```
C = C + CI
```

to step through a set of values, but for the moment we will let it pass.

Finding initialization errors is difficult and time-consuming. If you have access to a compiler that checks whether variables have been set before being used (such as WATFIV, IBM's PL/I checkout compiler, PL/C, etc.) use it. Worrying about the "cost" of using a debugging compiler is false economy. Your time is worth more than the small amount of machine time involved. More to the point, you may not find out about some error until it is too late.

Use debugging compilers.

There is another type of faulty initialization which will not be detected by debugging compilers. It occurs regularly in code that uses Fortran's DATA statement to set values. Here is an excerpt from a program we have already seen in Chapter 5:

```
      INTEGER NAME,COLOR,LAST,COL(6),COUNT(6)
      DATA COL/3HBLA,3HBLU,3HBRO,3HGRE,3HRED,3HWHI/,
     1 LAST,COUNT/4H0000,6*0/
      ...
```

Suppose this free-standing program is converted to a subroutine. If the programmer forgets to replace the DATA statement that initializes COUNT by executable code like

```
      DO 10 I = 1,6
         COUNT(I) = 0
   10 CONTINUE
```

the subroutine will fail when called a second time, because the old values will remain in COUNT. The rule is: as much as possible, use DATA for things that are truly constant, like the table of colors in the example; execute initializing code for variables, like counts and sums.

In PL/I, the INITIAL attribute re-initializes AUTOMATIC variables upon each invocation of a procedure, so the problem is less severe. But you should still distinguish between true constants and initialized variables; declare them separately, and comment them clearly.

Initialize constants with DATA *statements or* INITIAL *attributes; initialize variables with executable code.*

Another familiar class of errors is called "off-by-one": some action is done once too often or one time too few, because a test is botched or the limit of a loop is wrong. Let us look at some examples.

The following function sorts M numbers stored in the array V, by placing each number in turn in its correct position among the previous ones. (This is often called "insertion sorting.")