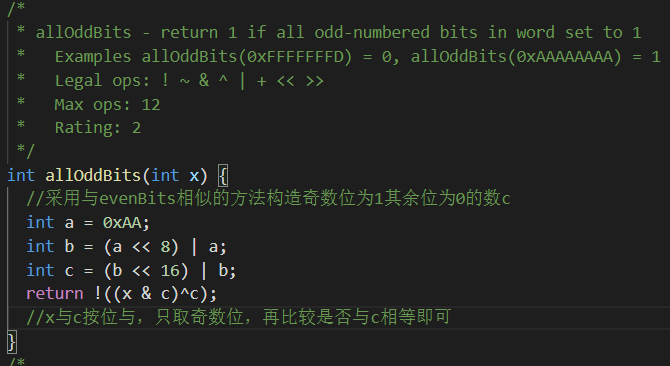
若int类型的x可以表示为有符号的n位二进制数，则x左移32-n位，再右移32-n位后得到的结果与它自身相等，检验相等用异或再取非即可。

6.upperBits



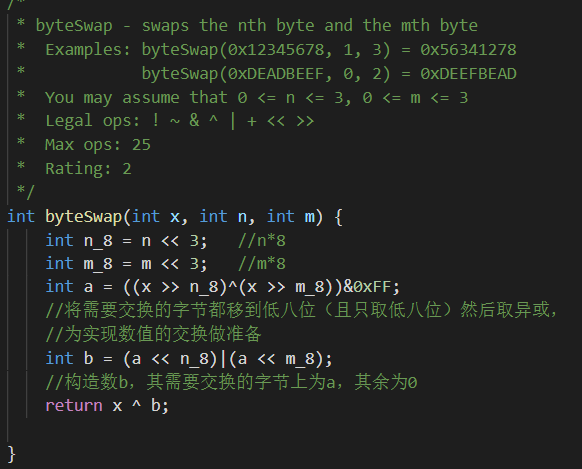
只有n=0时结果的最高位为0，其余情况都为1，所以构造first表示结果的最高位。由于右移时根据符号位决定补1或者补0，所以直接将first右移n-1位即可。

7.allOddBits



利用evenBits的思路，构造数c，使其的所有奇数位都为1，偶数位为0，然后将x与c按位与，可以取出x的奇数位，然后只需要与c异或取非，比较是否相等，即可检验x的奇数位是否全为1。

8.byteSwap

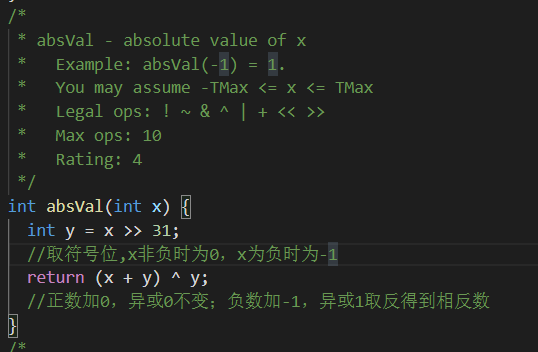


将m、n分别左移三位，得到m、n乘以8作为需要移动的位数。

利用异或可以不借助第三个变量实现两个变量的值交换的特性，将需要交换的字节都移到最低字节上取异或，所得的数a的低八位与m字节异或可得到n字节，与n字节异或可得到m字节。

所以再构造数b，使其第m、n个字节处均为a的低八位，其余字节处为0，则b与x异或即可实现m、n字节的交换，并保持其他字节不变。

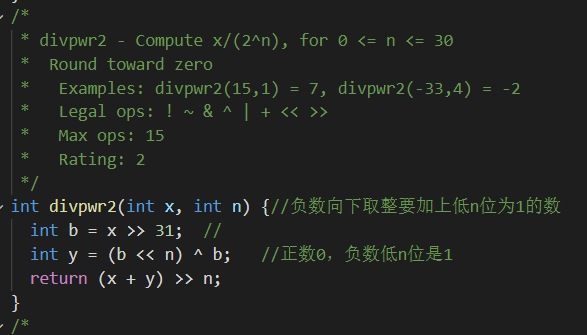
9.absVal



创建y表示x的符号位。x非负时y为0，x为负时y为-1(0xffffffff)。而负数变正数，只需要减一再取反。

利用与0异或等于不变，与1异或等于取反的特点，返回的表达式(x+y)^y对正数不产生影响，但能使负数变为正数，从而实现绝对值的功能。

10.divpwr2

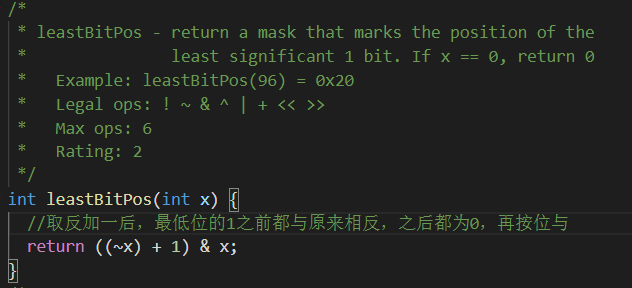


正数的除法直接移位便可以实现题目的取整要求，向下取整。但负数需要向上取整，要在移位前加上一个低n位全为1，其余位全为0的数，才可满足题目要求。

所以构造数b表示x的符号位，x非负则b为0；x为负则b为-1。

利用b构造y，x非负则y为0，x为负则y的低n位全为1，其余位全为0。则x+y恰好满足需要，x为正时实现向下取整，x为负时实现向上取整。

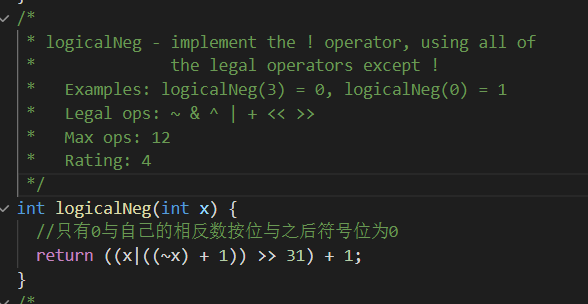
11.leastBitPos



本题利用了x取反加一后的特点，即最低位的1之前的数都取反，之后的数都为0.

据此特性，x取反加一后与x自身按位与，只有最低的1所在位能保留1，其余位将全变为0，满足要求。

12.logicalNeg



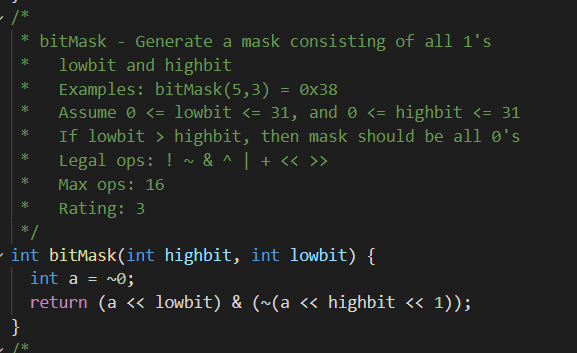
除0和0x80000000以外的数，取反加一后符号位都与原来相反，按位或得到1。

而0x80000000取反加一后等于它本身，符号位为1，按位或得到1。

因此只有0取反加一后与自己按位或得到的符号位仍为0，利用这一点即可判断是否为0：

将(x|((~x) + 1))右移31位，x为0会得到0，x非0则会得到0xffffffff，再加一则分别得到1和0，恰好实现逻辑非的功能。

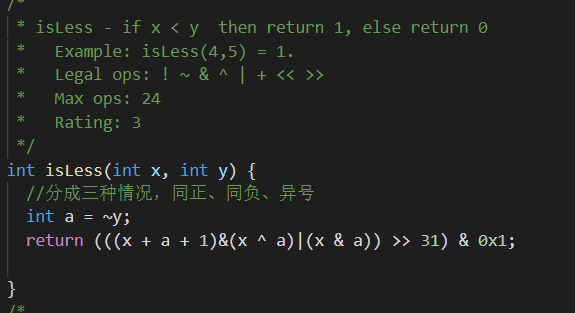
13.bitMask



构造两个数，其中一个在lowbit及以上的位全为1，通过将~0左移lowbit位得到。另外一个在highbit及以下的位全为1，通过将~0右移highbit+1位后取反得到。

将这两个数按位与，则只有lowbit与highbit之间的数为1，即可得到满足要求的结果。

14.isLess



若直接用x-y（x+~y+1）的正负情况来判断，可能会存在溢出的情况导致结果不准确，因此细分为三种情况：

x>0,y>0,x-y<0;

x<0,y<0,x-y<0;

x<0,y>0;

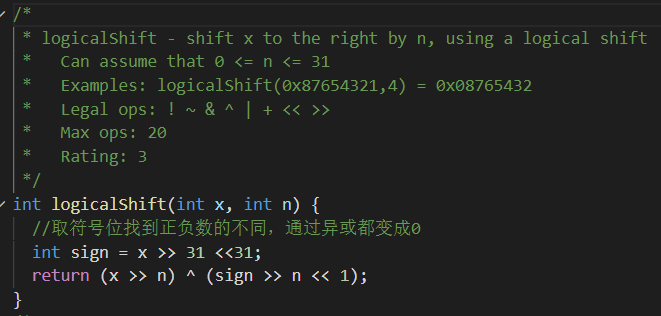
取a=~y；

第一二种情况，x、y同号，x-y不会溢出，直接判断x-y的符号即可（&1 |0对x-y的符号位不产生影响。）

第三种情况，xy异号，x-y可能会溢出，需要另外判断。这种情况下&(x^a)相当于&0，可以消掉x-y的影响。另外发现，只有x为负，y为非负的情况下x&a的符号位为1，因此|(x&a)可以区分异号的两种情况。

