



其对应的逻辑运算有相同的行为。

逻辑运算符 `&`、`&&` 和 `||` 与它们对应的位级运算 `&` 和 `|` 之间第二个重要的区别是，如果对第一个参数求值就能确定表达式的结果，那么逻辑运算符就不会对第二个参数求值。因此，例如，表达式 `a&&5/a` 将不会造成被零除，而表达式 `p&&*p++` 也不会导致间接引用空指针。

 **练习题 2.14** 假设 `x` 和 `y` 的字节值分别为 `0x66` 和 `0x39`。填写下表，指明各个 C 表达式的字节值。

表达式	值	表达式	值
<code>x &amp; y</code>		<code>x &amp;&amp; y</code>	
<code>x   y</code>		<code>x    y</code>	
<code>~x   ~y</code>		<code>!x    !y</code>	
<code>x &amp; !y</code>		<code>x &amp;&amp; ~y</code>	

 **练习题 2.15** 只使用位级和逻辑运算，编写一个 C 表达式，它等价于 `x==y`。换句话说，当 `x` 和 `y` 相等时它将返回 1，否则就返回 0。

## 2.1.9 C 语言中的移位运算

C 语言还提供了一组移位运算，向左或者向右移动位模式。对于一个位表示为  $[x_{w-1}, x_{w-2}, \dots, x_0]$  的操作数 `x`，C 表达式 `x<<k` 会生成一个值，其位表示为  $[x_{w-k-1}, x_{w-k-2}, \dots, x_0, 0, \dots, 0]$ 。也就是说，`x` 向左移动 `k` 位，丢弃最高的 `k` 位，并在右端补 `k` 个 0。移位量应该是一个  $0 \sim w-1$  之间的值。移位运算是从左至右可结合的，所以 `x<<j<<k` 等价于  $(x<<j)<<k$ 。

有一个相应的右移运算 `x>>k`，但是它的行为有点微妙。一般而言，机器支持两种形式的右移：逻辑右移和算术右移。逻辑右移在左端补 `k` 个 0，得到的结果是  $[0, \dots, 0, x_{w-1}, x_{w-2}, \dots, x_k]$ 。算术右移是在左端补 `k` 个最高有效位的值，得到的结果是  $[x_{w-1}, \dots, x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_k]$ 。这种做法看上去可能有点奇特，但是我们会发现它对有符号整数数据的运算非常有用。

让我们来看一个例子，下面的表给出了对一个 8 位参数 `x` 的两个不同的值做不同的移位操作得到的结果：

操作	值
参数 <code>x</code>	<code>[01100011] [10010101]</code>
<code>x &lt;&lt; 4</code>	<code>[00110000] [01010000]</code>
<code>x &gt;&gt; 4</code> (逻辑右移)	<code>[00000110] [00001001]</code>
<code>x &gt;&gt; 4</code> (算术右移)	<code>[00000110] [11111001]</code>

斜体的数字表示的是最右端(左移)或最左端(右移)填充的值。可以看到除了一个条目之外，其他的都包含填充 0。唯一的例外是算术右移 `[10010101]` 的情况。因为操作数的最高位是 1，填充的值就是 1。

C 语言标准并没有明确定义对于有符号数应该使用哪种类型的右移——算术右移或者逻辑右移都可以。不幸地，这就意味着任何假设一种或者另一种右移形式的代码都可能会遇到可移植性问题。然而，实际上，几乎所有的编译器/机器组合都对有符号数使用算术右移，且许多