The cascaded `if` statement could have been written this way instead (the changes are indicated in **bold**):

```
if (value < 2500.00f)
  commission = 30.00f + .017f * value;
else if (value >= 2500.00f && value < 6250.00f)
  commission = 56.00f + .0066f * value;
else if (value >= 6250.00f && value < 20000.00f)
  commission = 76.00f + .0034f * value;
...
```

Although the program will still work, the added conditions aren't necessary. For example, the first `if` clause tests whether `value` is less than 2500 and, if so, computes the commission. When we reach the second `if` test (`value >= 2500.00f && value < 6250.00f`), we know that `value` can't be less than 2500 and therefore must be greater than or equal to 2500. The condition `value >= 2500.00f` will always be true, so there's no point in checking it.

The "Dangling `else`" Problem

When `if` statements are nested, we've got to watch out for the notorious "dangling `else`" problem. Consider the following example:

```
if (y != 0)
  if (x != 0)
    result = x / y;
else
  printf("Error: y is equal to 0\n");
```

To which `if` statement does the `else` clause belong? The indentation suggests that it belongs to the outer `if` statement. However, C follows the rule that an `else` clause belongs to the nearest `if` statement that hasn't already been paired with an `else`. In this example, the `else` clause actually belongs to the inner `if` statement, so a correctly indented version would look like this:

```
if (y != 0)
  if (x != 0)
    result = x / y;
  else
    printf("Error: y is equal to 0\n");
```

To make the `else` clause part of the outer `if` statement, we can enclose the inner `if` statement in braces:

```
if (y != 0) {
  if (x != 0)
    result = x / y;
} else
    printf("Error: y is equal to 0\n");
```

This example illustrates the value of braces; if we'd used them in the original `if` statement, we wouldn't have gotten into this situation in the first place.