the first example, 1 is stored into `i`, then the new value of `i` is fetched but not used:

```
i = 1;
```

In the second example, the value of `i` is fetched but not used; however, `i` is decremented afterwards:

```
i--;
```

In the third example, the value of the expression `i * j - 1` is computed and then discarded:

```
i * j - 1;
```

Since `i` and `j` aren't changed, this statement has no effect and therefore serves no purpose.

A slip of the finger can easily create a "do-nothing" expression statement. For example, instead of entering

```
i = j;
```

we might accidentally type

```
i + j;
```

(This kind of error is more common than you might expect, since the = and + characters usually occupy the same key.) Some compilers can detect meaningless expression statements; you'll get a warning such as *"statement with no effect."*

Q & A

Q: **I notice that C has no exponentiation operator. How can I raise a number to a power?**

A: Raising an integer to a small positive integer power is best done by repeated multiplication (`i * i * i` is `i` cubed). To raise a number to a noninteger power, call the pow function.

pow function ➤23.3

Q: **I want to apply the % operator to a floating-point operand, but my program won't compile. What can I do? [p. 54]**

fmod function ➤23.3

A: The `%` operator requires integer operands. Try the `fmod` function instead.

Q: **Why are the rules for using the / and % operators with negative operands so complicated? [p. 54]**

A: The rules aren't as complicated as they may first appear. In both C89 and C99, the goal is to ensure that the value of `(a / b) * b + a % b` will always be equal to `a`