Table 4.2 A Partial List of C Operators

Precedence	*Name*	*Symbol(s)*	*Associativity*
1	increment (postfix) decrement (postfix)	++ --	left
2	increment (prefix) decrement (prefix) unary plus unary minus	++ -- + -	right
3	multiplicative	* / %	left
4	additive	+ -	left
5	assignment	= *= /= %= += -=	right

operator relative to the other operators in the table (the highest precedence is 1; the lowest is 5). The last column shows the associativity of each operator.

Table 4.2 (or its larger cousin in Appendix A) has a variety of uses. Let's look at one of these. Suppose that we run across a complicated expression such as

```
a = b += c++ - d + --e / -f
```

as we're reading someone's program. This expression would be easier to understand if there were parentheses to show how the expression is constructed from subexpressions. With the help of Table 4.2, adding parentheses to an expression is easy: after examining the expression to find the operator with highest precedence, we put parentheses around the operator and its operands, indicating that it should be treated as a single operand from that point onwards. We then repeat the process until the expression is fully parenthesized.

In our example, the operator with highest precedence is ++, used here as a postfix operator, so we put parentheses around ++ and its operand:

```
a = b += (c++) - d + --e / -f
```

We now spot a prefix -- operator and a unary minus operator (both precedence 2) in the expression:

```
a = b += (c++) - d + (--e) / (-f)
```

Note that the other minus sign has an operand to its immediate left, so it must be a subtraction operator, not a unary minus operator.

Next, we notice the / operator (precedence 3):

```
a = b += (c++) - d + ((--e) / (-f))
```

The expression contains two operators with precedence 4, subtraction and addition. Whenever two operators with the same precedence are adjacent to an operand, we've got to be careful about associativity. In our example, - and + are both adjacent to d, so associativity rules apply. The - and + operators group from left to right, so parentheses go around the subtraction first, then the addition:

```
a = b += (((c++) - d) + ((--e) / (-f)))
```