The Comma Operator

On occasion, we might like to write a `for` statement with two (or more) initialization expressions or one that increments several variables each time through the loop. We can do this by using a ***comma expression*** as the first or third expression in the `for` statement.

A comma expression has the form

comma expression

expr1 , *expr2*

where *expr1* and *expr2* are any two expressions. A comma expression is evaluated in two steps: First, *expr1* is evaluated and its value discarded. Second, *expr2* is evaluated; its value is the value of the entire expression. Evaluating *expr1* should always have a side effect; if it doesn't, then *expr1* serves no purpose.

For example, suppose that `i` and `j` have the values 1 and 5, respectively. When the comma expression `++i, i + j` is evaluated, `i` is first incremented, then `i + j` is evaluated, so the value of the expression is 7. (And, of course, `i` now has the value 2.) The precedence of the comma operator is less than that of all other operators, by the way, so there's no need to put parentheses around `++i` and `i + j`.

Occasionally, we'll need to chain together a series of comma expressions, just as we sometimes chain assignments together. The comma operator is left associative, so the compiler interprets

```
i = 1, j = 2, k = i + j
```

as

```
((i = 1), (j = 2)), (k = (i + j))
```

Since the left operand in a comma expression is evaluated before the right operand, the assignments `i = 1`, `j = 2`, and `k = i + j` will be performed from left to right.

The comma operator is provided for situations where C requires a single expression, but we'd like to have two or more expressions. In other words, the comma operator allows us to "glue" two expressions together to form a single expression. (Note the similarity to the compound statement, which allows us to treat a group of statements as a single statement.)

macro definitions ➤ 14.3

The need to glue expressions together doesn't arise that often. Certain macro definitions can benefit from the comma operator, as we'll see in a later chapter. The `for` statement is the only other place where the comma operator is likely to be found. For example, suppose that we want to initialize two variables when entering a `for` statement. Instead of writing

```
sum = 0;
for (i = 1; i <= N; i++)
  sum += i;
```

we can write