综上，通过(x + a + 1)&(x ^ a)|(x & a)的符号位即可判断x是否小于y

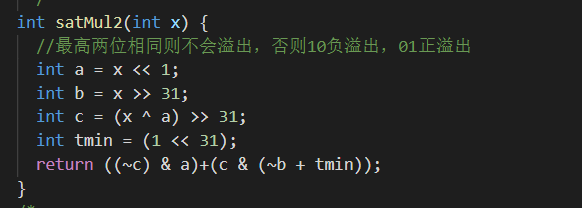
15.logicalShift



创建sign变量，通过右移31位再左移31位，只保留x的符号位。

x直接右移n位后，前n-1位都与符号位相同，所以将sign右移n-1位（右移n位再左移1位）后，与x>>n取异或，即可将前n-1位全置为0，从而实现逻辑右移。

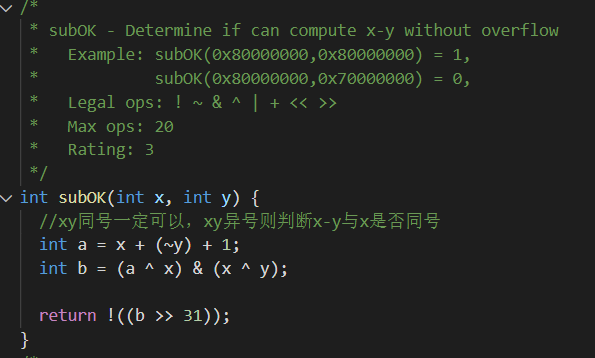
16.satMul2



第32位与第31位相同时，不会发生溢出；不同时，若为10则发生负溢出，01发生正溢出。

所以先用变量c来比较这两位上的数字是否相同，并将其扩展为32位以便操作。两位数字相同时，不会溢出，c为0，返回的结果为a（x左移1位）；两位数字不同时，发生溢出，利用tmin + 0xffffffff = tmax的性质，对变量b取反再加tmin，恰好返回对应的正负溢出结果。

17.subOK

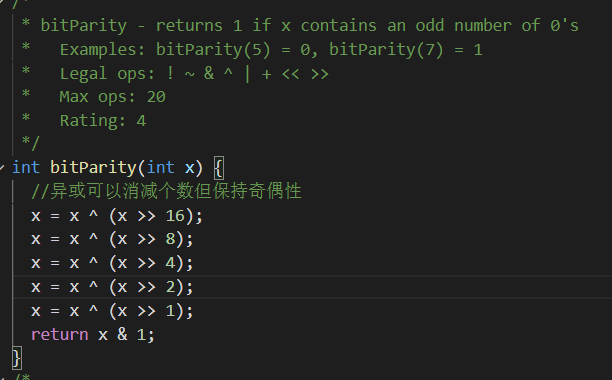


创建变量a = x-y（x+~y+1），如果xy同号，x-y一定不会溢出。如果xy异号，若x-y不发生溢出，则x与x-y同号；若x-y发生溢出，则x与x-y异号。

据此可以创建变量b=(a^x)&(x^y)，则只有当x-y发生溢出时，b的最高位为1；其余情况b的最高位都为0。

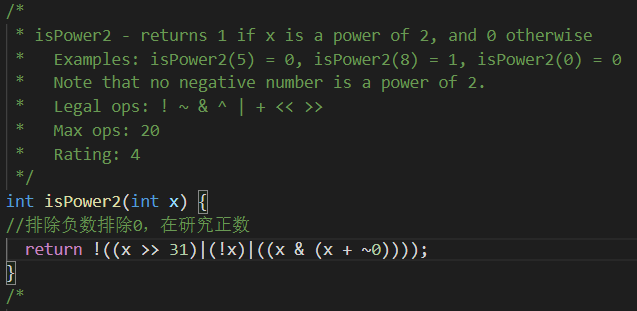
因此将b右移31位再取非，则只会在x-y溢出时返回0。

18.bitParity



与课上老师举的例子相似，0^0=0,0^1=1,1^0=1,1^1=0。可以看出异或运算能在消掉1的个数的情况下保持其奇偶性不变。所以这样依次折半取异或，最终可以在x的最低位上反映出1的个数的奇偶性：若最低位为0则为偶数个，最低位为1则为奇数个。x&1即可取最终结果。

