See? Repetition doesn't hurt the PC. Not one bit.

- The `for` loop has a start, a middle, and an end. The middle part is the `printf()` statement — the part that gets repeated. The rest of the loop, the start and end, are nestled within the `for` keyword's parentheses. (The next section deciphers how this stuff works.)
- Just as with the `if` structure, when only one statement belongs to the `for` command, you could write it like this:

  ```
      for(i=0 ; i<5 ; i=i+1)
          printf("Ouch! Please, stop!\n");
  ```

  The curly braces aren't required when you have only one statement that's repeated. But, if you have more than one, they're a must.
- The `for` loop repeats the `printf()` statement five times.
- Buried in the `for` loop is the following statement:

  ```
  i=i+1
  ```

  This is how you add 1 to a variable in the C language. It's called *incrementation,* and you should read Chapter 11 if you don't know what it is.
- Don't worry if you can't understand the `for` keyword just yet. Read through this whole chapter, constantly muttering "I *will* understand this stuff" over and over. Then go back and reread it if you need to. But read it all straight through first.

For doing things over and over, use the `for` keyword

The word *for* is one of those words that gets weirder and weirder the more you say it: for, for, fore, four, foyer. . . . For he's a jolly good fellow. These are for your brother. For why did you bring me here? An eye for an eye. For the better to eat you with, my dear. Three for a dollar. And on and on. For it can be maddening.

In the C programming language, the `for` keyword sets up a loop, logically called a `for` loop. The `for` keyword defines a starting condition, an ending condition, and the stuff that goes on while the loop is being executed over and over. The format can get a little deep, so take this one step at a time:

```
for(starting; while_true; do_this)
      statement;
```