PharmaSUG 2013 - Paper TF12

Let's get SAS®sy

Amie Bissonett, inVentiv Health Clinical, Minneapolis, MN

ABSTRACT

The SAS® language has a plethora of procedures, data step statements, functions, and options to boot. Most programmers with years of experience have yet to use all that SAS® has to offer. This paper gives a variety of tips and introduces a few handy tools to add to your SAS® arsenal, including good programming practices, uses of the RETAIN statement, and some macro tips.

INTRODUCTION

This paper provides several examples to demonstrate good programming practices and some SAS®sy tips.

GOOD PROGRAMMING PRACTICES

WRITING LEGIBLE PROGRAMS

Good programming practice starts with legible programs, which includes indentation, completion of each DATA step and PROC with a RUN or QUIT statement, whichever is appropriate, and use of the DATA= option on procedures.

While a DATA step or PROC will execute without its own RUN or QUIT statement when SAS® executes the next DATA step or PROC, it is best practice to conclude them all with a RUN or QUIT statement. This allows for quick recognition of the end of that particular block of code and ensures completed execution before the next set of code is compiled. There are rare cases where code will not run as expected without the use of the RUN or QUIT statement.

Indentation is necessary for programs to be clear to the original programmer and essential when programmers need to review and/or modify another person's programs. A general rule of thumb for indentation is to start the DATA step or PROC statement in column one and indent subsequent statement by at least one space on separate lines.

Example:

```
DATA ds2 ;
  set ds1 ;
  statement 1 ;
  statement 2 ;
run ;

PROC SORT data=ds2 ;
  by var1 ;
run ;
```

This example also shows the separation of PROC SORT from the DATA step as well as explicitly stating the data set name in the PROC SORT call. SAS® will automatically use the most recent data set in a procedure call if a data set is not specified in the DATA= option, however, best practice is to always specify the data set name. PROC SORT is a separate procedure and should not be written within the DATA step as just another statement, as in the following example.

Incorrect example:

```
DATA ds2 ;
  set ds1 ;
  statement 1 ;
  statement 2 ;
  proc sort ;
  by var1 ;
run ;
```

MACRO BASICS

Good practices should also be followed when utilizing the macro language. Indentation is important with macros as well. The %MACRO and %MEND statements should be located in column one and all statements within the macro should be indented at least one space.

When writing macros, each macro should be written individually, i.e. do not embed one macro definition within another macro. Embedding macros makes the code hard to follow for a programmer or reviewer, and more importantly, the embedded macro will be recompiled every time the outer macro is run. This is very inefficient.

A macro syntax topic up for discussion is whether or not to use a period at the end of a macro variable name. In general, the period is not required unless the macro variable is being used for a libname value.

```
%let libnm = sdtm ;

PROC SORT data=&libnm.dm out=dm ;
  by usubjid ;
run ;
```

Submission of this code will resolve the value of &libnm and the period will end the resolution of the macro variable resulting in no period separating the libname from the data set name.

```
ERROR: File WORK.SDTMDM.DATA does not exist.
```

To properly resolve the libname, a second period must be added to get the desired results.

```
PROC SORT data=&libnm..dm out=dm ;
  by usubjid ;
run ;
```

Will correctly resolve to:

```
PROC SORT data=sdtm.dm out=dm ;
  by usubjid ;
run ;
```

COMMENTS

In any programming language, the addition of comments is very important. SAS® gives us a few options for comment syntax.

```
* simple comment ;

/* line comment */

%* macro comment ;
```

The simple comment syntax should only be used in open code, and not within a macro. Any macro statements utilizing this syntax will be executed. Within a macro, the line or macro comment syntax should be used and both will hide macro statements from processing by the macro facility.

TIP! To quickly comment one or more lines of code with the line comment syntax, highlight the row(s) and press CTRL-/ and to un-comment them, press SHIFT-CTRL-/.

PROGRAMMING TIPS

THE RETAIN STATEMENT

The RETAIN statement is very useful when variable values need to be summed or comparisons made over multiple observations.

SYNTAX: RETAIN <variable list(s) <initial value(s)>> ;

Initial values are not required and without them the variable will be initialized with a missing value. Any initial-value assigned is assigned to all variables that precede it in the list up to any preceding initial-value. The RETAIN statement is often used with a sorted data set, a BY statement and FIRST. and LAST. processing as shown in the following example.

Example 1. Using the RETAIN statement to create first and last dose date variables for a subject.

