

Combining declarations in this way usually isn't a good idea, though; it can easily cause confusion.

Q: What happens if I specify a length for a one-dimensional array parameter? [p. 195]

A: The compiler ignores it. Consider the following example:

```
double inner_product(double v[3], double w[3]);
```

Other than documenting that `inner_product`'s arguments are supposed to be arrays of length 3, specifying a length doesn't buy us much. The compiler won't check that the arguments actually have length 3, so there's no added security. In fact, the practice is misleading in that it suggests that `inner_product` can only be passed arrays of length 3, when in fact we can pass arrays of arbitrary length.

***Q: Why can the first dimension in an array parameter be left unspecified, but not the other dimensions? [p. 197]**

A: First, we need to discuss how arrays are passed in C. As Section 12.3 explains, when an array is passed to a function, the function is given a *pointer* to the first element in the array.

Next, we need to know how the subscripting operator works. Suppose that `a` is a one-dimensional array passed to a function. When we write

```
a[i] = 0;
```

the compiler generates instructions that compute the address of `a[i]` by multiplying `i` by the size of an array element and adding the result to the address that `a` represents (the pointer passed to the function). This calculation doesn't depend on the length of `a`, which explains why we can omit it when defining the function.

What about multidimensional arrays? Recall that C stores arrays in row-major order, with the elements in row 0 stored first, then the elements in row 1, and so forth. Suppose that `a` is a two-dimensional array parameter and we write

```
a[i][j] = 0;
```

The compiler generates instructions to do the following: (1) multiply `i` by the size of a single row of `a`; (2) add this result to the address that `a` represents; (3) multiply `j` by the size of an array element; and (4) add this result to the address computed in step 2. To generate these instructions, the compiler must know the size of a row in the array, which is determined by the number of columns. The bottom line: the programmer must declare the number of columns in `a`.

Q: Why do some programmers put parentheses around the expression in a return statement?

A: The examples in the first edition of Kernighan and Ritchie's *The C Programming Language* always have parentheses in return statements, even though they aren't required. Programmers (and authors of subsequent books) picked up the habit from K&R. I don't use these parentheses, since they're unnecessary and