•• 2.60 假设我们将一个 w 位的字中的字节从 0(最低位)到 $w/8-1$(最高位)编号。写出下面 C 函数的代码，它会返回一个无符号值，其中参数 x 的字节 i 被替换成字节 b：

```
unsigned replace_byte (unsigned x, int i, unsigned char b);
```

以下示例，说明了这个函数该如何工作：

```
replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678
replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
```

无符号数 unsigned 共有 4(0~3)字节

先将字节 i 抹去. unsigned mask = 0xffffffff - 0xff<<(i<<3);

unsigned result = x & mask;

// 其中 mask 会将第 i 字节留空，由于有 8 位，0xff 左移 i<<3=8i 位后

// 就得到期望的掩码，i=2 时，mask = 0xff00ffff

// 暂时 result = 0x__00____，横线上是原代码.

再得到对应 b: unsigned replace = b<<(i<<3);

// 同样地 replace = 0x00"b"0000

按位或得结果 result = result | replace;

return result;

•• 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。

注意到异或运算可产生奇校验和，将 x 不断二分取异或，最后只剩 1 位时可得.

```
int odd_ones(unsigned x) {
    x = x ^ (x >> 0x10); // 低16位.
    x = x ^ (x >> 0x8);  // 低8位.
    x = x ^ (x >> 0x4);  // 低4位
    x = x ^ (x >> 0x2);  // 低2位
    x = x ^ (x >> 0x1);  // 最后1位.
    return x & 0x1; // 得到答案，只取最低1位，共11个运算.
}
```

•• 2.67 给你一个任务，编写一个过程 int_size_is_32()，当在一个 int 是 32 位的机器上运行时，该程序产生 1，而其他情况则产生 0。不允许使用 sizeof 运算符。下面是开始时的尝试：

```
1   /* The following code does not run properly on some machines */
2   int bad_int_size_is_32() {
3       /* Set most significant bit (msb) of 32-bit machine */
4       int set_msb = 1 << 31;
5       /* Shift past msb of 32-bit word */
6       int beyond_msb = 1 << 32;
7
8       /* set_msb is nonzero when word size >= 32
9          beyond_msb is zero when word size <= 32  */
10      return set_msb && !beyond_msb;
11  }
```

10…00 (32 bits)

0…00 (32 bits)

当在 SUN SPARC 这样的 32 位机器上编译并运行时，这个过程返回的却是 0。下面的编译器信息给了我们一个问题的指示：

```
warning: left shift count >= width of type
```

A. 我们的代码在哪个方面没有遵守 C 语言标准？

B. 修改代码，使得它在 int 至少为 32 位的任何机器上都能正确地运行。

C. 修改代码，使得它在 int 至少为 16 位的任何机器上都能正确地运行。

A、由warning信息可知line 6中 int beyond_msb=1<<32;出错.
左移自身位数超过了int类型的最大长度.

B、最开始想到的版本如下(由于在机器上写好了就不抄一遍了):

```
23
24 int size_of_int_is_32() {
25   int set_msb = 1 << 31;
26   int shift_back = set_msb >> 31;
27
28   return shift_back == -1;
29 }
30
```

由算术右移的特性可知当 1<<31被送到最高位后再执行右移31位，得到的数是
0x11…1=-1，由此检测.

但下一问在至少16bit的机器上测，显然不太合适，还是需要解决A中的问题.
于是作出以下尝试.

```
3 /*
4 int bad_size_of_int() {
5   int set_msb = 1 << 31;
6   int beyond_msb = 1 << 16 << 16;
7
8   return set_msb && !beyond_msb;
9 }
10 still receives warning:
11 size_of_int.c:6:28: warning: result of ‘65536 << 16’ requires 34 bits to
   represent, but ‘int’ only has 32 bits [-Wshift-overflow=]
12     6 |   int beyond_msb = 1 << 16 << 16;
13       |                            ^~
14
15 but after modifying line 6 to the following:
16 int beyond_msb = 1 << 16;
17 beyond_msb = beyond_msb << 16;
18
19 it can be written as:
20 beyond_msb = set_msb << 1;
21 as well.
22 */
23
```

① 这样改了之后仍会报warning

② 改成这样不在同一句中左移总计32位，
编译器不再警告

③ 也可以简化一下，借用已经移好31位的set_msb.

这是不作出太多改动的版本.

```
/* alternative */
/*
int size_of_int_is_32() {
  int set_msb = 1 << 31;
  int beyond_msb = set_msb << 1;

  return set_msb && !beyond_msb;
}
*/
```

C, 如图,和上面相似,将31拆成15+15+1即可.

```
int size_of_int_is_32_for_16_bit_machine() {
  int set_msb = 1 << 15;
  set_msb = set_msb << 15;
  set_msb = set_msb << 1;
  int beyond_msb = set_msb << 1;

  return set_msb && !beyond_msb;
}
```

** 2.68 写出具有如下原型的函数的代码:

```
/*
 * Mask with least signficant n bits set to 1
 * Examples: n = 6 --> 0x3F, n = 17 --> 0x1FFFF
 * Assume 1 <= n <= w
 */
int lower_one_mask(int n);
```

函数应该遵循位级整数编码规则。要注意 n=w 的情况。

```
int lower_one_mask(int n) {
    int w = sizeof(int) << 3; // 得到int型变量位数
    int all_ones = -1;        // 得到全为1的一个变量.
    return (int)(((unsigned) all_ones) >> (w-n));
    // 将其转为无符号数,右移(w-n)位,则高位全为0,低n位为1.
    // 再转回int
}
```