Paper 190-2012

Tales from the Help Desk 5: Yet More Solutions for Common SAS® Mistakes

Bruce Gilsen, Federal Reserve Board, Washington, DC, United States

INTRODUCTION

In 25 years as a SAS consultant at the Federal Reserve Board, I have seen SAS users make the same mistakes year after year. This paper reviews some common mistakes, and shows how to fix them. The following topics are reviewed.

1. Use comments in a macro.

2. Handle missing values: arithmetic calculations versus functions.

3. Use %SYSFUNC to execute DATA step functions in a macro.

4. Use the macro IN operator to check in a macro if a value equals one of the values in a list.

In the context of reviewing these mistakes, the paper provides details about SAS system processing that can help users employ the SAS system more effectively. This paper is the fifth of its type; see the references for four previous papers that review other common mistakes.

1. Use comments in a macro.

Note. In this section, the following terms are used.

- A *block comment*, also known as a PL/1 style comment, is a comment that begins with /* and ends with */.

```
/* this example is on one line */

/* this example starts on this line
   1 or more lines in between
   ends on this line */
```

- A *single-statement comment*, also known as an asterisk style comment, is a comment that begins with an asterisk (*) and ends with a semicolon (;).

```
* this example is on one line ;

* this example starts on this line
  1 or more lines in between
  ends on this line;
```

- A *macro comment* is a comment that begins with %* and ends with a semicolon (;). Though generally used in a macro, it can be used outside a macro as well.

```
%* this example is on one line ;

%* this example starts on this line
   1 or more lines in between
   ends on this line;
```

Data set USA has the following values.

```
Obs   year   consume   invest   gov

 1    2008      1         2       3
 2    2009      4         5       6
```

Initially, we calculate GNP in a DATA step and print the results with PROC PRINT, and the code works correctly.

```
data countryprint;
    * Calculate each year's GNP;
  set usa;
  gnp = consume + invest + gov;
run;
proc print data=countryprint;
run;
```

Then, we decide to generalize the code and put it in a macro, as follows.

```
%macro gnpprint(country=);
  data countryprint;
      * Calculate each year's GNP;
    set &country;
    gnp = consume + invest + gov;
  run;
  proc print data=countryprint;
  run;
%mend gnpprint;

%gnpprint(country=usa)
```

The macro does not invoke, and the following cryptic message is written to the SAS log.

```
WARNING: Missing %MEND statement.
```

If we re-submit this code with some additional statements, the problem becomes easier to identify.

```
%macro gnpprint(country=);
  data countryprint;
      * Calculate each year's GNP;
    set &country;
    gnp = consume + invest + gov;
    govplusconsume = gov + consume;
    govplusgovplusinvest = gov + gov + invest;
    consumeplusconsume = consume + comsume;
  run;
  proc print data=countryprint;
  run;
%mend gnpprint;

%gnpprint(country=usa)
```

Now, the following is written to the SAS log.

```
WARNING: The quoted string currently being processed has become more than 262
         characters long.  You may have unbalanced quotation marks.
WARNING: Missing %MEND statement.
```

The problem is that the single quote in the word *year's* is treated as the beginning of a string literal, which is comprised of all characters until the next single quote. While character values can have a maximum length of 32,767 bytes, quoted strings longer than 262 characters generate the warning shown above. Single-statement comments (as in the code above) and macro style comments cause this problem.

To prevent this problem, use block comments in a macro. In this example, change the comment to the following.

```
/* Calculate each year's GNP */
```

For more details about the use of comments in a macro, see SAS Note 32684 (2010).

2. Handle missing values: arithmetic calculations versus functions.

Data set ONE has the following values.

```
Obs    country    gnp2008    gnp2009    gnp2010

 1     usa           1          2          3
 2     japan         4          .          6
 3     china         .          .          9
```

Initially, we do an arithmetic calculation to calculate the mean of GNP2008, GNP2009, and GNP2010.

```
data two;
  set one;
  averagegnp = (gnp2008 + gnp2009 + gnp2010)/3;
run;
```

Data set TWO has the following values of AVERAGEGNP. In the second and third observations, at least one term in the calculation is a missing value (.), so the result is set to missing.

```
Obs    averagegnp

 1         2
 2         .
 3         .
```

Now, we replace the arithmetic calculation with the MEAN function. This generalizes the application because we no longer need to know how many values are used to calculate the mean.

```
data two;
  set one;
  averagegnp = mean(gnp2008, gnp2009, gnp2010);
run;
```

Data set TWO now has different values of AVERAGEGNP in the second and third observations, as follows.

```
Obs    averagegnp

 1         2
 2         5
 3         9
```

To understand why these answers differ, it is important to understand how SAS handles missing values.

