2.58：

```
Int is_little_endian(){
    Int x = 1;                   //最低字节为 1
    Return *((char*)&x);         //取低址字节，小端模式为 1 大端模式为 0
}
```

2.61:

```
A. !~x                           //所有位为 1
B. !x                            //所有位为 0
C. !~(x | ~0xff)                 //最低有效字节所有位为 1
D. !( (x>> ((sizeof(int)-1)<<3)) & 0xff)   //最高有效字节所有位为 0
```

2.77:

```
A. K=17: (x<<4) + x              //2^4+1=17
B. K=-7: x-(x<<3)                //1-2^3=-7
C. K=60: (x<<6)-(x<<2)           //2^6-2^2=60
D. K=-112: (x<<4)-(x<<7)         //2^4-2^7=-112
```

2.84:

```
((ux<<1)==0 && (uy<<1)==0)||     //同为 0，符合题意
(sx && !sy)||                    //x 为负，y 为正，符合题意
(!sx && !sy && ux <= uy)||       //同为正数，绝对值大的大
(sx && sy && ux >= uy)           //同为负数，绝对值大的小
```

2.89:

A. ture， 先强制转换 double 并不影响再强制转换 float

B. false， 当 x-y 越界时,左边 double 不会越界，而右边 int 会越界。

C. true,double 可以精确表示所有正负 2^53 以内的所有整数。所以三个数相加可以精确表示

D. false， double 无法精确表示 2^64 以内所有的数,该表达式很有可能不会相等。反例可以考虑比较大的值。

E. false， 0/0.0 为 NaN， (非 0)/0.0 为正负 inf。同号 inf 相减为 NaN,异号 inf 相减也为被减数的 inf。故该表达式可能不会相等。比如 0/0.0!=1/1.0

2.91:

A.pi 的二进制数表示为: 0 10000000 10010010000111111101011， E=128-127=1，它表示的二进制小数值为: 11 .0010010000111111101011

B.根据 2.83，可知 1/7 的表示为 0.001001[0011....

所以 22/7 为 11.0010010010010011[0011...

C.从第 9 位开始不同。