be punched in the "unused" field of the end-of-file card, it will corrupt the counts. This code may not fail often, but it is careless.

The use of mnemonics like "RED" and "BLU" instead of numeric codes like "1" and "2" is commendable, for it makes the program easier to use correctly. But if a color does not match any of the list, it is quietly skipped. Debugging input data is as important as debugging a program — some provision for locating bad input should be included. In this case, each bad color should be printed (with an indication of which card is in error, for the deck may contain hundreds of cards.)

The program "works", but with negligible effort it can be improved:

```
      INTEGER NAME, COLOR, LAST, COL(6), COUNT(6), CARDNO
      DATA COL(1), COL(2), COL(3) /'BLA', 'BLU', 'BRO'/
      DATA COL(4), COL(5), COL(6) /'GRE', 'RED', 'WHI'/
      DATA LAST /'0000'/
      DO 10 I = 1,6
         COUNT(I) = 0
   10 CONTINUE
      CARDNO = 0
   20 READ(5,21) NAME, COLOR
   21 FORMAT(A4, 15X, A3)
      IF (NAME .EQ. LAST) GOTO 50
         CARDNO = CARDNO + 1
         DO 30 I = 1,6
            IF (COLOR .NE. COL(I)) GOTO 30
               COUNT(I) = COUNT(I) + 1
               GOTO 40
   30    CONTINUE
C FALL OUT IF BAD COLOR
         WRITE(6,31) COLOR, CARDNO
   31    FORMAT(' BAD COLOR - ', A3, ' IN CARD NUMBER', I5)
   40    GOTO 20
C END OF DATA INPUT
   50 WRITE(6,51) (COL(I), COUNT(I), I = 1,6)
   51 FORMAT(4X, A3, I5)
      STOP
      END
```

We have written the input loop as a large DO-WHILE (while the NAME is not equal to LAST). If a color is recognized, we count it and skip to the next case (GOTO 40 is an early exit from the loop); otherwise we list the bad input color and then loop.

Identify bad input; recover if possible.

The same error, treating the end-of-file marker as legitimate data, is carried a step further in this program for computing student grade averages: