The opposite of a global variable is a *local variable.* It's what you have seen used elsewhere in this book. A local variable exists inside only one function — like the variable *x* in the BOMBER.C program. The *x* is a local variable, unique to the functions in which it's created and ignored by other functions in the program.

- A global variable is available to all functions in your program.
- A local variable is available only to the function in which it's created.
- Global variables can be used in any function without having to redeclare them. If you have a global integer variable `score`, for example, you don't have to stick an `int score;` declaration in each function which uses that variable. The variable has already been declared and is ready for use in any function.
- Because global variables exist all over the place, naming them is important. After you declare *x* as a global variable, for example, no other function can declare *x* as anything else without ticking off the compiler.

Making a global variable

Global variables differ from local variables in two ways. First, because they're global variables, any function in the program can use them. Second, they're declared *outside* of any function. Out there. In the emptiness. Midst the chaos. Strange, but true.

For example:

```
#include <stdio.h>

int score;

int main()
{
```

Etc. . . .

Think of this source code as the beginning of some massive program, the details of which aren't important right now. Notice how the variable `score` is declared. It's done outside of any function, traditionally just before the `main()` function and after the pound-sign dealies (and any prototyping nonsense). That's how global variables are made.

If more global variables are required, they're created in the same spot, right there before the `main()` function. Declaring them works as it normally does; the only difference is that it's done outside of any function.