

SAS PUZZLERS! -- Interesting and annoying facts to test your knowledge.

Over the past several years, I have discovered many tips and tricks with SAS -- some of them elegant and efficient, some of them workarounds to annoying features. Here I present a collection of PUZZLERS that I hope you will find entertaining.

I'd like to see this presentation become an annual affair; if you have any favorite PUZZLERS, or come across any during the next year, I encourage you to send them to me (all contributors will be fully acknowledged in any future presentations):

Conrad Reynolds
Baytown Chemical Plant
POB 4004
Baytown TX 77522-4004
713-425-2877

X.400 (all major carriers):
C=US P=EXXON LN=REYNOLDS FN=CONRAD
I=L ORG=CHEMICAL

Puzzler One:

Sometimes you would like to reorder the variables in an existing SAS dataset. You might want them to always print in a certain order without having to explicitly specify the order every time, or you are using the information from PROC CONTENTS to convert datasets to a different form. One way to do this is to write a DATA step and place a LENGTH statement before the SET statement for the existing dataset. This will reorder the variables in the new dataset, but you have to laboriously code length specifications for every variable. Is there an easier way?

Puzzler Two:

I had a user bring me an input dataset (named QTY) that looked like this:

<u>LINE</u>	<u>QUANTITY</u>
1	10
3	20
3	30
4	15
4	5
6	10

The variable LINE had nine possible values, 1 through 9. He wanted an output dataset that had nine observations (one for each possible value), each containing the sum of QUANTITY by LINE. The output dataset was downloaded into an existing spreadsheet, which was why he wanted an observation for each line. For example, the desired output from the input above would be:

<u>LINE</u>	<u>QUANTITY</u>
1	10
2	.
3	50
4	20
5	.
6	10
7	.
8	.
9	.

His solution was to create a dataset containing each number:

```
DATA NUMBERS;  
  DO LINE = 1 TO 9;  
    OUTPUT;  
  END;
```

And then merge them with PROC SQL:

```
PROC SQL;  
  CREATE TABLE OUT AS  
  SELECT NUMBERS.LINE, SUM(QTY.QUANTITY) AS SUM  
  FROM NUMBERS LEFT JOIN QTY  
  ON NUMBERS.LINE = QTY.LINE  
  GROUP BY LINE;
```

This worked, and PROC SQL does make for a rather succinct solution, but what if the variable LINE is not needed in the output dataset (yet you still want all nine observations in order)? Can you solve this without creating the NUMBERS dataset? Can you do it elegantly?

Puzzler Three:

At SUGI 15, Terry Lessmann presented a merging technique that won him a "Best Contributed Paper" award. He had one dataset containing county codes, and all the zip codes associated with them:

<u>STCNTY</u>	<u>ZIP</u>
01001	12345
01001	12366
01002	44444
01003	55555
02022	66666

He also had a dataset that contained county codes and the sales reps responsible for them; note that one county may have more than one rep assigned:

<u>STCNTY</u>	<u>NAME</u>
01001	JAKE
01003	JAKE
01001	SNAKE
01002	LARRY
01002	CURLY
01002	MOE
01003	SCHMEDLY
02022	FISHHEAD

He wanted a dataset that showed every rep and their zip codes for all the counties. The output was to be sorted by county:

<u>STCNTY</u>	<u>NAME</u>	<u>ZIP</u>
01001	JAKE	12345
01001	JAKE	12366
01001	SNAKE	12345
01001	SNAKE	12366
01002	LARRY	44444
01002	CURLY	44444
01002	MOE	44444
01003	JAKE	55555
01003	SCHMEDLY	55555
02022	FISHHEAD	66666

The macro-based solution that was presented required about 45 lines of code, and approximately 13 DATA step passes through the data. Can you find a simpler method?

Puzzler Four:

You have a dataset with SAS Date values and associated data points:

<u>DATE</u>	<u>RESULT</u>
01MAR94	3
02MAR94	4
04MAR94	2
09MAR94	1

You want to analyze the data by month. To do this, you can create a new numeric variable with the MONTH function:

```
MONTH = MONTH(Date);
```

What if you want to analyze the data by week? There is no "WEEK" function; how do you create a numeric variable whose value represents the week?

Puzzler Five (PROC SQL again):

PROC SQL allows you to easily manipulate relational datasets. But sometimes even the simplest report can be puzzling. For example, we create a SAS dataset with the following code:

```
DATA USERDB;  
INPUT USER $ JOBNAME $ CPUTIME;  
CARDS;  
CONRAD SASJOB 1.4  
MARK BRTEST 2.2  
CHERYL SECUPD 3.1  
;  
RUN;
```

Let's look at the SASLOG:

```
1 DATA USERDB;  
2 INPUT USER $ JOBNAME $ CPUTIME;  
3 CARDS;
```

NOTE: The data set WORK.USERDB has 3 observations and 3 variables.

```
7 ;  
8 RUN;
```

Now let's run a simple report with this code:

```
9 PROC SQL;  
10 SELECT * FROM USERDB  
11 WHERE JOBNAME = 'BRTEST';
```

This creates the following output:

```

The SAS System
09:15 Tuesday, June 22, 1993
1
USER      JOBNAME  CPUTIME
-----
MARK      BRTEST   2.2
```

That worked just fine. Now let's run another simple report:

```

12  SELECT * FROM USERDB
13  WHERE USER = 'CONRAD';
NOTE: No rows were selected.
```

What's going on here? Why didn't we get a report?

