```
memmove(&a[0], &a[1], 99 * sizeof(int));
```

Prior to C99, there was no way to document the difference between memcpy and memmove. The prototypes for the two functions were nearly identical:

```
void *memcpy(void *s1, const void *s2, size_t n);
void *memmove(void *s1, const void *s2, size_t n);
```

The use of `restrict` in the C99 version of memcpy's prototype lets the programmer know that `s1` and `s2` should point to objects that don't overlap, or else the function isn't guaranteed to work.

Although using `restrict` in function prototypes is useful documentation, that's not the primary reason for its existence. `restrict` provides information to the compiler that may enable it to produce more efficient code—a process known as ***optimization***. (The `register` storage class serves the same purpose.) Not every compiler attempts to optimize programs, however, and the ones that do normally allow the programmer to disable optimization. As a result, the C99 standard guarantees that `restrict` has no effect on the behavior of a program that conforms to the standard: if all uses of `restrict` are removed from such a program, it should behave the same.

`register` storage class ➤ 18.2

Most programmers won't use `restrict` unless they're fine-tuning a program to achieve the best possible performance. Still, it's worth knowing about `restrict` because it appears in the C99 prototypes for a number of standard library functions.

17.9 Flexible Array Members (C99)

Every once in a while, we'll need to define a structure that contains an array of an unknown size. For example, we might want to store strings in a form that's different from the usual one. Normally, a string is an array of characters, with a null character marking the end. However, there are advantages to storing strings in other ways. One alternative is to store the length of the string along with the string's characters (but with no null character). The length and the characters could be stored in a structure such as this one:

```
struct vstring {
  int len;
  char chars[N];
};
```

Here `N` is a macro that represents the maximum length of a string. Using a fixed-length array such as this is undesirable, however, because it forces us to limit the length of the string, plus it wastes memory (since most strings won't need all `N` characters in the array).

C programmers have traditionally solved this problem by declaring the length of `chars` to be 1 (a dummy value) and then dynamically allocating each string: