```
Able    Baker   Charlie
1       2       3
Alpha   Beta    Gamma
```

Though the \ts in the printf statements look sloppy, the output is definitely organized. Tabular, dude!


- The "tab stops" are preset to every eighth column in C's output. Using a \t inserts a given number of space characters in the output, lining up the next bit of text at the next tab stop. I mention this because some people assume that the tab always moves over eight (or however many) characters. That is not the case.
- The \f and \v characters display special symbols at the Windows command prompt. Rather than a form feed, \f displays the ankh character. Rather than a vertical tab, \v displays the male symbol.
- As long as you know a character's hexadecimal code value, you can always get it displayed by using the \x escape sequence. Just plug in the hexadecimal code and there you go!

# The Complex printf() Format

The printf() function can also be used to display the contents of variables, which you have been seeing throughout this book with integer variables and the %d placeholder, character variables and %c, and so on. To make it happen, printf() uses this format:

```
printf("format_string"[,var[,...]]);
```

Text still appears in double quotes, but after it's used to display the values in variables, it becomes a *format string*. (It's still the same text in double quotes.) The format string is followed by one or more variables, *var*.

Those *var* variables are plugged in to appropriate spots in the format_string according to special percent-sign placeholders. Those percent-sign placeholders are called *conversion characters*. For example:

```
printf("Yeah, I think %s is a jerk, too.\n",jerk);
```

The format string is text that printf() displays on the screen: Yeah, I think ____ is a jerk, too. The %s is a conversion character — a blank — that must be filled by a string of text. (I call them placeholders, but the lords of C claim that they're conversion characters.)