```
  if (data fails last test)
    continue;
  process data;
}
```

Q: What's so bad about the `goto` statement? [p. 114]

A: The `goto` statement isn't inherently evil; it's just that we usually have better alternatives. Programs that use more than a few `goto` statements can quickly degenerate into "spaghetti code," with control blithely jumping from here to there. Spaghetti code is hard to understand and hard to modify.

`goto` statements make programs hard to read because they can jump either forward or backward. (In contrast, `break` and `continue` only jump forward.) A program that contains `goto` statements often requires the reader to jump back and forth in an attempt to follow the flow of control.

`goto` statements can make programs hard to modify, since they make it possible for a section of code to serve more than one purpose. For example, a statement that is preceded by a label might be reachable either by "falling through" from the previous statement or by executing one of several `goto` statements.

Q: Does the null statement have any uses besides indicating that the body of a loop is empty? [p. 116]

A: Very few. Since the null statement can appear wherever a statement is allowed, there are many *potential* uses for the null statement. In practice, however, there's only one other use of the null statement, and it's rare.

Suppose that we need to put a label at the end of a compound statement. A label can't stand alone; it must always be followed by a statement. Putting a null statement after the label solves the problem:

```
{
  ...
  goto end_of_stmt;
  ...
  end_of_stmt: ;
}
```

Q: Are there any other ways to make an empty loop body stand out besides putting the null statement on a line by itself? [p. 117]

A: Some programmers use a dummy `continue` statement:

```
for (d = 2; d < n && n % d != 0; d++)
  continue;
```

Others use an empty compound statement:

```
for (d = 2; d < n && n % d != 0; d++)
  {}
```