Breaking a program into arbitrary pieces is not sufficient, however. The integration program at the end of Chapter 2 actually benefited by *eliminating* a separate module. This was because there were too many shared assumptions between the calling and called routines. Since it was not possible to ignore either module while studying the other, the separation of operations into two groups simply made more work, forcing the reader to skip back and forth.

Here is the subprocedure OUT once again:

```
OUT: PROCEDURE;
        AREA = AREA + LMTS;
     PUT SKIP EDIT  (MSSG3,K,AREA) (X(2),A(16),F(2),X(6),
          F(9,6));
      AREA = 0;
     RETURN;
   END;
```

To do its assigned task, OUT must be privy to the values of **AREA**, LMTS, MSSG3, and K. Moreover it relies on the calling routine to perform part of the **AREA** calculation; the calling routine in turn depends on OUT to reinitialize **AREA** to zero.

There is no way one can summarize all these relationships succinctly enough to permit separate maintenance of the two routines — yet that is the essence of modularity. It must be possible to describe the *function* performed by a module in the briefest of terms; and it is necessary to minimize whatever relationships exist with other modules, and display those that remain as explicitly as possible. This is how we obtain the minimum "coupling," and hence maximum independence, between modules.

In the case of OUT, this means that the **AREA** calculations should be performed completely in the calling routine, and the literal value of MSSG3 written in place of the variable. If we wish to keep this as a separate module, small as it has now become, then the remaining shared data, **AREA** and K, should be explicitly passed as arguments:

```
OUT: PROCEDURE(K, AREA);   /* PRINT STEP SIZE AND AREA */
   DECLARE K FIXED DECIMAL (2),
        AREA FIXED DECIMAL (8, 6);

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

Make the coupling between modules visible.

There are other considerations besides coupling that affect how to modularize a program. The following program computes the median of a set of numbers, and tells whether the number of elements is even or odd.