```
if (d < n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

The break statement is particularly useful for writing loops in which the exit point is in the middle of the body rather than at the beginning or end. Loops that read user input, terminating when a particular value is entered, often fall into this category:

```
for (;;) {
  printf("Enter a number (enter 0 to stop): ");
  scanf("%d", &n);
  if (n == 0)
    break;
  printf("%d cubed is %d\n", n, n * n * n);
}
```

A break statement transfers control out of the *innermost* enclosing while, do, for, or switch statement. Thus, when these statements are nested, the break statement can escape only one level of nesting. Consider the case of a switch statement nested inside a while statement:

```
while (...) {
  switch (...) {
    ...
    break;
    ...
  }
}
```

The break statement transfers control out of the switch statement, but not out of the while loop. I'll return to this point later.

The continue Statement

The continue statement doesn't really belong here, because it doesn't exit from a loop. It's similar to break, though, so its inclusion in this section isn't completely arbitrary. break transfers control just *past* the end of a loop, while continue transfers control to a point just *before* the end of the loop body. With break, control leaves the loop; with continue, control remains inside the loop. There's another difference between break and continue: break can be used in switch statements and loops (while, do, and for), whereas continue is limited to loops.

The following example, which reads a series of numbers and computes their sum, illustrates a simple use of continue. The loop terminates when 10 nonzero numbers have been read. Whenever the number 0 is read, the continue statement is executed, skipping the rest of the loop body (the statements sum += i; and n++;) but remaining inside the loop.