comments both suggest that there might not be two items.)

Make sure input cannot violate the limits of the program.

The statement

```
READ 1,N,(X(I),I=1,N)
```

in the sort program illustrates a risky way to read data. N has to be computed by the user, probably by hand. Since people make mistakes, this is an error-prone operation, especially if N is at all large. And while the statement is compact, it does not leave room for the checking that is so important with this form of input. It should be avoided.

As we said before, computers count better than people; let them do the work. Mark the end of the data, then read until the marker is encountered. (In some languages, the marker can be implicit, as in the

```
ON ENDFILE ... ;
```

statement of PL/I or the END=... construction in many Fortrans.) You would be annoyed if you had to count the number of cards in your source programs; thus compilers read until they find an END card or no more input. Do the same for *your* users.

Terminate input by end-of-file or marker, not by count.

The program below reads cards containing a name and a hair color, and totals the number of people with each color of hair.

```
      INTEGER NAME,COLOR,LAST,COL(6),COUNT(6)
      DATA COL/3HBLA,3HBLU,3HBRO,3HGRE,3HRED,3HWHI/,
     1 LAST,COUNT/4H0000,6*0/
    1 READ(5,2)NAME,COLOR
    2 FORMAT(A4,15X,A3)
      DO 3 I = 1,6
    3 IF( COLOR.EQ.COL(I) ) COUNT(I) = COUNT(I) + 1
      IF(NAME .NE. LAST) GO TO 1
      WRITE(6,4)
    4 FORMAT(12H COLOR COUNT)
      DO 5 I = 1,6
    5 WRITE(6,6)COL(I),COUNT(I)
    6 FORMAT(4X,A3,2X,I3)
      STOP
      END
```

Instead of forcing users to count their data cards correctly, the program uses an explicit "end-of-file" test — a card with a name field of "0000" marks the end of the input. Excellent! But notice that the end-of-file test is made only *after* the data from the end-of-file marker card has been checked and accumulated. Should a color