As another example, less artificial, consider the bowling score program that we rewrote in Chapter 3. That used identifiers like X, Y and L, which were devoid of mnemonic value. Here is the same program, this time with variable names chosen to indicate the function of the variable.

```
SCORE = 0;
BALL = 1;
DO FRAME = 1 TO 10;
   IF PINS(BALL) = 10 THEN DO;    /* STRIKE */
      SCORE = SCORE + 10 + PINS(BALL+1) + PINS(BALL+2);
      BALL = BALL + 1;
   END;
   ELSE IF PINS(BALL) + PINS(BALL+1) = 10 THEN DO;    /* SPARE */
      SCORE = SCORE + 10 + PINS(BALL+2);
      BALL = BALL + 2;
   END;
   ELSE DO;          /* REGULAR */
      SCORE = SCORE + PINS(BALL) + PINS(BALL+1);
      BALL = BALL + 2;
   END;
END;
RETURN(SCORE);
```

Which version would you rather have to figure out? Which version will be easier to change a year from now?

Use variable names that mean something.

Statement labels (in PL/I) or numbers (in Fortran) are "mnemonics" just as variable names are — they serve to aid the memory of the person reading the code. Make them meaningful. Look back at the "efficient" sort program of Chapter 7. The sequence of statement numbers in it was

```
1, 6, 19, 3, 21, 20, 2, 8, 7, 9, 16, 17
```

When a statement in the middle of the code says

```
GOTO 19
```

which way do you go? Fortran statement numbers should be used sparingly (avoid the arithmetic IF, which forces at least two upon you), and should be sequenced in increasing order, with gaps between for later insertions. In PL/I, statement labels are rarely needed, but when they are, make them descriptive of their function.

Use statement labels that mean something.

The physical layout of a program should also assist the reader (whether the original programmer or a later modifier) to understand the logical structure. In Chapter 7 we looked at