- If your source code has more than one function, the order in which they're listed is important; you cannot use a function inside your source code unless it has first been declared or prototyped. If you have multiple functions in your source code, order them so that if one function calls another, that second function is listed first. Otherwise, you're again saddled with prototyping errors.

The Tao of Functions

The C language allows you to put as many functions as you want in your source code. There really is no limit, though most programmers like to keep their source-code text files to a manageable size.

- What is "manageable size"? It depends.
- The larger the source code file, the longer it takes to compile.
- Often times, it pays to break off functions into their own, separate source code files. It not only aids in debugging, but also makes recompiling larger files easier.
- This book's companion volume, *C All-in-One Desk Reference For Dummies* (Wiley), contains information on creating and managing multimodule source code files.

The function format

Here's the format of a typical function:

```
type name(stuff)
```

The `type` tells the compiler whether the function returns a value. If the type is `void`, the function doesn't return any value. (It merely *functs*.) Otherwise, the type describes which type of value the function returns: `char`, `int`, `float`, or any of the standard C language variable declarations.

The *`name`* is the function's name. It must be a unique name, not any keywords or names of other C language library functions, such as `printf()` or `atio()`. (For more on names, see the next section.)

Parentheses after the function's name are required, as they are on all C language functions. The `stuff` inside the parentheses, if needed, defines whatever value (or values) are sent off to the function for evaluation, manipulation, or mutilation. I cover this subject in Chapter 22. If there's no *stuff,* the parentheses can be left empty or the word `void` can be used.