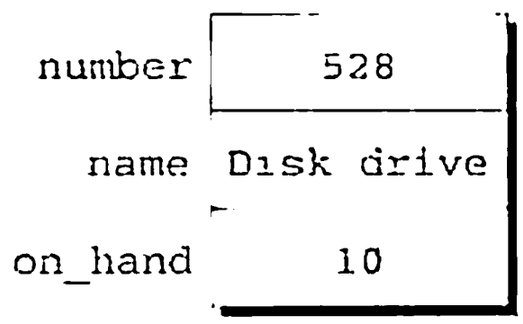
The values in the initializer must appear in the same order as the members of the structure. In our example, the `number` member of `part1` will be 528, the `name` member will be `"Disk drive"`, and so on. Here's how `part1` will look after initialization:



Structure initializers follow rules similar to those for array initializers. Expressions used in a structure initializer must be constant; for example, we couldn't have used a variable to initialize `part1`'s `on_hand` member. (This restriction is relaxed in C99, as we'll see in Section 18.5.) An initializer can have fewer members than the structure it's initializing; as with arrays, any "leftover" members are given 0 as their initial value. In particular, the bytes in a leftover character array will be zero, making it represent the empty string.

C99

Designated Initializers

C99's designated initializers, which were discussed in Section 8.1 in the context of arrays, can also be used with structures. Consider the initializer for `part1` shown in the previous example:

```
{528, "Disk drive", 10}
```

A designated initializer would look similar, but with each value labeled by the name of the member that it initializes:

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

The combination of the period and the member name is called a *designator*. (Designators for array elements have a different form.)

Designated initializers have several advantages. For one, they're easier to read and check for correctness, because the reader can clearly see the correspondence between the members of the structure and the values listed in the initializer. Another is that the values in the initializer don't have to be placed in the same order that the members are listed in the structure. Our example initializer could be written as follows:

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

Since the order doesn't matter, the programmer doesn't have to remember the order in which the members were originally declared. Moreover, the order of the members can be changed in the future without affecting designated initializers.