Side Effects

We don't normally expect operators to modify their operands, since operators in mathematics don't. Writing `i + j` doesn't modify either `i` or `j`; it simply computes the result of adding `i` and `j`.

Most C operators don't modify their operands, but some do. We say that these operators have ***side effects***, since they do more than just compute a value. The simple assignment operator is the first operator we've seen that has side effects; it modifies its left operand. Evaluating the expression `i = 0` produces the result 0 and—as a side effect—assigns 0 to `i`.

Since assignment is an operator, several assignments can be chained together:

```
i = j = k = 0;
```

The = operator is right associative, so this assignment is equivalent to

```
i = (j = (k = 0));
```

The effect is to assign 0 first to `k`, then to `j`, and finally to `i`.

Watch out for unexpected results in chained assignments as a result of type conversion:

```
int i;
float f;

f = i = 33.3f;
```

`i` is assigned the value 33, then `f` is assigned 33.0 (not 33.3, as you might think).

In general, an assignment of the form *v* = *e* is allowed wherever a value of type *v* would be permitted. In the following example, the expression `j = i` copies `i` to `j`; the new value of `j` is then added to `1`, producing the new value of `k`:

```
i = 1;
k = 1 + (j = i);
printf("%d %d %d\n", i, j, k);   /* prints "1 1 2" */
```

Using the assignment operator in this fashion usually isn't a good idea. For one thing, "embedded assignments" can make programs hard to read. They can also be a source of subtle bugs, as we'll see in Section 4.4.

Lvalues

Q&A

Most C operators allow their operands to be variables, constants, or expressions containing other operators. The assignment operator, however, requires an ***lvalue***