Paper 27-2010

Macrotize! A Beginner's Guide to Designing and Writing Macros

Stacey D. Phillips, i3 Statprobe, Madison, WI

Abstract

Macro language can often be challenging and intimidating to the novice SAS programmer and sometimes it's hard to know how to get started. This paper is an introduction to the design aspects of creating macros and focuses less on nitty-gritty macro language details. The paper will demonstrate how to think about macro design using a real-world example where the programmer needed to reduce more than a hundred data sets of regression output down to three summary tables. Using several easy steps, the reader will learn some great ways to make macro writing simpler and see an example of how to reduce a seemingly overwhelming task down to a simple and efficient macro.

Introduction

For beginning SAS programmers, the "%" and "&" signs of the SAS macro facility can often be mysterious and perhaps even intimidating. Learning to write SAS macros is a very valuable skill that every programmer should attempt though it can be difficult to know where to start. In the past ten years that I've been programming in SAS I've developed a set of steps and techniques that I utilize each time I write a new macro. For this paper I will focus less on the technical programming details of macro language and focus more on the concepts of designing and writing simple macros for programmers who are just getting started with the SAS macro facility. This paper is meant for beginner SAS programmers who might have some experience with SAS macros and who are interested in learning a set of techniques to further develop their programming expertise. In this paper I will use a real example from a project I recently worked on that allowed me to take a seemingly complex task and break it out down into a set of fairly simple macros to generate three similar outputs.

Five Easy Steps to Writing and Designing Macros

Certainly every programmer has their own unique skill set and way of doing things and as is the case with most programming tasks there is more than one way to do something. This paper will demonstrate the method that I use to design and write macros though there are, of course, many other ways to think about macro design. I've broken my thought process down into the following five steps and will go into further detail of each step in the following pages. I will also provide specific examples from a project where I used this technique to develop several macros for generating three similar outputs from a large number of small data sets.

Step 1 - Know where you are and where you're going.
Step 2 - Pretend you're not writing a macro.
Step 3 - Look for patterns.
Step 4 - Vary one parameter at a time.
Step 5 - Put it all together.

Why Use Macros?

Use of the SAS macro facility is not always appropriate and, in fact, can often complicate your code unnecessarily because macro language is typically more difficult to debug and takes longer to write than regular SAS code. In general, it is best to use macro language when you find yourself writing the same or very similar code over and over as it will save you time in the long run. The macro facility is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do a particular task. The macro facility enables you to assign a name to character strings or groups of SAS programming statements and uses them in the same way it uses the ones you enter in the standard manner. In other words, SAS macros are used to reduce the amount of code one has to write by automating similar operations that are likely to be repeated. Because SAS macro language can be more difficult to debug than standard SAS code you will only want to employ the macro facility for code that is likely to be run often.

Some reasons you might consider utilizing the SAS macro facility:

1. Macros allow you to make a change in one location of your program so that SAS can cascade the change throughout your program.
2. Macros allow you to write a section of code once and use it over and over again in the same program or even different programs.
3. Macros allow you to make programs data driven letting SAS decide what to do based on actual data values.

The Quality of Life (QOL) Real-world Example: Reducing Many Small Data Sets into Multiple Table Outputs

I was recently asked by a study statistician to work on an adhoc request involving a regression analysis based on some quality of life (QOL) data. I was provided with the table shell to model the outputs after and since the regression models had already been run, I merely had to access the data, manipulate and reformat it to match the shells and generate the outputs in RTF format. The task seemed simple enough until I discovered that hundreds of regression models had been run and each individual statistic from each individual model had been output into a separate data set! Ultimately I determined that I needed to reduce 225 individual data sets into six RTF table outputs though I will only show the creation of three of the tables in this paper. In the interest of simplicity, the examples provided for each step below will only include partial sections of the code required to generate the requested QOL tables. The complete code can be found at the end of paper in Step 5. Following is the table shell I was provided with as a model for the requested outputs:

Table 1.1 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ - C30
QOL Analysis Set

