```
   MEDIAN: PROCEDURE (X, N) RETURNS (FIXED DECIMAL (5, 2));
      DECLARE (X(*), N) FIXED DECIMAL (3);

      IF N <= 0 THEN
         RETURN(0);
      ELSE IF MOD(N, 2) = 0 THEN
         RETURN((X(N/2) + X(N/2 + 1)) / 2);
      ELSE
         RETURN(X((N+1)/2));
   END;

END;
```

Testing whether N is zero both in the main routine and in MEDIAN may seem like paranoia, but it is actually a small example of defensive programming, a topic we will talk more about in Chapter 6. Someday, someone else will use the median routine, and it would be nice to know that it will do its task sensibly even if the user doesn't take special precautions.

We also chose to isolate the business of reading a list of numbers in a separate procedure because it is a task that occurs in many programs. Not only that, but it involves a messy test for end of file that is expressed in PL/I by an ON-condition and a GOTO. The bulk of the program does not have to know what is involved in detecting the end of input data. In fact, it is much better off *not* knowing exactly what mechanism is used; the input code only adds to the general confusion and may well have to be changed later on.

For all these reasons, it is best to hide the details of reading a list of numbers inside a function that has a simple interface to the outside world. Input/output is almost always messy and subject to change, so we make a point of hiding input and output in separate modules. One good test of the worth of a module, in fact, is how good a job it does of hiding some aspect of the problem from the rest of the code.

Make sure every module hides something.

Program organization, deciding what gets done where, is often given insufficient consideration. This can be true even when the format of input or output data strongly suggests the most convenient order of processing. Failure to heed such suggestions leads to code that is hard to relate to the problem being solved, and hence likely to contain mistakes. Here is part of a program for processing customer accounts: