```
struct { int a[10]; } a1, a2;

a1 = a2;   /* legal, since a1 and a2 are structures */
```

The = operator can be used only with structures of ***compatible*** types. Two structures declared at the same time (as `part1` and `part2` were) are compatible. As we'll see in the next section, structures declared using the same "structure tag" or the same type name are also compatible.

Other than assignment, C provides no operations on entire structures. In particular, we can't use the == and != operators to test whether two structures are equal or not equal.

Q&A

16.2 Structure Types

Although the previous section showed how to declare structure *variables*, it failed to discuss an important issue: naming structure *types*. Suppose that a program needs to declare several structure variables with identical members. If all the variables can be declared at one time, there's no problem. But if we need to declare the variables at different points in the program, then life becomes more difficult. If we write

```
struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part1;
```

in one place and

```
struct {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} part2;
```

in another, we'll quickly run into problems. Repeating the structure information will bloat the program. Changing the program later will be risky, since we can't easily guarantee that the declarations will remain consistent.

But those aren't the biggest problems. According to the rules of C, `part1` and `part2` don't have compatible types. As a result, `part1` can't be assigned to `part2`, and vice versa. Also, since we don't have a name for the type of `part1` or `part2`, we can't use them as arguments in function calls.

To avoid these difficulties, we need to be able to define a name that represents a *type* of structure, not a particular structure *variable*. As it turns out, C provides two ways to name structures: we can either declare a "structure tag" or use `typedef` to define a type name.

Q&A

type definitions ➤7.5