```
struct vstring {
  int len;
  char chars[1];
};
...
struct vstring *str = malloc(sizeof(struct vstring) + n - 1);
str->len = n;
```

We're "cheating" by allocating more memory than the structure is declared to have (in this case, an extra n – 1 characters), and then using the memory to store additional elements of the chars array. This technique has become so common over the years that it has a name: the "struct hack."

The struct hack isn't limited to character arrays; it has a variety of uses. Over time, it has become popular enough to be supported by many compilers. Some (including GCC) even allow the chars array to have zero length, which makes this trick a little more explicit. Unfortunately, the C89 standard doesn't guarantee that the struct hack will work, nor does it allow zero-length arrays.

In recognition of the struct hack's usefulness, C99 has a feature known as the ***flexible array member*** that serves the same purpose. When the last member of a structure is an array, its length may be omitted:

```
struct vstring {
  int len;
  char chars[];    /* flexible array member - C99 only */
};
```

The length of the chars array isn't determined until memory is allocated for a vstring structure, normally using a call of malloc:

```
struct vstring *str = malloc(sizeof(struct vstring) + n);
str->len = n;
```

In this example, str points to a vstring structure in which the chars array occupies n characters. The sizeof operator ignores the chars member when computing the size of the structure. (A flexible array member is unusual in that it takes up no space within a structure.)

A few special rules apply to a structure that contains a flexible array member. The flexible array member must appear last in the structure, and the structure must have at least one other member. Copying a structure that contains a flexible array member will copy the other members but not the flexible array itself.

A structure that contains a flexible array member is an ***incomplete type***. An incomplete type is missing part of the information needed to determine how much memory it requires. Incomplete types, which are discussed further in one of the Q&A questions at the end of this chapter and in Section 19.3, are subject to various restrictions. In particular, an incomplete type (and hence a structure that contains a flexible array member) can't be a member of another structure or an element of an array. However, an array may contain pointers to structures that have a flexible array member; Programming Project 7 at the end of this chapter is built around such an array.