Each of the ~, &, ^, and | operators has a different precedence:

Highest:	~
	&
	^
Lowest:	|

As a result, we can combine these operators in expressions without having to use parentheses. For example, we could write `i & ~j | k` instead of `(i & (~j)) | k` and `i ^ j & ~k` instead of `i ^ (j & (~k))`. Of course, it doesn't hurt to use parentheses to avoid confusion.

---

table of operators ➤Appendix A

The precedence of &, ^, and | is lower than the precedence of the relational and equality operators. Consequently, statements like the following one won't have the desired effect:

```
if (status & 0x4000 != 0) ...
```

Instead of testing whether `status & 0x4000` isn't zero, this statement will evaluate `0x4000 != 0` (which has the value 1), then test whether the value of `status & 1` isn't zero.

The compound assignment operators &=, ^=, and |= correspond to the bitwise operators &, ^, and |:

```
i = 21;    /* i is now 21 (binary 0000000000010101) */
j = 56;    /* j is now 56 (binary 0000000000111000) */
i &= j;    /* i is now 16 (binary 0000000000010000) */
i ^= j;    /* i is now 40 (binary 0000000000101000) */
i |= j;    /* i is now 56 (binary 0000000000111000) */
```

Using the Bitwise Operators to Access Bits

When we do low-level programming, we'll often need to store information as single bits or collections of bits. In graphics programming, for example, we may want to squeeze two or more pixels into a single byte. Using the bitwise operators, we can extract or modify data that's stored in a small number of bits.

Let's assume that `i` is a 16-bit `unsigned short` variable. Let's see how to perform the most common single-bit operations on `i`:

- *Setting a bit.* Suppose that we want to set bit 4 of `i`. (We'll assume that the leftmost—or ***most significant***—bit is numbered 15 and the least significant is numbered 0.) The easiest way to set bit 4 is to *or* the value of `i` with the constant `0x0010` (a "mask" that contains a 1 bit in position 4):

```
i = 0x0000;         /* i is now 0000000000000000 */
i |= 0x0010;        /* i is now 0000000000010000 */
```

More generally, if the position of the bit is stored in the variable `j`, we can use a shift operator to create the mask: