```
/* MAKE V AN IDENTITY MATRIX */
      V = 0.0;
      DO I = 1 TO N;
         V(I,I) = 1.0;
      END;
```

In either case, it is more important to make the purpose of the code unmistakable than to display virtuosity. Even storage requirements and execution time are unimportant by comparison, for setting up an identity matrix must surely be but a small part of the whole program. The problem with obscure code is that debugging and modification become much more difficult, and these are already the hardest aspects of computer programming. Besides, there is the added danger that a too-clever program may not say what you thought it said.

Write clearly — don't be too clever.

Let's pause for a moment and look at what we've done. We studied part of a program, taken verbatim from a programming textbook, and discussed what was good about it and what was bad. Then we made it better. (Not necessarily perfect — just better.) And then we drew a rule or a general conclusion from our analysis and improvements, a rule that would have sounded like a sweeping generality in the abstract, but which makes sense and can be applied once you've seen a specific case.

The rest of the book will be much the same thing — an example from a text, discussion, improvements, and a rule, repeated over and over. When you have finished reading the book, you should be able to criticize your own code. More important, you should be able to write it better in the first place, with less need for criticism.

We have tried to sort the examples into a logical progression, but as you shall see, real programs are like prose — they often violate simultaneously a number of rules of good practice. Thus our classification scheme may sometimes seem arbitrary and we will often have to digress.

Most of the examples will be bigger than the one we just saw, but not excessively so; with the help of our discussion, you should be able to follow them even if you're a beginner. In fact, most of the bigger programs will shrink before your very eyes as we modify them. Sheer size is often an illusion, reflecting only a need for improvement.

The examples are all in either Fortran or PL/I, but if one or both of these languages is unfamiliar, that shouldn't intimidate you any more than size should. Although you may not be able to write a PL/I program, say, you will certainly be able to read one well enough to understand the point we are making, and the practice in reading will make learning PL/I that much easier.

For example, here is a small part of a PL/I program that we will discuss in detail in Chapter 4: