```
      GRVAL = A(1)
      DO 25 I = 2,10
      IF (A(I).GT.GRVAL) GO TO 30
      GO TO 25
   30 GRVAL = A(I)
   25 CONTINUE
```

The IF controls a branch that branches around the branch that branches around the statement we wanted to do in the first place! Turning things right side up gives

```
      GRVAL = A(1)
      DO 25 I = 2,10
         IF (A(I) .GT. GRVAL) GRVAL = A(I)
   25 CONTINUE
```

We leave it to the reader to decide whether the IF statement should be replaced by

```
      GRVAL = AMAX1(GRVAL, A(I))
```

when we are finding the larger of just two elements.

Avoid unnecessary branches.

Even though PL/I has adequate facilities for writing programs without any branches at all, they are often neglected, in a style of coding called "Fortran with semicolons." Abuse of PL/I ultimately leads to code like this sorting routine:

```
      DO  M = 1 TO N;
      K =  N-1;
      DO J = 1 TO K;
      IF ARAY(J) - ARAY(J+1) >= 0
                THEN GO TO RETRN;
                     ELSE;
      SAVE = ARAY(J);
      ARAY(J) = ARAY(J+1);
     ARAY(J+1) = SAVE;
RETRN:    END;
      END;
```

The construction THEN GOTO might be an early exit from a loop, but more often is a tipoff that something is amiss. Here it only branches around three statements, not out of the loop. Why not turn the test around so no GOTO or label is required? (The ELSE with no statement after it, a "null ELSE," serves no purpose whatsoever; it only confuses the issue.) Subtraction and comparison against zero is a bad idea because of the danger of overflow or underflow; a direct comparison would be safer and far easier to understand. The outer loop need only be done N–1 times, and the inner N–M times. Of course PL/I allows expressions in the limits of DO loops, so there is no need for the temporary variable K. And the erratic indentation should be changed so it tells how the statements are related to each other. Putting these improvements all together gives