ber entered by the user determines the length of a; the programmer doesn't have to choose a fixed length, unlike in the original version of the program.

The length of a VLA doesn't have to be specified by a single variable. Arbitrary expressions, possibly containing operators, are also legal. For example:

```
int a[3*i+5];
int b[j+k];
```

Like other arrays, VLAs can be multidimensional:

```
int c[m][n];
```

static storage duration ➤*18.2*

The primary restriction on VLAs is that they can't have static storage duration. (We haven't yet seen any arrays with this property.) Another restriction is that a VLA may not have an initializer.

Variable-length arrays are most often seen in functions other than main. One big advantage of a VLA that belongs to a function f is that it can have a different length each time f is called. We'll explore this feature in Section 9.3.

Q & A

Q: **Why do array subscripts start at 0 instead of 1? [p. 162]**

A: Having subscripts begin at 0 simplifies the compiler a bit. Also, it can make array subscripting marginally faster.

Q: **What if I want an array with subscripts that go from 1 to 10 instead of 0 to 9?**

A: Here's a common trick: declare the array to have 11 elements instead of 10. The subscripts will go from 0 to 10, but you can just ignore element 0.

Q: **Is it possible to use a character as an array subscript?**

A: Yes, because C treats characters as integers. You'll probably need to "scale" the character before you use it as a subscript, though. Let's say that we want the letter_count array to keep track of a count for each letter in the alphabet. The array will need 26 elements, so we'd declare it in the following way:

```
int letter_count[26];
```

However, we can't use letters to subscript letter_count directly, because their integer values don't fall between 0 and 25. To scale a lower-case letter to the proper range, we can simply subtract 'a'; to scale an upper-case letter, we'll subtract 'A'. For example, if ch contains a lower-case letter, we'd write

```
letter_count[ch-'a'] = 0;
```

to clear the count that corresponds to ch. A minor caveat: this technique isn't completely portable, because it assumes that letters have consecutive codes. However, it works with most character sets, including ASCII.