第二次作业

张运吉 211300063

13.

2： *x = a, *y = a ^ b;

3: *x = a ^ (a ^ b) = b, *y = a ^ b;

4: *x = b, * y = b ^ (a ^ b) = a;

14.

(1). 当len为奇数的时候，最后一次for循环中，向xor_swap传递的是同一个地址，执行完xor_swap后，*x = *y = 0， 可能会改变原来的数值；

(2). left = right = (len - 1) / 2;

(3). xor_swap没有返回值，但是调用后 *x = *y = 0；

(4). 把for循环的结束条件改为 left < right。

15.

x	y	x^y	x&y	x|y	~x|~y	x&!y	x&&y	x||y	!x||!y	x&&~y
0x5F	0xA0	0xFF	0x00	0xFF	0xFF	0x00	0x01	0x01	0x00	0x01
0xC7	0xF0	0x37	0xC0	0xF7	0x3F	0x00	0x01	0x01	0x00	0x01
0x80	0x7F	0xFF	0x00	0xFF	0xFF	0x00	0x01	0x01	0x00	0x01
0x07	0x55	0x52	0x05	0x57	0xFA	0x00	0x01	0x01	0x00	0x01

16.

(1). x >> (n - 8) << (n - 8)

(2). (unsigned)x << (n - 4) >> (n - 4)

(3). (~x) >> 8 << 8

(4). x | 0xFF

17.

b8010000 = 440

14 = 20

58feffff = -424

74feffff = -396

44 = 68

c8feffff = -312

10 = 16

0c = 12

ecfeffff = -276

20 = 32

18.

因为strlen函数的返回值是unsigned int类型，两个unsigned int类型的数相减得到的还是unsigned int类型，其值总是大于0，当strlen(str1) < strlen(str2) 时，返回值为1， 错误。 只要把代码第3行修改为return strlen(str1) > strlen(str2)即可。

21.

M = 15, N = 4。

30.

```
int ch_mul_overflow(int x, int y) {
    long long ans = (long long)x * y;
    return ans != (int)ans;
}
```

判断规则：若ans的高32位每一位都与低32位的最高位相同，则不溢出，否则一处。
函数的第一行把x强制类型转化为long long类型后再相乘，得到的是64位的乘积(准确)，第二行的(int)ans把64位截断成32位，因为要进行!=判断，所以又会符号拓展为64位，如果没有溢出，则ans != (int)ans的结果时false，否则时true。

31.

不能，上述改动虽然能使arraysize的值准确，但若arraysize的值大于unsigned int能表示的最大值时，当调用malloc函数的时候，还是只能按照32位去申请空间，还是会有整数溢出漏洞，修改方案如下：先判断arraysize的值是不是大于unsigned int能表示的最大值，若是则返回-1表示不成功，若否再申请空间复制数组。

```
int copy_array(int *array, int count) {
    int i;
    unsigned long long arraysize = count * (unsigned long long)sizeof(int);
    if (arraysize != (size_t)arraysize)
        return -1;
    if (myarray == NULL)
        return -1;
    for (i = 0; i < count; i++) {
        myarray[i] = array[i];
    }
    return count;
}
```

34.

(1). (x * x) >= 0 非永真，反例：当$x = 2^{16} - 1$时，$x * x = 2^{32} - 2^{18} + 4(mod 2^{32}) < 0$

(2). x - 1 < 0 || x > 0 非永真，反例：当x = 80000000H(机器数)时， x < 0, x - 1 = 7FFFFFFFH > 0, x - 1 < 0 || x > 0为假。

(3). x < 0 || -x <= 0 永真，x < 0，式子为真，x = 0，-x = 0，式子为真，x > 0, x的最高位为0，并且0到31位不是全0，则-x的机器数最高位是1，-x < 0, 式子为真，综上，上式永真。

(4). x > 0 || -x >= 0 非永真，反例，当x的机器数位0x80000000时，x < 0, -x的机器数为0x80000000，式子不成立。

(5). x&0xf!=15 || (x << 28) < 0 非永真，因为!=的优先级比&优先级高，所以x = 0时，上式为假。

(6). x > y == (-x < -y) 非永真，如(4)中的例子，若x = 0x80000000，则x和-x都小于0，显然上式不成立。

(7). ~x + ~y == ~(x + y) 永假，$[-x]_{补} = \sim[x]_{补} + 1, [-y]_{补} = \sim[y]_{补} + 1, [-(x + y)]_{补} = \sim[(x + y)]_{补} + 1$, 所以左边 = $\sim[x]_{补} + \sim[y]_{补} = [-x]_{补} + [-y]_{补} - 2$, 右边 = $\sim[x + y]_{补} =$

$[-(x+y)]_{补}-1=[-x]_{补}+[-y]_{补}-1$，可知，右边比左边大1。

(8). (int)(ux - uy) == -(y - x) 永真，因为ux-uy和x-y的机器数相同，把ux-uy按照int类型解释，其值和x-y=-(y-x)相同。

(9). ((x >> 2) << 2) <= x 永真，因为右移是向负无穷方向取整。

(10). x * 4 + y * 8 == (x << 2) + (y << 3) 永真，带符号数x乘以2^k等于x左移k位。

(11). x / 4 + y / 8 == (x >> 2) + (y >> 3) 非永真，当x或y为负数且不能被x不能被4整除或y不能被8整除是，上式不成立(要进行校正才能得到正确结果)。

(12). x * y == ux ** uy 永真，x * y 和 ux ** uy的低32位是完全一样的。

(13). x + y == ux + uy 永真，因为x + y 和 ux + uy机器数是完全一样的。

(14). x * ~y + ux * uy == -x 永真，因为 x * ~y + ux * uy = x * (-y - 1) + x * y = -x。

35.

(1). dx * dy >= 0 永真， 因为浮点数乘法数值和符号分开计算，不论结果是否溢出都不影响符号位。

(2). (double)(float)x == dx 非永真，因为int类型有31位有效位，而float只有23位有效位，可能会在转化的时候造成精度损失，而int转化位double是不会损失精度，所以上式不一定成立。

(3). dx + dy == (double)(x + y) 非永真，因为x + y可能会溢出，但是dx + dy则不会。

(4).(dx + dy) + dz == dx + (dy + dz) 永真，double类型可以精确表示int类型，并且上式的计算结果不会超过double可表示的范围，不会舍入，所以上式永真。

(5). dx * dy * dz == dz * dy * dz 非永真，当发生溢出时，会有舍入的情况。

(5). dx / dx == dy / dy 非永真，当x和y其中一个为0时，上式不成立。