Q: How can I share a structure type among several files in a program?

A: Put a declaration of the structure tag (or a `typedef`, if you prefer) in a header file, then include the header file where the structure is needed. To share the `part` structure, for example, we'd put the following lines in a header file:

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

Notice that we're declaring only the structure *tag*, not variables of this type.

Incidentally, a header file that contains a declaration of a structure tag or structure type may need protection against multiple inclusion. Declaring a tag or `typedef` name twice in the same file is an error. Similar remarks apply to unions and enumerations.

protecting header files ➤ 15.2

Q: If I include the declaration of the `part` structure into two different files, will `part` variables in one file be of the same type as `part` variables in the other file?

A: Technically, no. However, the C standard says that the `part` variables in one file have a type that's compatible with the type of the `part` variables in the other file. Variables with compatible types can be assigned to each other, so there's little practical difference between types being "compatible" and being "the same."

C99 The rules for structure compatibility in C89 and C99 are slightly different. In C89, structures defined in different files are compatible if their members have the same names and appear in the same order, with corresponding members having compatible types. C99 goes one step further: it requires that either both structures have the same tag or neither has a tag.

Similar compatibility rules apply to unions and enumerations (with the same difference between C89 and C99).

Q: Is it legal to have a pointer to a compound literal?

A: Yes. Consider the `print_part` function of Section 16.2. Currently, the parameter to this function is a `part` structure. The function would be more efficient if it were modified to accept a *pointer* to a `part` structure instead. Using the function to print a compound literal would then be done by prefixing the argument with the `&` (address) operator:

```
print_part(&(struct part) {528, "Disk drive", 10});
```

Q: Allowing a pointer to a compound literal would seem to make it possible to
C99 **modify the literal. Is that the case?**

A: Yes. Compound literals are lvalues that can be modified, although doing so is rare.

Q: I saw a program in which the last constant in an enumeration was followed by a comma, like this: