•• 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 的最低有效字节中的位都等于 1。

D. x 的最高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。

A !~x

B !x

C !~(x|~0xff)

D !(x>>(sizeof(int)-1)<<3))

• 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。

A. 这段代码最后返回的是无符号数

B.
```
int xbyte(packet_t word, int bytenum)
{
    int ret= word<<((3-bytenum)<<3);
    return ret>>24;
}
```

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	208	01110 1010	208
1 00111 110	-7/1024	1 0000 0111	-7/1024
0 00000 101	$5/2^{17}$	0 00000000	0
1 11011 000	-4096	1 1111 0000	-inf
0 11000 100	768	0 1111 0000	inf

*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. 不总为 1，当 x 大于等于 2^{24}-1,即 16777217 时，有可能出现这种情况。因为 float 型不能精确表示所有整数，但是 double 可以，当 int 型过大时，转换为 float 就会丢失精度

B. 不总为 1，右侧的 x+y 会溢出，转换为 double 后为溢出后的值，但是左侧的 dx+dy 因为是 double 型所以不会发生溢出，所以仍然是真实值。

C. 总为 1，double 型可以精确表示所有 int 型整数

D. 不总为 1，当 xyz 都特别大时，double 可能无法精确表示其结果，导致交换律不成立

```
root@VM-8-13-ubuntu:~/test# ./work
186302039824879493869836697 6
186302039824879466382046003 2
root@VM-8-13-ubuntu:~/test# cat work.c
#include<stdio.h>

int main()
{
        int x=1111111112;
        int y=1231234425;
        int z=5656786234;
        double dx =(double)x;
        double dy =(double)y;
        double dz =(double)z;
        double r1 =dx*dy*dz;
        double  r2=dz*dy*dx;
        printf("%.f\n",r1);
        printf("%.f\n",r2);
        return 0;

}
```

E. 不总为 1，当 x=0，y!=0 时

```
root@VM-8-13-ubuntu:~/test# ./work
-nan
1
```

```
#include<stdio.h>

int main()
{
        int x=0;
        int y=1;
        double dx =(double)x;
        double dy =(double)y;
        printf("%.f\n",dx/dx);
        printf("%.f\n",dy/dy);
        return 0;

}
```