Not all values listed in a designated initializer need be prefixed by a designator. (This is true for arrays as well, as we saw in Section 8.1.) Consider the following example:

```
{.number = 528, "Disk drive", .on_hand = 10}
```

The value `"Disk drive"` doesn't have a designator, so the compiler assumes that it initializes the member that follows `number` in the structure. Any members that the initializer fails to account for are set to zero.

Operations on Structures

Since the most common array operation is subscripting—selecting an element by position—it's not surprising that the most common operation on a structure is selecting one of its members. Structure members are accessed by name, though, not by position.

To access a member within a structure, we write the name of the structure first, then a period, then the name of the member. For example, the following statements will display the values of `part1`'s members:

```
printf("Part number: %d\n", part1.number);
printf("Part name: %s\n", part1.name);
printf("Quantity on hand: %d\n", part1.on_hand);
```

lvalues ➤*4.2*

The members of a structure are lvalues, so they can appear on the left side of an assignment or as the operand in an increment or decrement expression:

```
part1.number = 258;          /* changes part1's part number */
part1.on_hand++;    /* increments part1's quantity on hand */
```

table of operators ➤*Appendix A*

The period that we use to access a structure member is actually a C operator. It has the same precedence as the postfix ++ and -- operators, so it takes precedence over nearly all other operators. Consider the following example:

```
scanf("%d", &part1.on_hand);
```

The expression `&part1.on_hand` contains two operators (`&` and `.`). The `.` operator takes precedence over the `&` operator, so `&` computes the address of `part1.on_hand`, as we wished.

The other major structure operation is assignment:

```
part2 = part1;
```

The effect of this statement is to copy `part1.number` into `part2.number`, `part1.name` into `part2.name`, and so on.

Since arrays can't be copied using the = operator, it comes as something of a surprise to discover that structures can. It's even more surprising when you consider that an array embedded within a structure is copied when the enclosing structure is copied. Some programmers exploit this property by creating "dummy" structures to enclose arrays that will be copied later: