The `while` keyword (a formal introduction)

The `while` keyword is used in the C language to repeat a block of statements. Unlike the `for` loop, `while` only tells the computer when to end the loop. The loop must be set up before the `while` keyword, and when it's looping, the ending condition — the sizzling fuse or ticking timer — must be working. Then, the loop goes on, la-de-da, until the condition that `while` monitors suddenly becomes FALSE. Then, the party's over, and the program goes on, sadder but content with the fact that it was repeating itself for a while (sic).

Here's the rough format:

```
starting;
while(while_true)
{
      statement(s);
      do_this;
}
```

First, the loop must be set up, which is done with the *starting* statement. For example, this statement (or a group of statements) may declare a variable to be a certain value, to wait for a keystroke, or to do any number of interesting things.

while_true is a condition that `while` examines. If the condition is TRUE, the *statements* enclosed in curly braces are repeated. `while` examines that condition after each loop is repeated, and only when the statement is FALSE does the loop stop.

Inside the curly braces are *statements* repeated by the `while` loop. One of those *statements, do_this,* is required in order to control the loop. The *do_this* part needs to modify the *while_true* condition somehow so that the loop eventually stops or is broken out of.

`While` loops have an advantage over `for` loops in that they're easier to read in English. For example:

```
while(ch!='~')
```

This statement says "While the value of variable `ch` does not equal the tilde character, repeat the following statements." For this to make sense, you must remember, of course, that `!` means *not* in C. Knowing which symbols to pronounce and which are just decorations is important to understanding C programming.