The statements belonging to the function require curly braces to hug them close. Those statements are the instructions that carry out what the function is supposed to do. Therefore, the full format for the function is shown here:

```
type name(stuff)
{
    statement(s);
    /* more statements */
}
```

The function must be prototyped before it can be used. You do that by either listing the full function earlier than it's first used in your source code or restating the function's declaration at the start of your source code. For example:

```
type name(stuff);
```

This line, with a semicolon, is required in order to prototype the function used later on in the program. It's just a copy-and-paste job, but the semicolon is required for the prototype. (If you forget, your compiler may ever so gently remind you with a barrage of error messages.)

- Call it *defining* a function. Call it *declaring* a function. Call it *doing* a function. (The official term is *defining* a function.)
- Naming rules for functions are covered in the next section.
- Your C language library reference lists functions by using the preceding format. For example:

  ```
  int atoi(const char *s);
  ```

  This format explains the requirements and product of the `atoi()` function. Its *type* is an `int`, and its *stuff* is a character string, which is how you translate `const char *s` into English. (Also noted in the format is that the `#include <stdlib.h>` thing is required at the beginning of your source code when you use the `atoi()` function.)

How to name your functions

Functions are like your children, so, for heaven's sake, don't give them a dorky name! You're free to give your functions just about any name, but keep in mind these notes:

- Functions are named by using letters of the alphabet and numbers. Almost all compilers insist that your functions begin with a letter. Check your compiler's documentation to see whether this issue is something your compiler is fussy about.