Clutter not thy head with this comparison nonsense

The comparison in the `if` statement doesn't have to use any symbols at all! Strange but true. What the C compiler does is to figure out what you have put between the parentheses. Then it weighs whether it's true or false.

For a comparison using <, >, ==, or any of the horde in Table 12-1, the compiler figures out whether the comparison is true or false. However, you can stick just about anything — any valid C statement — between the parentheses and the compiler determines whether it works out to true or false. For example:

```
if(input=1)
```

This `if` statement doesn't figure out whether the value of the `input` variable is equal to `1`. No, you need *two* equal signs for that. Instead, what happens between these parentheses is that the numeric variable `input` is given the value `1`. It's the same as

```
input=1;
```

The C compiler obeys this instruction, stuffing 1 into the `input` variable. Then, it sits back and strokes its beard and thinks, "Does that work out to be true or false?" Not knowing any better, it figures that the statement must be true. It tells the `if` keyword, and the cluster of statements that belong to the `if` statement are then executed.

The final solution to the income-tax problem

I have devised what I think is the fairest and most obviously well-intentioned way to decide who must pay the most in income taxes. You should pay more taxes if you're taller and more taxes if it's warmer outside. Yessir, it would be hard to dodge this one.

This problem is ideal for the `if` keyword to solve. You pay taxes based on either your height or the temperature outside, multiplied by your favorite number and then 10. Whichever number is higher is the amount of tax you pay. To figure out which number is higher, the program TAXES.C uses the `if` keyword with the greater-than symbol. It's done twice — once for the height value and again for the temperature outside: