Table 12-2	`if` Comparisons and Their Opposites
`if` ***Comparison***	`else` ***Statement Executed By This Condition***
<	>= (Greater than or equal to)
==	!= (Not equal to)
>	<= (Less than or equal to)
<=	> (Greater than)
>=	< (Less than)
!=	== (Is equal to)

- I don't know about you, but I think that all those symbols in Table 12-2 would certainly make an interesting rug pattern.
- The `else` keyword is used only with `if`.
- Both `if` and `else` can have more than one statement enclosed in their curly braces. `if`'s statements are executed when the comparison is true; `else`'s statements are executed when the comparison is false.

- To *execute* means to run. C programs execute, or run, statements from the top of the source code (the first line) to the bottom. Each line is executed one after the other unless statements like `if` and `else` are encountered. In that case, the program executes different statements, depending on the comparison that `if` makes.
- When your program doesn't require an either-or decision, you don't have to use `else`. For example, the TAXES program has an either-or decision. But, suppose that you're writing a program that displays an error message when something doesn't work. In that case, you don't need `else`; if an error doesn't occur, the program should continue as normal.

- If you're the speaker of another programming tongue, notice that the C language has no `end-else` word in it. This isn't smelly old Pascal, for goodness' sake. The final curly brace signals the end of the `else` statement, just as it does with `if`.

The strange case of `else-if` and even more decisions

The C language is rich with decision making. The `if` keyword helps if you need to test for only one condition. True or false, `if` handles it. And, if it's true, a group of statements is executed. Otherwise, it's skipped over. (After the `if`'s group of statements is executed, the program continues as before.)