
Don't sacrifice clarity for small gains in "efficiency."

Sometimes a preoccupation with minutiae lets obvious things slip by unnoticed. In the code

```
      I = K + 1
      DO 6 J = 1,30
    6 A(J) = B(J) * C(I)
```

one text observes correctly that C(I) is a constant within the loop, and thus there is no need to make the subscript computation repeatedly. The suggested remedy is

```
      I = K + 1
      TEMP = C(I)
      DO 6 J = 1,30
    6 A(J) = B(J) * TEMP
```

Many compilers will do the trivial optimization of moving constants out of a loop without being asked. (By knowing too much, you may even impede their efforts.) But even for a simple-minded compiler, the first two lines should certainly be combined into

```
      TEMP = C(K+1)
```

Let your compiler do the simple optimizations.

The attempt to re-use pieces of code often leads to tightly knotted programs, difficult to get right, to understand, and to modify later, as in this program that decides if a number is prime:

```
ST1: GET LIST(N);
  IF N¬>1 THEN
    PUT EDIT('ILLEGAL INPUT N=',N,' <= 1')(SKIP,X(10),A,F(5),A);
  ELSE DO; IF N<=3 THEN GO TO APRIME;
    IF N=2*FLOOR(N/2) THEN  NOPRIME: PUT EDIT(N,' IS NOT A PRIME ',
         'NUMBER')(SKIP,F(15),2 A);
    ELSE DO; DO R=3 TO SQRT(N) BY 2; IF N=R*FLOOR(N/R) THEN
         GO TO NOPRIME; END /* OF DO LOOP */;
APRIME: PUT EDIT(N,' IS A PRIME NUMBER')(SKIP,F(15),A); END; END;
  GO TO ST1;
```

The dilemma seems to have been how to avoid duplicating the PUT statement that reports non-primes. The resulting code is almost unreadable. (This is partly the fault of its layout, which we will comment on in Chapter 8.) For instance, it contains a transfer from within an ELSE to the beginning of the corresponding THEN! And instead of using the MOD function to test divisibility, it simulates Fortran's truncating division with the FLOOR function. The code cries out for revision: