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

Here's how `print_part` might be called:

```
print_part(part1);
```

Our second function returns a `part` structure that it constructs from its arguments:

```
struct part build_part(int number, const char *name,
                       int on_hand)
{
  struct part p;

  p.number = number;
  strcpy(p.name, name);
  p.on_hand = on_hand;
  return p;
}
```

Notice that it's legal for `build_part`'s parameters to have names that match the members of the `part` structure, since the structure has its own name space. Here's how `build_part` might be called:

```
part1 = build_part(528, "Disk drive", 10);
```

Passing a structure to a function and returning a structure from a function both require making a copy of all members in the structure. As a result, these operations impose a fair amount of overhead on a program, especially if the structure is large. To avoid this overhead, it's sometimes advisable to pass a *pointer* to a structure instead of passing the structure itself. Similarly, we might have a function return a pointer to a structure instead of returning an actual structure. Section 17.5 gives examples of functions that have a pointer to a structure as an argument and/or return a pointer to a structure.

There are other reasons to avoid copying structures besides efficiency. For example, the `<stdio.h>` header defines a type named `FILE`, which is typically a structure. Each `FILE` structure stores information about the state of an open file and therefore must be unique in a program. Every function in `<stdio.h>` that opens a file returns a pointer to a `FILE` structure, and every function that performs an operation on an open file requires a `FILE` pointer as an argument.

FILE type ➤22.1

On occasion, we may want to initialize a structure variable inside a function to match another structure, possibly supplied as a parameter to the function. In the following example, the initializer for `part2` is the parameter passed to the `f` function:

```
void f(struct part part1)
{
  struct part part2 = part1;
  ...
}
```