allows the controlled statement to be compound, and therefore arbitrarily complex, but we will save more complicated examples for Chapter 3. Some of the worst examples of misused conditional expressions are in Fortran, since the limited facilities of that language encourage greater atrocities.

Part of the reason for this is historical. Fortran II had only the arithmetic IF statement, which does not perform as we suggested in the previous paragraph. Instead it causes a branch to one of three statement numbers, depending on whether an arithmetic expression is negative, zero, or positive. The logical IF was added to Fortran IV as a cleaner way of expressing conditionals. It should always be used instead of the arithmetic IF, especially since two of the three labels are almost always the same in practice.

One of the most productive ways to make a program easier to understand is to reduce the degree of interdependence between statements, so that each part can be studied and understood in relative isolation. One of the major liabilities of the Fortran arithmetic IF is that it *increases* the connections between statements:

```
50 IF(C-COMMA) 55,70,55
55 IF(C-SCOL) 60,70,60
60 IF(C-DASH) 65,70,65
65 NC=NC+1
70 ...
```

The wall-to-wall statement numbers are the first thing to strike the eye. The first line says that if C-COMMA is negative or positive, control transfers to statement 55; if it is zero, control goes to 70. In other words, if C equals COMMA, branch to 70; if not, fall through to the next statement, 55. Similar reasoning applies at statements 55 and 60.

Putting everything together, if C is not a comma and C is not a semicolon and C is not a dash, the statement NC=NC+1 is executed. Or, in Fortran,

```
50 IF (C.NE.COMMA .AND. C.NE.SCOL .AND. C.NE.DASH) NC = NC + 1
```

Most people "understand" an arithmetic IF by mentally translating it into a logical IF, just as we did here. There is little reason ever to use an arithmetic IF.

There is another difficulty with the arithmetic IF version of this program. All those labels in the left margin represent potential targets for branches from other parts of the program. Without reading through *all* of the program from which this excerpt comes, you can't be certain that no other statement branches into the middle of the construction. But when the group of statements is collapsed into a single IF, there is no doubt about how to get to it — it is entered at the beginning and exited at the end, and it has no other connections with the rest of the program. The logical IF reduces the apparent complexity of the program.

Occasionally the third branch of an arithmetic IF can serve to direct an "impossible" condition to error-handling code. It is always good practice to think through such conditions and deal with them properly. Even when all three branches of the arithmetic IF are distinct, however, readability is better served by substituting two logical IF's and a GOTO.