19.isPower2

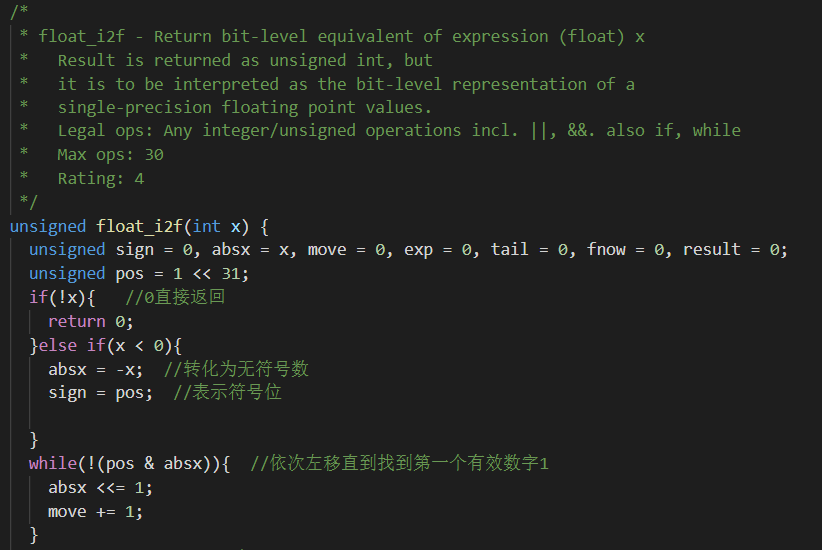


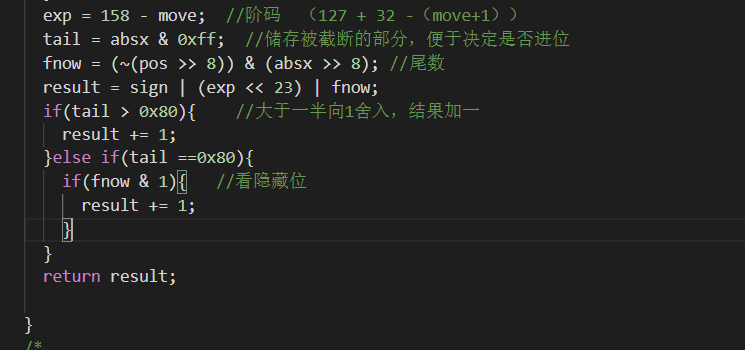
满足要求的x应该只有一位是1，其他位都是0，且不能符号位为1。因此满足要求的x与x-1（x+~0）按位与的结果是0。

用x右移31位的方式取符号位，排除负数。

用！x的方式排除0，因此(x >> 31)|(!x)|(x & (x + ~0))只在x为2的幂时返回0，再取非即满足要求。

20.float\_i2f





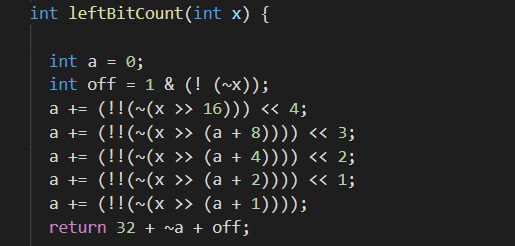
将整形int转化为浮点数float。如果x为0可以直接返回0，x不为0时，先取符号位sign，并将剩余部分取为正数形式absx，得到无符号的数值。

再将有数字部分依次左移，将第一个有效数字1移动到最高位，并记录移动的位数。对于阶码部分，由于move记录的是小数点从31位右数到第一个1，但实际上需要处理的是从第0位到第一位，所以exp=127+32-（move+1）=158-move

由于尾数只用23位，int转化时会截断，产生误差，需要进行舍入。所以用变量tail存储被截断的部分，并通过它的大小和尾数的最低位决定舍入。

结果result等于符号位、阶码部分、尾数部分取或，再加上舍入部分即可。

21.leftBitCount



如果x右移n位后得到0xffffffff，那么x从第n位到最高位上的数字应该全为1。

采用二分法，分别依次检验x的前16位是否全部为1，若全为1则将后16位再次二分检验前8位是否全为1；否则将前16位再次二分检验前8位是否全为1，依此类推，最终得到的a的值等于31减连续的1的数量，所以31-a即为所求结果。

但此时还无法区分0xffffffff和0xfffffffe两种情况，所以构造变量off，仅当x为~0时它的值为1，从而进行区分。