declares $i$ and $j$ before the loop, because you can’t combine two declarations with a comma operator. That’s because declarations already use the comma for another purpose—separating items in a list. You can use a single declaration-statement expression to create and initialize two variables, but it’s a bit confusing visually:

```c
int j = 0, i = strlen(word) - 1;
```

In this case the comma is just a list separator, not the comma operator, so the expression declares and initializes both $j$ and $i$. However, it looks as if it declares only $j$.

Incidentally, you can declare `temp` inside the `for` loop:

```c
int temp = word[i];
```

This results in `temp` being allocated and deallocated each loop cycle. This might be a bit slower than declaring `temp` once before the loop. On the other hand, after the loop is finished, `temp` is discarded if it’s declared inside the loop.

**Comma Operator Tidbits**

By far the most common use for the comma operator is to fit two or more expressions into a single `for` loop expression. But C++ does provide the operator with two additional properties. First, it guarantees that the first expression is evaluated before the second expression. Expressions such as the following are safe:

```c
i = 20, j = 2 * i      // i set to 20, j set to 40
```

Second, C++ states that the value of a comma expression is the value of the second part. The value of the preceding expression, for example, is 40, because that is the value of $j = 2 \times i$.

The comma operator has the lowest precedence of any operator. For example, the statement

```c
cata = 17,240;
```

gets read as