The other option is to provide an external definition for `average` so that calls are permitted from other files. One way to do this is to write the `average` function a second time (without using `inline`) and put the second definition in a different source file. Doing so is legal, but it's not a good idea to have two versions of the same function, because we can't guarantee that they'll remain consistent when the program is modified.

Here's a better approach. First, we'll put the inline definition of `average` in a header file (let's name it `average.h`):

```
#ifndef AVERAGE_H
#define AVERAGE_H

inline double average(double a, double b)
{
  return (a + b) / 2;
}

#endif
```

Next, we'll create a matching source file, `average.c`:

```
#include "average.h"

extern double average(double a, double b);
```

Now, any file that needs to call the `average` function may simply include `average.h`, which contains the inline definition of `average`. The `average.c` file contains a prototype for `average` that uses the `extern` keyword, which causes the definition of `average` included from `average.h` to be treated as an external definition in `average.c`.

The general rule in C99 is that if all top-level declarations of a function in a particular file include `inline` but not `extern`, then the definition of the function in that file is inline. If the function is used anywhere in the program (including the file that contains its inline definition), then an external definition of the function will need to be provided by some other file. When the function is called, the compiler may choose to perform an ordinary call (using the function's external definition) or perform inline expansion (using the function's inline definition). There's no way to tell which choice the compiler will make, so it's crucial that the two definitions be consistent. The technique that we just discussed (using the `average.h` and `average.c` files) guarantees that the definitions are the same.

Restrictions on Inline Functions

Since inline functions are implemented in a way that's quite different from ordinary functions, they're subject to different rules and restrictions. Variables with static storage duration are a particular problem for inline functions with external linkage. Consequently, C99 imposes the following restrictions on an inline function with external linkage (but not on one with internal linkage):