This example assumes that the earliest date for a subject is the first dose date and the latest date is the last dose date. At the first record for a subject, fdosdt is set to startdt and the value will retain over all of the subject's records. At LAST.usubjid, ldosdt is set to startdt and the record is output. The resultant data set will contain one record per subject with their first and last dose dates.

```
DATA ds1a (keep=usubjid fdosdt ldosdt) ;
  set ipadmin ;
  by usubjid startdt ;

  retain fdosdt ldosdt ;

  if first.usubjid then fdosdt = startdt ;
  if last.usubjid then do ;
    ldosdt = startdt ;
    output ;
  end ;
run;
```

Example 2. Using the RETAIN statement to sum values over multiple observations.

Summing values over multiple observations can be accomplished simply with the use of a sum statement without explicit use of the RETAIN statement. When the sum statement is written this way, SAS® will automatically retain the value. This is a simple example that will count the number of observations in the data set and create a macro variable, nobs, with that value.

```
DATA ds2 ;
  set ds1 end=eof ;

  /* this will retain the value of nobs and increment at each record */
  nobs + 1 ;

  if eof then call symput('nobs', nobs) ;
run ;
```

Example 3.

The RETAIN statement may be used with a SAS® function to identify a variable value with a specific characteristic. In this example, a data set is created containing the maximum Common Terminology Criteria for Adverse Events (CTCAE) grade each subject experienced for each distinct lab test.

```
DATA hilabs ;
  set lb ;
  by usubjid lbtestcd ;

  /* retain the value of maxctc across observations */
  retain maxctc ;

  /* initialize maxctc to missing for each lbtestcd within usubjid */
  if first.lbtestcd then maxctc=. ;

  /* compares maxctc to current ctcgrade and sets maxctc to the maximum
      value  */
  maxctc = max(maxctc, ctcgrade) ;

  /* output a record at the last record within usubjid lbtestcd */
  if last.lbtestcd then output ;
run ;
```

Obs	usubjid	lbtestcd	maxctc
1	123	SGPT	0
2	123	TRIG	2
3	123	WBC	0
4	234	ALB	2
5	234	ALP	1

Table 3. Sample portion of output data set, hilabs

Example 4. Using RETAIN to manipulate variable order

Retaining the values of a variable across observations is not the only use of the RETAIN statement. It can also be used to reorder the variables in a data set. The RETAIN statement needs to be prior to the SET statement.

Variables listed in the RETAIN statement will exist prior to any other variables in the output data set. Variables in the input data set which are not listed in the RETAIN statement will appear in the order they exist in the input data set.

The output data set, ds2, will have USUBJID SITEID TRT01A TRTSDT as the first four variables in the data set.

```
DATA ds2 ;
  RETAIN usubjid siteid trt01a trtsdt ;
  set ds1 ;
run ;
```

THE LAG FUNCTION

The LAG function will return values from a queue. It provides another option for retaining a value over observations of a data step.

SYNTAX: LAG<n> (argument) ;

The *n* specifies the number of lagged values and *argument* specifies a numeric or character constant, variable or expression. If the LAG function returns a value to a character variable that has not yet been assigned a length, by default the variable is assigned a length of 200. The memory limit for the LAG function is based on the memory allocated by SAS®. This varies with different operating environments, but generally this is not an issue if a small finite number is applied to *n*.

Example: Using LAG to create new variable.

In this example, the variable y takes on the value of x from the previous observation as dictated by *n*=1. Further, the variable z takes on the value of x from two previous observations as dictated by *n*=2.

```
DATA ds2 ;
  set ds1 ;

  y = lag1(x) ;
  z = lag2(x) ;
run ;
```

Obs	x	y	z
1	1	.	.
2	2	1	.
3	3	2	1
4	4	3	2
5	5	4	3
6	6	5	4

Table 4. Output data set, ds2

CONCLUSION

This paper contains some programming guidance and several programming tips to enhance your SAS® knowledge.

RECOMMENDED READING

SAS(R) 9.2 Language Reference: Dictionary, RETAIN statement, LAG function

Henderson, Don. Carpenter, Art. 2012. "Macro Programming Best Practices: Styles, Guidelines and Conventions Including the Rationale Behind Them". SAS Global Forum 2012.
http://support.sas.com/resources/papers/proceedings12/002-2012.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Amie Bissonett
Enterprise: inVentiv Health Clinical
E-mail: abissonett@inventivhealth.com