

Q: It seems like a designated initializer could end up initializing an array element more than once. Consider the following array declaration:

```
int a[] = {4, 9, 1, 8, [0] = 5, 7};
```

Is this declaration legal, and if so, what is the length of the array? [p. 166]

A: Yes, the declaration is legal. Here's how it works: as it processes an initializer list, the compiler keeps track of which array element is to be initialized next. Normally, the next element is the one following the element that was last initialized. However, when a designator appears in the list, it forces the next element be the one represented by the designator, *even if that element has already been initialized*.

Here's a step-by-step look at how the compiler will process the initializer for the array `a`:

The 4 initializes element 0; the next element to be initialized is element 1.

The 9 initializes element 1; the next element to be initialized is element 2.

The 1 initializes element 2; the next element to be initialized is element 3.

The 8 initializes element 3; the next element to be initialized is element 4.

The `[0]` designator causes the next element to become 0, so the 5 initializes element 0 (replacing the 4 previously stored there). The next element to be initialized is element 1.

The 7 initializes element 1 (replacing the 9 previously stored there). The next element to be initialized is element 2 (which is irrelevant since we're at the end of the list).

The net effect is the same as if we had written

```
int a[] = {5, 7, 1, 8};
```

Thus, the length of this array is four.

Q: The compiler gives me an error message if I try to copy one array into another by using the assignment operator. What's wrong?

A: Although it looks quite plausible, the assignment

```
a = b;    /* a and b are arrays */
```

is indeed illegal. The reason for its illegality isn't obvious; it has to do with the peculiar relationship between arrays and pointers in C, a topic we'll explore in Chapter 12.

The simplest way to copy one array into another is to use a loop that copies the elements, one by one:

```
for (i = 0; i < N; i++)
    a[i] = b[i];
```

`memcpy` function ► 23.6

Another possibility is to use the `memcpy` ("memory copy") function from the `<string.h>` header. `memcpy` is a low-level function that simply copies bytes from one place to another. To copy the array `b` into the array `a`, use `memcpy` as follows: