**2.61 写一个C表达式，在下列描述的条件下产生1，而在其他情况下得到0。假设x是int类型。

A. x的任何位都等于1。

B. x的任何位都等于0。

C. x的最高有效字节中的位都等于1。

D. x的最低有效字节中的位都等于0。

代码应该遵循位级整数编码规则，另外还有一个限制，你不能使用相等（==）和不相等（!=）测试。

答：

①(!~x)||(!x)||(!~(x|0x00ffffff)||(!(x&0x000000ff))

(任何位都为1)||(任何位都为0)||(掩码计算，最高有效字节位都为1)||(掩码计算，最低有效位字节都为0)

②(!~x)||(!x)||(!~(x>>24))||(!(x<<24))

(任何位都为1)||(任何位都为0)||(移位计算，最高有效字节位都为1，逻辑右移，补1)||(移位计算，最低有效字节为都为0，补0)

思路是令正确的条件表达式为1，四个表达式相与。而逻辑运算中0的返回值为0，非零的返回值为1，为了令唯一的值为1，我们让正确的值得到结果0，再取反即可。

*2.71 你刚刚开始在一家公司工作，他们要实现一组过程来操作一个数据结构，要将4个有符号字节封装成一个32位unsigned。一个字中的字节从0（最低有效字节）编号到3（最高有效字节）。分配给你的任务是：为一个使用补码运算和算术右移的机器编写一个具有如下原型的函数：

```
/* Declaration of data type where 4 bytes are packed
   into an unsigned */
typedef unsigned packed_t;

/* Extract byte from word.  Return as signed integer */
int xbyte(packed_t word, int bytenum);
```

也就是说，函数会抽取出指定的字节，再把它符号扩展为一个32位int。

你的前任（因为水平不够高而被解雇了）编写了下面的代码：

```
/* Failed attempt at xbyte */
int xbyte(packed_t word, int bytenum)
{
    return (word >> (bytenum << 3)) & 0xFF;
}
```

A. 这段代码错在哪里？

B. 给出函数的正确实现，只能使用左右移位和一个减法。

答：

A.word是一个无符号数，只能使用逻辑右移，扩展时前面的位数补0，不符合题目要求。

B.packed_t word是一个无符号数，bytenum是指定被扩展的那个字节编号，题目想要抽出字节编号指定的字节并根据它的符号扩展为一个32位的int型整数。而我们只能使用左右移位和一个减法操作，所以可以先将要提取的字节左移到最高位，转化为int之后进行默认的算术右移，再移至最低位即可。

```
int xbyte(packed_t word,int bytenum)
{
   return ((int)word<<((3-bytenum)<<3))>>24;}
```

**2.87 考虑下面两个基于 IEEE 浮点格式的 9 位浮点表示。

1. 格式 A
- 有一个符号位。
- 有 $k=5$ 个阶码位。阶码偏置量是 15。
- 有 $n=3$ 个小数位。

2. 格式 B
- 有一个符号位。
- 有 $k=4$ 个阶码位。阶码偏置量是 7。
- 有 $n=4$ 个小数位。

下面给出了一些格式 A 表示的位模式，你的任务是把它们转换成最接近的格式 B 表示的值。如果需要舍入，你要向 $+\infty$ 舍入。另外，给出用格式 A 和格式 B 表示的位模式对应的值。要么是整数（例如，17），要么是小数（例如，17/64 或 $17/2^6$）。

格式A		格式B	
位	值	位	值
1 01110 001	$\frac{-9}{16}$	1 0110 0010	$\frac{-9}{16}$
0 10110 101			
1 00111 110			
0 00000 101			
1 11011 000			
0 11000 100			

格式 A		格式 B	
位	值	位	值
1 01110 001	-9/16	1 0110 0010	-9/16
0 10110 101	208	0 1110 1010	208
1 00111 110	-7*2^-10	1 0000 0111	-7*2^-10
0 00000 101	5*2^-17	0 0000 0001	1*2^-10
1 11011 000	-2^12	1 1111 0000	-inf
0 11000 100	3*2^8	0 1111 0000	+inf

格式 B 的范围小于格式 A，是 2^-6 ~ 2^7.

*2.88 我们在一个 int 类型为 32 位补码表示的机器上运行程序。float 类型的值使用 32 位 IEEE 格式，而 double 类型的值使用 64 位 IEEE 格式。

我们产生随机整数 x、y 和 z，并且把它们转换成 double 类型的值：

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to double */
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

对于下列的每个 C 表达式，你要指出表达式是否总是为 1。如果它总是为 1，描述其中的数学原理。否则，列举出使它为 0 的参数的例子。请注意，不能使用 IA32 机器运行 GCC 来测试你的答案，因为对于 float 和 double，它使用的都是 80 位的扩展精度表示。

A. `(double)(float) x == dx`

B. `dx + dy == (double) (x+y)`

C. `dx + dy + dz == dz + dy + dx`

D. `dx * dy * dz == dz * dy * dx`

E. `dx / dx == dy / dy`

A. 不总是为 1。

比较==时由于 float 的精度不如 double，可能会有 float 无法精确表达而 double 可以精确表达的情况，可能等式不成立。并且 float 的表示范围也小于 double。取 x=67,108,863，其二进制尾数有 25 个 1，其阶码 E=1048=1023+25，尾数 M=1.11……(小数点后 25 个 1)。进行 32 位表示时由于 32 位的尾数只有 23 位，无法精确表示，导致不相等。

B. 不总是为 1

整数加法是精确的，但当转换为 double 类型时，由于浮点数的表示限制（不能精确表示每个整数），可能产生舍入误差。当 x 和 y 的和非常大或非常接近 int 类型的边界时，这种舍入误差可能导致 dx + dy 和 (double)(x+y) 不相等。而溢出问题也会导致两边不成立。

取 x=y=(1.11……1)*2^1023，即所能表示最大的数，此时右边溢出，等式不成立。

C. 总是为 1

加法满足交换律，即使溢出，溢出值也相同。

D. 不总是为 1

三个浮点数的乘法运算不可交换，由于可能发生溢出，或者由于舍入而失去精度。如，le20*le20*le-20 的值为正无穷，而 le-20*le20*le20 的值为 le20。

E. 不总是为 1

浮点除法中规定除 0 将得到无穷，取 x=0,y≠0，两边将不相等。