Puzzler Six:

Look at this bit of SASLOG:

```

9  DATA A;
10  DATE = '06MAY89';
11  MONTH = SUBSTR(DATE,3,3);
12  YEAR = SUBSTR(DATE,6,2);
13  CONCAT = MONTH || YEAR;
14  PUT 'MONTH YEAR ' MONTH YEAR;
15  PUT 'CONCAT      ' CONCAT;
16  RUN;

MONTH YEAR MAY 89
CONCAT    MAY   89
NOTE: The data set WORK.A has 1 observations and 4 variables.
```

Notice how the value of CONCAT has some extra spaces in it. We can see that MONTH has no trailing spaces, and YEAR has no leading spaces. Where did they come from?

Puzzler Seven:

Here's a simple macro:

```

%MACRO SIMPLE(DSN);
  DATA &DSN;
    SET MASTERDS END=LAST;
    IF LAST THEN CALL SYMPUT('LORDER',ORDERNO);
  RUN;
%MEND SIMPLE;
```

Let's run this in a program, and look at the SASLOG:

```
7  OPTIONS MACROGEN;
8
9  %SIMPLE(INVENTORY)
MACROGEN(SIMPLE):  DATA INVENTORY;
MACROGEN(SIMPLE):  SET SASDB.MASTERDS END=LAST;
MACROGEN(SIMPLE):  IF LAST THEN CALL SYMPUT('LORDER',ORDERNO);
MACROGEN(SIMPLE):  RUN;

NOTE: The data set WORK.INVENTORY has 14 observations and 2 variables.

10
11  PROC PRINT;
12  TITLE "Last Order is &LORDER";
WARNING: Apparent symbolic reference LORDER not resolved.
```

Now why is the reference to &LORDER not resolved?

Answer to Puzzler One:

The RETAIN statement will also order the variables in the output dataset in the order that they are specified on the RETAIN statement. It is often easier to use than LENGTH.

Answer to Puzzler Two:

I don't have a really good answer for this one; the following DATA step uses an array:

```
DATA X (KEEP=QUANTITY);
  ARRAY LIST{9} QTY1-QTY9;
  RETAIN QTY1-QTY9;
  SET QTY END=EOF;
  BY LINE;
  IF FIRST.LINE THEN LIST{LINE} = QUANTITY;
  ELSE LIST{LINE} = LIST{LINE} + QUANTITY;
  IF EOF THEN DO IND = 1 TO 9;
    QUANTITY = LIST{IND};
    OUTPUT;
  END;
RUN;
```

This solution requires coding the "9" too many times for my taste (even if I made it a macro variable reference); I'd be interested to see other solutions to this problem.

Answer to Puzzler Three:

This is another case where PROC SQL can radically simplify your code if you are performing relational operations. Here's the easy way:

```
%LET MERGVAR = STCNTY;
PROC SQL;
    CREATE TABLE FINAL(DROP = MATCHVAR) AS
    SELECT * FROM FIRST,
            SECOND(RENAME=(&MERGVAR=MATCHVAR))
    WHERE &MERGVAR = MATCHVAR
    ORDER BY &MERGVAR;
```

Here the beauty of PROC SQL is that you tell it what you want; in this case, all the matches on "&MERGVAR" from the two datasets. The ORDER BY clause just sorts the output.

Answer to Puzzler Four:

You can use the INTNX function to do this. Normally, the INTNX function is used to add an interval from a date or datetime value, but if you specify zero as the value to add, INTNX will set the result to the first of the specified interval. So the solution is:

```
WEEK = INTNX('WEEK',DATE,0);
```

When this is run with the sample input, the result is:

<u>DATE</u>	<u>RESULT</u>	<u>WEEK</u>
01MAR94	3	27FEB94
02MAR94	4	27FEB94
04MAR94	2	27FEB94
09MAR94	1	06MAR94

Answer to Puzzler Five:

This one has bitten me more than once. In PROC SQL, "USER" is a literal that functions as the &SYSJOBID macro variable; that is, it takes on the value of the userid that submitted the SAS job.

Answer to Puzzler Six:

If the target variable of a SUBSTR is undefined, then its length is set to the length of the source argument, not the length specified in the SUBSTR function. Therefore the length of MONTH is 7, not 3, and it really does have trailing spaces. (The PUT statement in line 14 trimmed them off, which is why we don't see them there).

Answer to Puzzler Seven:

This one is straight from the "SAS Guide to Macro Processing" (a book that I highly recommend). SAS does not resolve the reference because the DATA step is complete within the environment of the SIMPLE macro, and therefore the macro variable `LORDER` is local and not known outside the macro. You can make `LORDER` available in open code by moving the `RUN;` statement outside the macro, or previously defining `LORDER`. See "SAS Guide to Macro Processing", Chapter 2, for a full discussion.