A: When the body of a `for` loop contains a `continue` statement, the `while` pattern shown in Section 6.3 is no longer valid. Consider the following example from Section 6.4:

```
n = 0;
sum = 0;
while (n < 10) {
  scanf("%d", &i);
  if (i == 0)
    continue;
  sum += i;
  n++;
}
```

At first glance, it looks as though we could convert the `while` loop into a `for` loop:

```
sum = 0;
for (n = 0; n < 10; n++) {
  scanf("%d", &i);
  if (i == 0)
    continue;
  sum += i;
}
```

Unfortunately, this loop isn't equivalent to the original. When `i` is equal to 0, the original loop doesn't increment `n`, but the new loop does.

Q: Which form of infinite loop is preferable, `while (1)` or `for (;;)`? [p. 108]

A: C programmers have traditionally preferred `for (;;)` for reasons of efficiency; older compilers would often force programs to test the `1` condition each time through the `while` loop. With modern compilers, however, there should be no difference in performance.

Q: I've heard that programmers should never use the `continue` statement. Is this true?

A: It's true that `continue` statements are rare. Still, `continue` is handy once in a while. Suppose we're writing a loop that reads some input data, checks that it's valid, and, if so, processes the input in some way. If there are a number of validity tests, or if they're complex, `continue` can be helpful. The loop would look something like this:

```
for (;;) {
  read data;
  if (data fails first test)
    continue;
  if (data fails second test)
    continue;
  .
  .
  .
```