idiom

```
i |= 1 << j;          /* sets bit j */
```

For example, if j has the value 3, then 1 << j is 0x0008.

- *Clearing a bit.* To clear bit 4 of i, we'd use a mask with a 0 bit in position 4 and 1 bits everywhere else:

```
i = 0x00ff;           /* i is now 0000000011111111 */
i &= ~0x0010;         /* i is now 0000000011101111 */
```

Using the same idea, we can easily write a statement that clears a bit whose position is stored in a variable:

idiom

```
i &= ~(1 << j);       /* clears bit j */
```

- *Testing a bit.* The following if statement tests whether bit 4 of i is set:

```
if (i & 0x0010) ...   /* tests bit 4 */
```

To test whether bit j is set, we'd use the following statement:

idiom

```
if (i & 1 << j) ...   /* tests bit j */
```

To make working with bits easier, we'll often give them names. For example, suppose that we want bits 0, 1, and 2 of a number to correspond to the colors blue, green, and red, respectively. First, we define names that represent the three bit positions:

```
#define BLUE  1
#define GREEN 2
#define RED   4
```

Setting, clearing, and testing the BLUE bit would be done as follows:

```
i |= BLUE;          /* sets BLUE bit   */
i &= ~BLUE;         /* clears BLUE bit */
if (i & BLUE) ...   /* tests BLUE bit  */
```

It's also easy to set, clear, or test several bits at time:

```
i |= BLUE | GREEN;              /* sets BLUE and GREEN bits   */
i &= ~(BLUE | GREEN);           /* clears BLUE and GREEN bits */
if (i & (BLUE | GREEN)) ...     /* tests BLUE and GREEN bits  */
```

The if statement tests whether either the BLUE bit *or* the GREEN bit is set.

Using the Bitwise Operators to Access Bit-Fields

Dealing with a group of several consecutive bits (a ***bit-field***) is slightly more complicated than working with single bits. Here are examples of the two most common bit-field operations:

- *Modifying a bit-field.* Modifying a bit-field requires a bitwise *and* (to clear the bit-field), followed by a bitwise *or* (to store new bits in the bit-field). The following statement shows how we might store the binary value 101 in bits 4–6 of the variable i: