```
            IF CTR > 45 THEN GO TO OVFLO;
          ELSE GO TO RDCARD;
OVFLO:
      ...
```

The first GOTO simply goes around the second GOTO, which seems a bit disorganized. If we replace > by <=, we can write

```
      IF CTR <= 45 THEN GOTO RDCARD;
OVFLO:
      ...
```

One less statement, simpler logic, and, as it happens, we no longer need the label OVFLO. The lesson? Don't branch around branches: turn relational tests around if it makes the program easier to understand. We will soon see a Fortran example of exactly the same failing, which brings up an important point: although details vary from language to language, *the principles of style are the same.* Branching around branches is confusing in any language. So even though you program in Cobol or Basic or assembly language or whatever, the guidelines you find here still apply.

It might seem that we're making a great fuss about a little thing in this last example. After all, it's still pretty obvious what the code says. The trouble is, although any single weakness causes no great harm, the cumulative effect of several confusing statements is code that is simply unintelligible.

Our next example is somewhat larger:

The following is a typical program to evaluate the square root (B) *of a number* (X):

```
      READ(5,1)X
1     FORMAT(F10.5)
      A=X/2
2     B=(X/A+A)/2
      C=B-A
      IF(C.LT.0)C=-C
      IF(C.LT.10.E-6)GOTO 3
      A=B
      GOTO 2
3     WRITE(6,1)B
      STOP
      END
```

Because it is bigger, we can study it on several levels and learn something from each. For instance, before we analyze the code in detail, we might consider whether this program is truly "typical." It is unlikely that a square root routine would be packaged as a main program that reads its input from a file — a function with an argument would be far more useful. Even assuming that we really do want a main program that computes square roots, is it likely that we would want it to compute only one before stopping?

This unfortunate tendency to write overly restricted code influences how we write programs that are supposed to be general. Soon enough we shall meet programs designed to keep track of exactly seventeen salesmen, to sort precisely 500 numbers, to trace through just one maze. We can only guess at how much of the program rewriting that goes on every day actually amounts to entering parameters via the compiler.