作业 2

2.61、2.63、2.65、2.71、2.72、2.77、2.87、2.90

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)) & 0xff)

2.63 将下面的 C 函数代码补充完整。函数 srl 用算术右移(由值 xsra 给出)来完成逻辑右移，后面的其他操作不包括右移或者除法。函数 sra 用逻辑右移(由值 xsrl 给出)来完成算术右移，后面的其他操作不包括右移或者除法。可以通过计算 8*sizeof(int)来确定数据类型 int 中的位数 w。位移量 k 的取值范围为 $0 \sim w-1$。

```
unsigned srl(unsigned x, int k) {
    /* Perform shift arithmetically */
    unsigned xsra = (int) x >> k;
    .
    .
    .
    .
    .
}

int sra(int x, int k) {
    /* Perform shift logically */
    int xsrl = (unsigned) x >> k;
    .
    .
    .
    .
    .
}
```

```
unsigned srl(unsigned x, int k) {
    unsigned xsra = (int) x >> k;
    int temp = (int) -1 << ((8 * sizeof(int)) – k);
    return xsra & ~temp;
}

int sra(int x, int k) {
    int xsrl = (unsigned) x >> k;
    unsigned temp = 2 << ((8 * sizeof(int)) – k – 1) – 1;
    return xsra & temp;
}
```

2.65 写出代码实现如下函数：

```
/* Return 1 when x contains an odd number of 1s; 0 otherwise.
   Assume w=32 */
int odd_ones(unsigned x);
```

函数应该遵循位级整数编码规则，不过你可以假设数据类型 int 有 $w=32$ 位。你的代码最多只能包含 12 个算术运算、位运算和逻辑运算。

```
int odd_ones(unsigned x) {
    x ^= x >> 16;
    x ^= x >> 8;
    x ^= x >> 4;
    x ^= x >> 2;
    x ^= x >> 1;
    return (x & 0x1);
}
```

•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. 得到的结果是 unsigned，即该代码只能处理正数而无法正确扩展出负数。

B. 将原始数据转化为 int，待抽取字节先左移到最高字节再右移到最低字节。

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

• 2.72 给你一个任务，写一个函数，将整数 val 复制到缓冲区 buf 中，但是只有当缓冲区中有足够可用的空间时，才执行复制。

你写的代码如下：

```
/* Copy integer into buffer if space is available */
/* WARNING: The following code is buggy */
void copy_int(int val, void *buf, int maxbytes) {
    if (maxbytes-sizeof(val) >= 0)
        memcpy(buf, (void *) &val, sizeof(val));
}
```

这段代码使用了库函数 memcpy。虽然在这里用这个函数有点刻意，因为我们只是想复制一个 int，但是它说明了一种复制较大数据结构的常见方法。

你仔细地测试了这段代码后发现，哪怕 maxbytes 很小的时候，它也能把值复制到缓冲区中。

A. 解释为什么代码中的条件测试总是成功。**提示**：sizeof 运算符返回类型为 size_t 的值。

B. 你该如何重写这个条件测试，使之工作正确。

A. sizeof 运算符返回的 size_t 是无符号整数，因此判断条件中左边被减数 maxbytes 也会转换为无符号整数进行运算，得到的差为无符号整数，大于等于 0 恒成立。

B. 判断条件改为 if(maxbytes > 0 && maxbytes >= sizeof(val))

•• 2.77 假设我们有一个任务：生成一段代码，将整数变量 x 乘以不同的常数因子 K。为了提高效率，我们想只使用＋、－和≪运算。对于下列 K 的值，写出执行乘法运算的 C 表达式，每个表达式中最多使用 3 个运算。

A. $K=17$

B. $K=-7$

C. $K=60$

D. $K=-112$

A. C = (x << 4) + x

B. C = x – (x << 3)

C. C = (x << 6) – (x << 2)

D. C = (x << 4) - (x << 7)

• 2.87 2008 版 IEEE 浮点标准，即 IEEE 754-2008，包含了一种 16 位的“半精度”浮点格式。它最初是由计算机图形公司设计的，其存储的数据所需的动态范围要高于 16 位整数可获得的范围。这种格式具有 1 个符号位、5 个阶码位($k=5$)和 10 个小数位($n=10$)。阶码偏置量是 $2^{5-1}-1=15$。

对于每个给定的数，填写下表，其中，每一列具有如下指示说明：

Hex：描述编码形式的 4 个十六进制数字。

M：尾数的值。这应该是一个形如 x 或 $\frac{x}{y}$ 的数，其中 x 是一个整数，而 y 是 2 的整数幂。例如：0、$\frac{67}{64}$ 和 $\frac{1}{256}$。

E：阶码的整数值。

V：所表示的数字值。使用 x 或者 $x\times 2^z$ 表示，其中 x 和 z 都是整数。

D：(可能近似的)数值，用 printf 的格式规范 %f 打印。

举一个例子，为了表示数 $\frac{7}{8}$，我们有 $s=0$，$M=\frac{7}{4}$ 和 $E=-1$。因此这个数的阶码字段为 01110_2(十进制值 $15-1=14$)，尾数字段为 1100000000_2，得到一个十六进制的表示 3B00。其数值为 0.875。

标记为“—”的条目不用填写。

描　述	Hex	M	E	V	D
−0	8000	0	-14	−0	−0.0
最小的>2 的值	4001	1025/1024	1	1025/512	2.00195312
512	6000	1	9	512	512.0
最大的非规格化数	03FF	1023/1024	-14	1023/2^24	6.09755516e-5
−∞	FC00	—	—	−∞	−∞
十六进制表示为 3BB0 的数	3BB0	123/64	-1	123/128	0.9609375

2.90 分配给你一个任务，编写一个 C 函数来计算 2^x 的浮点表示。你意识到完成这个任务的最好方法是直接创建结果的 IEEE 单精度表示。当 x 太小时，你的程序将返回 0.0。当 x 太大时，它会返回 $+\infty$。填写下列代码的空白部分，以计算出正确的结果。假设函数 u2f 返回的浮点值与它的无符号参数有相同的位表示。

```
float fpwr2(int x)
{
    /* Result exponent and fraction */
    unsigned exp, frac;
    unsigned u;

    if (x < -149) {
        /* Too small.  Return 0.0 */
        exp = 0;
        frac = 0;
    } else if (x < -126) {
        /* Denormalized result */
        exp = 0;
        frac = 1<<(x+149);

    } else if (x < 128) {
        /* Normalized result. */
        exp = x+127;
        frac = 0;
    } else {
        /* Too big.  Return +oo */
        exp = 255;
        frac = 0;
    }

    /* Pack exp and frac into 32 bits */
    u = exp << 23 | frac;
    /* Return as float */
    return u2f(u);
}
```

float 的 k = 8，n = 23，bias = 2^(k-1) - 1 = 127。
最小的正非规格化数为 2^(1-bias-n) = 2^-149。
最小的规格化数为 2^(0-bias)*2 = 2^-126。
最大的规格化数为 2^(2^k-2-bias) = 2^127。