```
TRAPZ: PROCEDURE OPTIONS(MAIN);
   DECLARE (J,K) FIXED DECIMAL (2),
            AREA FIXED DECIMAL (8,6);

   PUT SKIP EDIT ('AREA UNDER THE CURVE',
                  'BY THE TRAPEZOIDAL RULE')
                  (X(9), A, SKIP, X(7), A);
   PUT SKIP;

   DO K = 4 TO 10;
      AREA = 0.5/K;

      DO J = 1 TO K-1;
         AREA = AREA + ((J/K)**2)/K;
      END;

      PUT SKIP EDIT ('FOR DELTA X=1/', K, AREA)
                    (X(2), A, F(2), X(6), F(9,6));
   END;
END;
```

The program now reflects how straightforward the calculation really is. (Both the original and our version are quite specialized. See problem 2.4.)

The original program gave correct answers, yet we were able to improve upon it considerably. It is clear that successful operation is no guarantee of a good program. The changes we made were not designed to decrease execution time (which is too short to measure reliably) or to decrease storage utilization (which improved by thirty percent). Had we been concerned with optimization in the usual sense, we would have factored $1/K^3$ out of the AREA calculation.

What then did we improve? Readability, principally, but also locality and simplicity of structure. AREA is initialized just before it is used, not in two widely separated and illogical places. The calculation now proceeds from top to bottom without the pointless excursion to a sub-procedure. The original program was puffed up with needless declarations and expressions, with over-simple computations and over-complex control structure.

Programs are not used once and discarded, nor are they run forever without change. They evolve. The new version of the integration program has a greater likelihood of surviving changes later without acquiring bugs. It assists instead of intimidating those who must maintain it. This will be the goal of all our revisions.

To summarize some of the specific points of this chapter:

(1) Write clearly. If you find your code branching around branches or around single statements, turn relational tests around. For each GOTO, ask if it could be cleanly eliminated. Avoid constructions like Fortran's arithmetic IF that force GOTO's and labels upon you.

(2) Be sparing with temporary variables. The clutter from too many temporaries confuses readers (including you), and may well thwart an optimizing compiler.