- In an arithmetic calculation, SAS sets the results to missing if any of the values are missing.
- For DATA step functions such as MEAN, SUM, and MAX, SAS performs the operation on the non-missing values. For example, the second observation has two non-missing values, so the MEAN is the average of these two values. The result is set to missing only when all the values are missing.

If there are a large number of variables, it can be more convenient to specify them in an abbreviated form. For data set ONE, the following statements are equivalent.

```
(1) averagegnp = mean(gnp2008 + gnp2009 + gnp2010);
(2) averagegnp = mean(of gnp2008 - gnp2010);
(3) averagegnp = mean(of gnp:);
```

In (2), a numbered range list specifies a group of variables whose names differ by a numeric suffix. All variables in the range (GNP2008, GNP2009, and GNP2010) are used to calculate the mean. Any variables in the range that do not exist are created with a value of missing. As discussed in Williams (2006), if OF is omitted, GNP2008-GNP2010 is treated as a difference, not a range list, and SAS calculates the mean of that difference, as illustrated for the first observation.

```
averagegnp = mean(gnp2008 - gnp2010);
             mean(1 - 3);
             mean(-2);
             -2;
```

In (3), the name prefix operator ":" tells SAS to calculate the mean of all variables that begin with "GNP". The OF keyword is required; otherwise an error occurs.

3. Use %SYSFUNC to execute DATA step functions in a macro.

The SAS macro facility provides a modest number of functions. The %SYSFUNC function greatly increases the set of macro tools by allowing you to use all DATA step functions except the following (as of Version 9.2): ALLCOMB, ALLPERM, DIF, DIM, HBOUND, IORCMSG, INPUT, LAG, LBOUND, LEXCOMB, LEXCOMBI, LEXPERK, LEXPERM, MISSING, PUT, RESOLVE, SYMGET, and all variable information functions such as VNAME and VLABEL.

Instead of INPUT and PUT, you can use INPUTN, INPUTC, PUTN, and PUTC, where N or C are for numeric or character.

This section shows two situations where function calls in a DATA step must be modified for use with %SYSFUNC in a macro.

3.1. Quote character values.

Example 1 illustrates a simple use of %SYSFUNC. Example 2 illustrates a problem with quoting character values when using %SYSFUNC.

Example 1. Calculate the mean of some macro variables.

First, let's illustrate the MEAN function in a DATA step. Data set ONE has the following values.

```
Obs   aa     bb

 1    10     20
 2    30     40
 3    50     60
```

In a DATA step, use the MEAN function.

```
data two;
  set one;
  meanab = mean (aa,bb);
run;
```

Data set TWO has the following values.

```
Obs   aa     bb     meanab

 1    10     20       15
 2    30     40       35
 3    50     60       55
```

Alternately, call the MEAN function in a macro with %SYSFUNC, passing in either macro variables or numbers.

```
%macro sysfunc1;
  %let aa1=10;
  %let bb1=20;
  %let aa2=30;
  %let bb2=40;

  %let meanab1 = %sysfunc( mean(&aa1,&bb1) );
  %let meanab2 = %sysfunc( mean(&aa2,&bb2) );
  %let meanab3 = %sysfunc( mean(50,60) );
%mend sysfunc1;
%sysfunc1
```

The macro variables created by this macro are as follows. These results are consistent with the DATA step calculations above.

```
Macro variable    Value
   MEANAB1         15
   MEANAB2         35
   MEANAB3         55
```

Example 2. Use the COUNTC function to count the number of times the character "z" appears in a character string.

