This program creates a 10-by-9 square (okay — *grid*) of numbers and letters by using a nested-loop arrangement.

Enter the source code into your editor. Save your efforts to disk as GRID.C.

Compile the program. Notice that putting two `for` statements together doesn't cause the compiler to spew errors at you (unless you made a typo somewhere).

Run. Here's what the output should look like:

```
Here is thy grid...
1-A 1-B 1-C 1-D 1-E 1-F 1-G 1-H 1-I 1-J
2-A 2-B 2-C 2-D 2-E 2-F 2-G 2-H 2-I 2-J
3-A 3-B 3-C 3-D 3-E 3-F 3-G 3-H 3-I 3-J
4-A 4-B 4-C 4-D 4-E 4-F 4-G 4-H 4-I 4-J
5-A 5-B 5-C 5-D 5-E 5-F 5-G 5-H 5-I 5-J
6-A 6-B 6-C 6-D 6-E 6-F 6-G 6-H 6-I 6-J
7-A 7-B 7-C 7-D 7-E 7-F 7-G 7-H 7-I 7-J
8-A 8-B 8-C 8-D 8-E 8-F 8-G 8-H 8-I 8-J
9-A 9-B 9-C 9-D 9-E 9-F 9-G 9-H 9-I 9-J
```

Wow. Such efficiency should please any government bureaucracy.

- The first, outer `for` loop counts from 1 to 10.
- The inner `for` loop may seem strange, but it's not. It's only taking advantage of the dual-number/-character nature of letters in a computer. The character variable `b` starts out equal to the letter *A* and is incremented one letter at a time, up to the letter *K*. What happens is that `b` is set equal to the letter *A*'s ASCII value, which is 65. The variable `b` then increments up to the letter *K*'s ASCII value, which is 75. It's sneaky, but doable.
- The `printf()` function displays the numbers and letters as the inner loop spins. You can see this process on your screen: The outer loop stays at one number while the letters *A* through *K* are printed. Then, the outer loop is incremented, and the next row of letters is printed.
- Note that the `printf()` function has a space after the `%c` character. That's what keeps the columns in the grid from running into each other.
- The `putchar()` function displays a single character on the screen. In GRID.C, it's used to display a `\n` newline character at the end of each row.

Break *the Brave and* Continue *the Fool*

Two C language keywords can be used to directly control loops in your programs. The keywords are `break` and `continue`. The `break` keyword should be familiar to you, having been introduced in Chapter 15 and tossed at you every now and again since then. The `continue` keyword is a new beast, but it plays an important role — one that you may even find handy.