```
SMALL = AMIN1(X, Y, Z)
```

One line replaces ten. How can a piece of code that is an order of magnitude too large be considered reliable? There is that much greater chance for confusion, and hence for the introduction of bugs. There is that much more that must be understood in order to make changes.

Library functions like AMIN1 are one way to reduce the apparent complexity of a program; they help to keep program size manageable, and they let you build on the work of others, instead of starting from scratch each time.

Use library functions.

Code that is excessively clever is at least as hard to understand as code that is too simple-minded. For example,

```
DCL TEXT CHAR(200)VAR;
GET LIST(TEXT);
N=0;
START: A=INDEX(TEXT,' ');
IF A=0 THEN GO TO FINISH;
N=N+1;
TEXT=SUBSTR(TEXT,A+1);
GO TO START;
FINISH: PUT LIST(N);
```

Even though this uses PL/I's built-in functions INDEX and SUBSTR, it is hardly clear. INDEX(TEXT,' ') returns the position of the first blank in TEXT, or zero if there is no blank. SUBSTR(TEXT,A+1) produces the substring of TEXT that begins at position A+1; this is re-assigned to TEXT, thus disposing of characters up to and including the leftmost remaining blank. So after a bit of thought, we can see that this program counts the number of blanks in TEXT.

Suppose that you were trying to teach a novice programmer how to count the blanks in a character string? How would you do it? Surely not by this elegant but mystifying method — instead you would say "Look at each character, and if it's a blank, count it." Or, in PL/I,

```
DECLARE TEXT CHARACTER(200) VARYING;
GET LIST (TEXT);
N = 0;
DO I = 1 TO LENGTH(TEXT);
   IF SUBSTR(TEXT, I, 1) = ' ' THEN
      N = N + 1;
END;
PUT LIST (N);
```

This too uses the built-in functions that PL/I provides, but it uses them in a way that clarifies the method of solution, rather than obscuring it. Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Peculiar modes of expression often arise out of attempts to write "efficient" code. The programmer has some knowledge about how a particular compiler