cases, would have brought this bug to light before it became a bad example.)

You might ask if it's fair to criticize such a "typographical" error, as we have done several times. Our answer is that, regrettably, a program with a typo in it won't work. If you're lucky, it will fail to compile. Worse, like this one, it may run but provide subtly wrong answers.

One flaw in a program is often a clue that more are present. The whole purpose of the strange computation in the CASH loop is to determine the maximum number of times the I-th denomination A(I) can be removed from DIFF without causing overdraw. This is exactly what division does. In fact, DIFF/A(I) gives the desired result when truncated to the next lower integer, and the built-in function FLOOR performs just such a truncation. The CASH loop is not only incorrect, but wasteful and unnecessary.

Let us reconsider the data representation. It hardly seems necessary to rattle down a series of IF's to output the appropriate denomination name. If instead we use indexing, we can rewrite the fragment as:

```
DECLARE NAME(8) CHARACTER(19) INITIAL (
   'TWENTY DOLLAR BILLS', 'TEN DOLLAR BILLS    ',
   'FIVE DOLLAR BILLS  ', 'ONE DOLLAR BILLS    ',
   'QUARTERS           ', 'DIMES               ',
   'NICKELS            ', 'PENNIES             ');
DECLARE DENOM(8) REAL DECIMAL FIXED (6,2)
   INITIAL (20.00, 10.00, 5.00, 1.00, 0.25, 0.10, 0.05, 0.01);
DECLARE NT REAL DECIMAL FIXED (3);
DECLARE DIFF REAL DECIMAL FIXED (8,2);
...

DO I = 1 TO 8;
   NT = FLOOR(DIFF/DENOM(I));
   DIFF = MOD(DIFF, DENOM(I));
   IF NT > 0 THEN
      PUT SKIP(2) LIST (NT, NAME(I));
END;
```

The different denomination names are collected into a character-string array NAME so we can use the computed index NT to select the one to print. NT now performs a useful function and so is retained. Not only is this version smaller and more readable, but it works correctly for any positive value that DIFF can assume. (We have not shown the code for the case when DIFF is negative or zero.)

It is interesting that this program failed to use truncating division when it should have, while the very first example in Chapter 1 used it to excess. In either case, the result is obscure code. Some friendly criticism could have helped each toward a happier middle ground. Curiously, manuscripts are almost always reviewed before publication, yet programs are most often inspected only by the original coder and the compiler. Both have blind spots.

Use data arrays to avoid repetitive control sequences.
