In the s structure, i and d occupy different memory locations: the total size of s is 12 bytes. In the u union, i and d overlap (i is really the first four bytes of d), so u occupies only eight bytes. Also, i and d have the same address.

Members of a union are accessed in the same way as members of a structure. To store the number 82 in the i member of u. we would write

```
u.i = 82;
```

To store the value 74.8 in the d member, we would write

```
u.d = 74.8;
```

Since the compiler overlays storage for the members of a union, changing one member alters any value previously stored in any of the other members. Thus, if we store a value in u.d, any value previously stored in u.i will be lost. (If we examine the value of u.i, it will appear to be meaningless.) Similarly, changing u.i corrupts u.d. Because of this property, we can think of u as a place to store either i *or* d. not both. (The structure s allows us to store i *and* d.)

The properties of unions are almost identical to the properties of structures. We can declare union tags and union types in the same way we declare structure tags and types. Like structures, unions can be copied using the = operator, passed to functions, and returned by functions.

Unions can even be initialized in a manner similar to structures. However, only the first member of a union can be given an initial value. For example, we can initialize the i member of u to 0 in the following way:

```
union {
  int i;
  double d;
} u = {0};
```

Notice the presence of the braces, which are required. The expression inside the braces must be constant. (The rules are slightly different in C99. as we'll see in Section 18.5.)

C99
Designated initializers, a C99 feature that we've previously discussed in the context of arrays and structures. can also be used with unions. A designated initializer allows us to specify which member of a union should be initialized. For example, we can initialize the d member of u as follows:

```
union {
  int i;
  double d;
} u = {.d = 10.0};
```

Only one member can be initialized. but it doesn't have to be the first one.

There are several applications for unions. We'll discuss two of these now. Another application—viewing storage in different ways—is highly machine-dependent, so I'll postpone it until Section 20.3.