`register` isn't nearly as popular among C programmers as it once was. Today's compilers are much more sophisticated than early C compilers; many can determine automatically which variables would benefit the most from being kept in registers. Still, using `register` provides useful information that can help the compiler optimize the performance of a program. In particular, the compiler knows that a `register` variable can't have its address taken, and therefore can't be modified through a pointer. In this respect, the `register` keyword is related to C99's `restrict` keyword.

The Storage Class of a Function

Function declarations (and definitions), like variable declarations, may include a storage class, but the only options are `extern` and `static`. The word `extern` at the beginning of a function declaration specifies that the function has external linkage, allowing it to be called from other files. `static` indicates internal linkage, limiting use of the function's name to the file in which it's defined. If no storage class is specified, the function is assumed to have external linkage.

Consider the following function declarations:

```
extern int f(int i);
static int g(int i);
int h(int i);
```

`f` has external linkage, `g` has internal linkage, and `h` (by default) has external linkage. Because it has internal linkage, `g` can't be called directly from outside the file in which it's defined. (Declaring `g` to be `static` doesn't completely prevent it from being called in another file; an indirect call via a function pointer is still possible.)

Declaring functions to be `extern` is like declaring variables to be `auto`—it serves no purpose. For that reason, I don't use `extern` in function declarations. Be aware, however, that some programmers use `extern` extensively, which certainly does no harm.

Declaring functions to be `static`, on the other hand, is quite useful. In fact, I recommend using `static` when declaring any function that isn't intended to be called from other files. The benefits of doing so include:

- *Easier maintenance.* Declaring a function `f` to be `static` guarantees that `f` isn't visible outside the file in which its definition appears. As a result, someone modifying the program later knows that changes to `f` won't affect functions in other files. (One exception: a function in another file that's passed a pointer to `f` might be affected by changes to `f`. Fortunately, that situation is easy to spot by examining the file in which `f` is defined, since the function that passes `f` must also be defined there.)
- *Reduced "name space pollution."* Since functions declared `static` have internal linkage, their names can be reused in other files. Although we proba-