First, in a DATA step, the number of occurrences, N_COUNT, is 5.

```
data two;
  n_count =  countc("zbcdddABczzzzbc","z");
run;
```

Alternately, call COUNTC in a macro with %SYSFUNC.

```
%macro sysfunc2;
  %let nn_count = %sysfunc( countc("zbcdddABczzzzbc","z") );
%mend sysfunc2;
%sysfunc2
```

We do not get the correct answer. NN_COUNT is 7, not 5. Because %SYSFUNC is a macro function, all values are treated as character text, and character values should not be quoted, unlike in the DATA step. The quotes are treated as ordinary text characters, so

- The first argument to COUNTC, the string to check, includes the quotes, and has the following value: "zbcdddABczzzzbc".

- The second argument to COUNTC includes the quotes as values to check for, so COUNTC finds all occurrences of double quote, z, or double quote and finds them 7 times.

The %SYSFUNC documentation is misleading about this issue, suggesting that quoting character values is optional rather than incorrect. For example, according to the *SAS 9.2 Macro Reference*,

> Because %SYSFUNC is a macro function, you do not need to enclose character values in quotation marks as you do in DATA step functions. For example, the arguments to the OPEN function are enclosed in quotation marks when the function is used alone, but do not require quotation marks when used within %SYSFUNC.

To get the correct answer, remove the quotes, as in the following code.

```
%macro sysfunc3;
  %let nn_count = %sysfunc( countc(zbcdddABczzzzbc,z) );
%mend sysfunc3;
%sysfunc3
```

3.2. Nest functions.

Another quirk of %SYSFUNC, as discussed in Gilsen (2009), is that functions cannot be nested.

The TODAY function returns the current date as a SAS date value, and the WEEKDAY function returns the day of the week for a SAS date value (1=Sunday, 2=Monday, ... 7=Saturday). In a DATA step, the following statement assigns the day of the week for the current date to the variable WEEK_DAY.

```
week_day = weekday(today());
```

Since functions cannot be nested within %SYSFUNC, the following statement generates an error.

```
%let week_day = %sysfunc(weekday(today()));
```

The following is written to the SAS log.

```
ERROR: Required operator not found in expression: today()
ERROR: Argument 1 to function WEEKDAY referenced by the %SYSFUNC or %QSYSFUNC
       macro function is not a number.
ERROR: Invalid arguments detected in %SYSCALL, %SYSFUNC, or %QSYSFUNC argument
       list.  Execution of %SYSCALL statement or %SYSFUNC or %QSYSFUNC function
       reference is terminated.
```

To resolve the error, use %SYSFUNC once for the WEEKDAY function and once for the TODAY function.

```
%let week_day = %sysfunc(weekday(%sysfunc(today())));
```

4. Use the macro IN operator to check in a macro if a value equals one of the values in a list.

In the DATA step, the IN operator allows you to check if a value equals one of the values in a list, e.g.,

```
   if x in (1 2 3);
```

New in Version 9.2 is the macro IN operator, specified with IN or #. It allows you to check in a macro if a value equals one of the values in a list. This operator is not available by default and is enabled with the system option MINOPERATOR.

Macro IN operator syntax is similar to the DATA step IN operator, except that the list of values to check is just a list of values not enclosed in parentheses. The default separator between each value is a space. Specify a different separator with the MINDELIMITER= system option. For example, to separate with a comma, use: options mindelimiter=',';

Here is a macro that includes some examples of the macro IN operator.

```
   options MINOPERATOR;     /* enable macro IN operator */
   %macro intest1(macvar1=);

       /* Does macro variable MACVAR1 have the value aa, bb, or cc */
     %if &macvar1 in aa bb cc
       %then %put test 1 is in;              /* this text is printed */
       %else %put test 1 is not in;

       /* Same as the previous comparison: # is the same as IN */
     %if &macvar1 # aa bb cc
       %then %put test 2 is in;              /* this text is printed */
       %else %put test 2 is not in;

       /* Value to check is literal text instead of a macro variable */
     %if cc in aa bb cc
       %then %put test 3 is in;              /* this text is printed */
       %else %put test 3 is not in;

     %let macvar1=qq;
     %if &macvar1 in aa bb cc
       %then %put test 4 is in;
       %else %put test 4 is not in;          /* this text is printed */

   %mend intest1;

   %let mymacvar=bb;
   %intest1 (macvar1=&mymacvar)   /* Call the macro */
```

This macro generates the following output.

```
   test 1 is in
   test 2 is in
   test 3 is in
   test 4 is not in
```

Here is a macro that does not use the macro IN operator, and has run successfully in the past.

```
   %macro intest2;
     %let blah=in;
     %if &blah = ABC
       %then %put Blah matches value;
       %else %put Blah does not match value;
   %mend intest2;
   %intest2
```

This macro generates the following output.

```
   Blah does not match value
```

However, if you enable the macro IN operator with OPTIONS MINOPERATOR;, macro INTEST2 stops working and generates the following error.

```
ERROR: Operand missing for IN operator in argument to %EVAL function.
ERROR: The macro INTEST2 will stop executing.
```

The problem is that enabling the macro IN operator introduces an important compatibility issue: in a logical or arithmetic macro expression, a character string that begins with "#" or equals "in" (or "IN", "iN", or "In") must be masked with a macro quoting function. Otherwise, an error occurs because the macro processor treats the “#” or “in” as the macro IN operator, not ordinary text.

To fix the problem, mask BLAH with the macro quoting function %BQUOTE in the %IF statement so that the value will be seen as ordinary text. As modified, this macro generates the expected output.

```
%macro intest2;
  %let blah=in;
  %if %bquote(&blah) = ABC
    %then %put Blah matches value;
    %else %put Blah does not match value;
%mend intest2;
%intest2
```

Note that the error also occurs in the original version of INTEST2 if the value of BLAH begins with "#", but does not occur if the “#” is after the first character or if “IN” is followed by other characters.

```
%let blah=#abc;       /* the error occurs */
%let blah=a#b#c;      /* the error does not occur */
%let blah=INDIANA;    /* the error does not occur */
```

CONCLUSION

This paper reviewed and showed how to fix some common mistakes made by SAS users, and, in the context of discussing these mistakes, provided details about SAS system processing. It is hoped that reading this paper enables users to better understand SAS system processing and thus employ the SAS system more effectively in the future.

For more information, contact the author, Bruce Gilsen, by mail at Federal Reserve Board, Mail Stop N-122, Washington, DC 20551; by e-mail at bruce.gilsen@frb.gov; or by phone at 202-452-2494.

REFERENCES

Gilsen, Bruce (2003), "Deja-vu All Over Again: Common Mistakes by New SAS Users," *Proceedings of the Sixteenth Annual NorthEast SAS Users Group Conference*. <http://www.nesug.org/Proceedings/nesug03/bt/bt010.pdf>

Gilsen, Bruce (2007), "More Tales from the Help Desk: Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2007 Conference*. <http://www2.sas.com/proceedings/forum2007/211-2007.pdf>

Gilsen, Bruce (2009), "Tales from the Help Desk 3: More Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2009 Conference*. <http://support.sas.com/resources/papers/proceedings09/137-2009.pdf>

Gilsen, Bruce (2010), "Tales from the Help Desk 4: Still More Solutions for Common SAS Mistakes," *Proceedings of the SAS Global Forum 2010 Conference*. <http://support.sas.com/resources/papers/proceedings09/137-2009.pdf>

SAS Institute Inc. (2009), "*SAS 9.2 Language Reference: Concepts*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2009), "*SAS 9.2 Language Reference: Dictionary, Volumes 1, 2, and 3*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2009), "*SAS 9.2 Macro Language: Reference*," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2010), SAS Note 32684, "Using comments within a macro." <http://support.sas.com/kb/32/684.html>

Williams, Christianna S. (2006), "Those Sneaky SAS® Functions: Beware of Unexpected Handling of Missing Data and Variable Lists," *Proceedings of the Nineteenth Annual NorthEast SAS Users Group Conference*. <http://www.nesug.org/proceedings/nesug06/cc/cc31.pdf>

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Bill Jackson, Heidi Markovitz and Donna Hill at the Federal Reserve Board. Their support is greatly appreciated.