amount of effort on both the part of the function that was called and the one that called it. The cumulative work required to call a function and later return from it is often referred to as "overhead," since it's extra work above and beyond what the function is really supposed to accomplish. Although the overhead of a function call slows the program by only a tiny amount, it may add up in certain situations, such as when a function is called millions or billions of times, when an older, slower processor is in use (as might be the case in an embedded system), or when a program has to meet very strict deadlines (as in a real-time system).

parameterized macros ➤ 14.3

In C89, the only way to avoid the overhead of a function call is to use a parameterized macro. Parameterized macros have certain drawbacks, though. C99 offers a better solution to this problem: create an *inline function*. The word "inline" suggests an implementation strategy in which the compiler replaces each call of the function by the machine instructions for the function. This technique avoids the usual overhead of a function call, although it may cause a minor increase in the size of the compiled program.

Declaring a function to be `inline` doesn't actually force the compiler to "inline" the function, however. It merely suggests that the compiler should try to make calls of the function as fast as possible, perhaps by performing an inline expansion when the function is called. The compiler is free to ignore this suggestion. In this respect, `inline` is similar to the `register` and `restrict` keywords, which the compiler may use to improve the performance of a program but may also choose to ignore.

Inline Definitions

An inline function has the keyword `inline` as one of its declaration specifiers:

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

Here's where things get a bit complicated. `average` has external linkage, so other source files may contain calls of `average`. However, the definition of `average` isn't considered to be an external definition by the compiler (it's an *inline definition* instead), so attempting to call `average` from another file will be considered an error.

There are two ways to avoid this error. One option is to add the word `static` to the function definition:

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

`average` now has internal linkage, so it can't be called from other files. Other files may contain their own definitions of `average`, which might be the same as this definition or might be different.