structures we have described tend to split programs into *computational units* like these and thus lead to internal modularity.

Yet we can still improve the program, by improving its data structure. A cumulative table of days must be calculated by someone and checked by someone else. Since few people are familiar with the number of days up to the end of a particular month, neither writing nor checking is easy. But if instead we use a table of days per month, we can let the computer count them for us. ("Let the machine do the dirty work.")

```
DATES: PROCEDURE OPTIONS (MAIN);
   DECLARE NDAYS(0:1, 1:12) FIXED BINARY INITIAL(
      31,28,31,30,31,30,31,31,30,31,30,31,   /* NON-LEAP */
      31,29,31,30,31,30,31,31,30,31,30,31);  /* LEAP */
   DECLARE (YEAR, DATE, LEAP, MONTH) FIXED BINARY;
   DECLARE TRUE BIT(1) INITIAL ('1'B);

   DO WHILE (TRUE);
      GET LIST (YEAR, DATE) COPY;

      IF MOD(YEAR,400)=0 | (MOD(YEAR,100)¬=0 & MOD(YEAR,4)=0) THEN
         LEAP = 1;
      ELSE
         LEAP = 0;

      IF YEAR<1753 | YEAR>3999 | DATE<=0 | DATE>365+LEAP THEN
         PUT SKIP LIST('BAD YEAR, DATE -', YEAR, DATE);
      ELSE DO;
         DO MONTH = 1 TO 12 WHILE (DATE > NDAYS(LEAP, MONTH));
            DATE = DATE - NDAYS(LEAP, MONTH);
         END;
         PUT SKIP LIST(MONTH, DATE, YEAR);
      END;
   END;
END DATES;
```

Most people know the lengths of the different months ("Thirty days hath September ..."), so the table in this version can be more quickly checked for accuracy. The program may take a bit more time counting the number of days every time it is called, but it is more likely to get the right answer than you are, and even if the program is used a lot, I/O conversions are sure to use more time than the actual computation of the date. The double computation of `NDAYS(LEAP,MONTH)` falls into the same category. Write it clearly so it works; then check later whether it's worth your while to rewrite parts of it.

A second, more general, lesson to be drawn from these variations is that no program is ever perfect; there is always room for improvement. Of course, it is foolish to polish a program beyond the point of diminishing returns, but most programmers do too little revision; they are satisfied too early.

Don't stop with your first draft.
