In general, you either can include or exclude variable names in the argument lists for prototypes. The variable names in the prototype just act as placeholders, so if you do use names, they don't have to match the names in the function definition.

**C++ Versus ANSI C Prototyping**

ANSI C borrowed prototyping from C++, but the two languages do have some differences. The most important is that ANSI C, to preserve compatibility with classic C, made prototyping optional, whereas C++ makes prototyping mandatory. For example, consider the following function declaration:

```c
void say_hi();
```

In C++, leaving the parentheses empty is the same as using the keyword `void` within the parentheses. It means the function has no arguments. In ANSI C, leaving the parentheses empty means that you are declining to state what the arguments are. That is, it means you're foregoing prototyping the argument list. The C++ equivalent for not identifying the argument list is to use ellipsis:

```c
void say_bye(...); // C++ abdication of responsibility
```

Normally this is needed only for interfacing with C functions having a variable number of arguments, such as `printf()`.

**What Prototypes Do for You**

You've seen that prototypes help the compiler. But what do they do for you? They greatly reduce the chances for program errors. In particular, prototypes ensure the following:

- The compiler correctly handles the function return value.
- The compiler checks that you use the correct number of function arguments.
- The compiler checks that you use the correct type of arguments. If not, it converts...