(and indeed, both standards guarantee that this is the case, provided that the value of `a / b` is "representable"). The problem is that there are two ways for `a / b` and `a % b` to satisfy this equality if either `a` or `b` is negative, as seen in C89, where either `-9 / 7` is –1 and `-9 % 7` is –2, or `-9 / 7` is –2 and `-9 % 7` is 5. In the first case, `(-9 / 7) * 7 + -9 % 7` has the value $-1 \times 7 + -2 = -9$, and in the second
C99 case, `(-9 / 7) * 7 + -9 % 7` has the value $-2 \times 7 + 5 = -9$. By the time C99 rolled around, most CPUs were designed to truncate the result of division toward zero, so this was written into the standard as the only allowable outcome.

Q: If C has lvalues, does it also have rvalues? [p. 59]

A: Yes, indeed. An *l*value is an expression that can appear on the *left* side of an assignment; an *r*value is an expression that can appear on the *right* side. Thus, an rvalue could be a variable, constant, or more complex expression. In this book, as in the C standard, we'll use the term "expression" instead of "rvalue."

***Q: You said that *v* += *e* isn't equivalent to *v* = *v* + *e* if *v* has a side effect. Can you explain? [p. 60]**

A: Evaluating *v* += *e* causes *v* to be evaluated only once; evaluating *v* = *v* + *e* causes *v* to be evaluated twice. Any side effect caused by evaluating *v* will occur twice in the latter case. In the following example, `i` is incremented once:

```
a[i++] += 2;
```

If we use = instead of +=, here's what the statement will look like:

```
a[i++] = a[i++] + 2;
```

The value of `i` is modified as well as used elsewhere in the statement, so the effect of executing the statement is undefined. It's likely that `i` will be incremented twice, but we can't say with certainty what will happen.

Q: Why does C provide the `++` and `--` operators? Are they faster than other ways of incrementing and decrementing, or they are just more convenient? [p. 61]

A: C inherited `++` and `--` from Ken Thompson's earlier B language. Thompson apparently created these operators because his B compiler could generate a more compact translation for `++i` than for `i = i + 1`. These operators have become a deeply ingrained part of C (in fact, many of C's most famous idioms rely on them). With modern compilers, using `++` and `--` won't make a compiled program any smaller or faster; the continued popularity of these operators stems mostly from their brevity and convenience.

Q: Do `++` and `--` work with `float` variables?

A: Yes; the increment and decrement operations can be applied to floating-point numbers as well as integers. In practice, however, it's fairly rare to increment or decrement a `float` variable.