```
     CTR = 0;
   GO TO OVFLO;
   RDCARD: READ FILE (CARDIN) INTO (CARD);
/*TABLE LOOKUP FOR VALID CUSTOMER NUMBER*/
     LOOP: DO I = 1 TO 20;
       IF CARD.NUM = NUM_TBL(I)
     THEN GO TO NN;
             ELSE;
       END;
     GO TO RDCARD;
     NN: ...

   IF CARD.AMT > 0 THEN GO TO CR_RTN;
             ELSE GO TO DR_RTN;
     CR_RTN: DETAIL.CREDIT = CARD.AMT;
     DETAIL.DEBIT = 0;
       WRITE FILE (PRTFLE) FROM (DETAIL);
               GO TO TST_CTR;
     DR_RTN: DETAIL.DEBIT = CARD.AMT;
     DETAIL.CREDIT = 0;
       WRITE FILE (PRTFLE) FROM (DETAIL);
     TST_CTR: CTR = CTR + 1;
               IF CTR > 45 THEN GO TO OVFLO;
             ELSE GO TO RDCARD;
   OVFLO:
   WRITE FILE (PRTFLE) FROM (HDR);
   WRITE FILE (PRTFLE)  FROM (COL_HDR);
   WRITE FILE (PRTFLE) FROM (LINE);
     CTR = 0;
   GO TO RDCARD;
```

The program evidently intends to produce a report with a header and up to 46 detail lines per page. But what happens if there are zero transactions, or exactly 46, or 92, or any other multiple of 46? Sure enough, the column headings are printed on an extra page, even though there is no data to go under them. The test for end-of-page should happen before line 47 is printed, not after line 46.

The problem is that the structure of the program is only loosely related to the structure of the output report to be generated. It should be no surprise that the two structures don't always agree. What we want to generate can be described as

```
zero or more pages, each of which has
   a header, and
   one to 46 detail lines
```

We should be able to use our knowledge of the report format to find our way around in the code; yet the header is generated near the bottom of the program and cards are read for the detail lines near the top. The GOTO at the top of the program is a giveaway that things don't happen naturally at the right places.

A pseudo-code program that more closely resembles the report format is

```
WHILE (there's more input)
   IF (we're at top of page)
      write header
   compute detail information
   write detail line
```

This calls for reading an input record and using the success or failure of that