					Treatment Comparison		
						95% CI	
Scale	Treatment	n *	Mean Estimate	Comparison	Comparison Estimate	Lower	Upper
Appetite loss scale	A	x	xx.xx	A-C	x.xx	x.xxx	x.xxx
	B	x	xx.xx	B-C	x.xx	x.xxx	x.xxx
	C	x	xx.xx				
Constipation scale	A	x	xx.xx	A-C	x.xx	x.xxx	x.xxx
	B	x	xx.xx	B-C	x.xx	x.xxx	x.xxx
	C	x	xx.xx				
Diarrhoea scale	A	x	xx.xx	A-C	x.xx	x.xxx	x.xxx
	B	x	xx.xx	B-C	x.xx	x.xxx	x.xxx
	C	x	xx.xx				
Additional scales to be included in subsequent pages…							

Page X of Y

Footnote 1
Footnote X

Program: xxx.sas
Output: xxxrtf Source Data: QOL

I also needed to create two additional tables based on this shell using a different QOL parameter for a total of three complete tables:

Table 1.2 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ – LC13
QOL Analysis Set

Table 1.3 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ – TSQM
QOL Analysis Set

The Five Steps in Depth

Step 1: Know where you are and where you are going.

Do you even need to write a macro? The first thing that needs to be determined is whether or not writing a macro is appropriate for the task at hand. Sometimes when we're new to macro language and really starting to understand it we'll write macros that might not be necessary and might add too much complexity where something simpler would be advantageous. So how do we determine whether or not we should be writing a macro or not? My general rule of thumb is to start thinking about macro language whenever I am producing something that involves repetition and/or pattern. The benefits of using SAS macro language include its adaptability and portability which make it ideal for reducing the amount of code necessary when you find yourself repeating code over and over. Here are a few of the many, many examples where use of the SAS Macro facility might be a good idea:

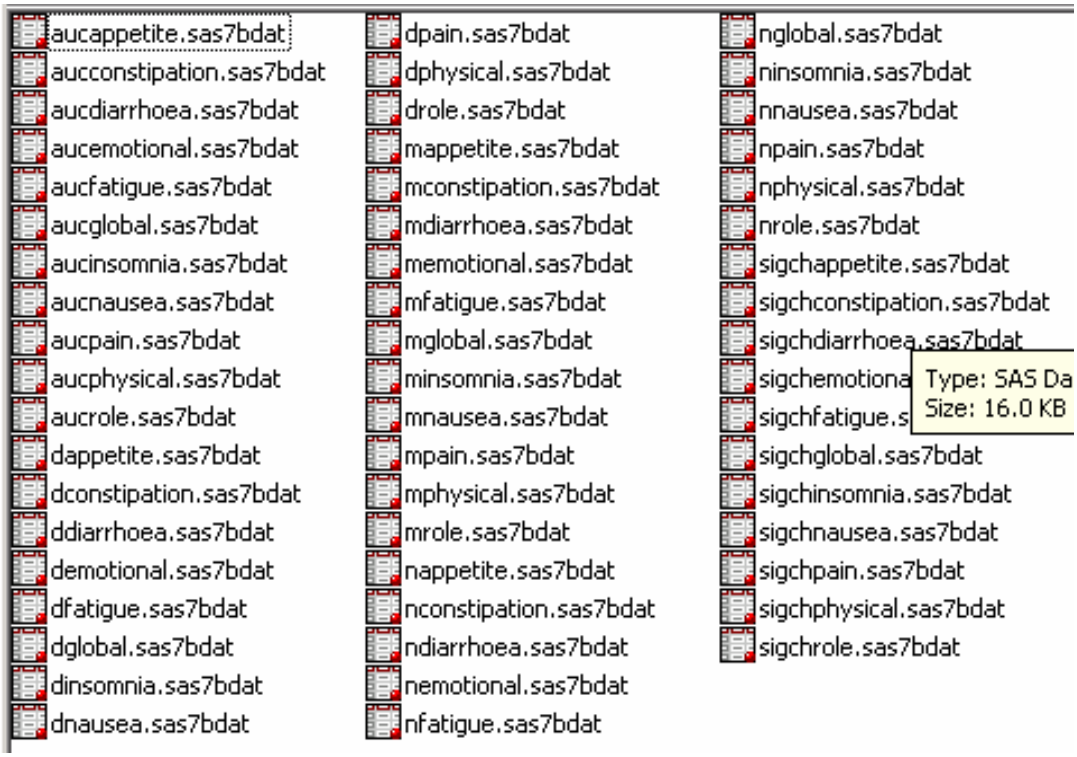
1. You find yourself doing the same statistical calculation in multiple programs so decide to create and store an external macro (macro contained in a completely separate program) in the macro library that can be called whenever these calculations are necessary.
    a. Example: A macro named %count01 is created to count subjects, calculate percentages and format the data into character variables appropriate for display in any RTF table.

2. You are producing the similar output (eg. a data set or table or graph, etc.) for different groups of a certain population.
    a. Example: A macro named %by_state is created to generate an economic profile table for each of the 50 states.

3. You want to run multiple statistical analyses with only slight modifications to the inputs.
    a. Example: A macro named %reg is created to run a series of regression models and includes parameters to specify both the independent and dependent variables for each individual model without retyping the whole PROC REG code.

QOL example: The first step in setting up a macro for this QOL (quality of life) regression assignment is to figure out **where I am**. I have a bunch of data sets that contain all the information that I need but it isn't immediately clear to me what goes where and which data sets belong to which table. I really need to dig into the data and start taking note of how things fit together. To do this, I need to know exactly **where I am going** so I create a copy of the table shell provided by the statistician and annotate onto the shell the data sets and variables required for each section of the table to help organize my thoughts.

Where are we?



As you can see from the screen shot of the input data, there are multiple data sets with similar names but they are prefixed by different letters. To generate this table I've determined that I need the three data sets prefixed by the letters "n", "m" and "d." For example, to create the "appetite loss scale" section of the table I will need three data sets: nappetite.sas7bdat (containing the "n" values), mappetite.sas7bdat (containing the "mean" values) and dappetite.sas7bdat (containing the "comparison estimates and 95% confidence intervals) as annotated in the "Scale" column of the below table shell. Next, I determine the variable names required for each column and annotate them onto the shell. As you can see, patterns of data set naming conventions and variable names are starting to emerge which will be useful for setting up our macro later in Step 3.

Where are we going?

					Treatment Comparison		
						95% CI	
Scale	Treatment **cohortcd**	n	Mean Estimate	Compar-ison	Comparison Estimate	Lower	Upper
Appetite loss scale	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
Data:	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
<n/m/d>appetite	C	**n.n**	**m.estimate**				
Constipation scale	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
Data:	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
<n/m/d>constipation	C	**n.n**	**m.estimate**				
Diarrhoea scale	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
<n/m/d>diarrhoea	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
	C	**n.n**	**m.estimate**				
Additional scales to be included in subsequent pages…							

Page X of Y

Footnote 1
Footnote X

Program: xxx.sas
Output: xxx.rtf Source Data: QOL

Step 2: Pretend you're not writing a macro.

Now that you've determined you need to write a macro I want you to pretend that you're not going to write one. What?! Yes, I want you to forget for a moment that you are ultimately going to produce a SAS macro and do the first pass of your code in standard SAS language only. Absolutely do not include any macro parameters or any other macro language in this first round of writing. Why am I suggesting you do this? The main reason is to focus your code writing on the first task at hand which is accurately producing your output. Using an optional SAS macro will likely add to your efficiency in the long-run but in the short-term you run the risk of overcomplicating your code before you are sure the logic of your basic programming is working correctly. Therefore, it is best to keep the macro language out of the code until you've nailed down your logic. Further, you might find that writing the code out in its "long" form will often help you to notice patterns that will ultimately be useful when you start adding macro variables and other parameters.

QOL example: Building on our example with the QOL data, I've now got a handle on which data sets and which variables are required to output a table that will match the shell provided by the statistician. I'm now going to work on getting the data into the correct format (an RTF table in this case) and am not going to concern myself with macro code at this time.

The plan for generating the first RTF table is to access the required data sets from the directory, manipulate and reformat the data to be consistent with the table shell and then output the file to RTF using PROC REPORT. For this paper I am most interested in demonstrating how to access and compile the data from the directory as that is the part of the process that most utilizes the SAS macro facility. The sections of the code related to variable manipulation and outputting the data using PROC REPORT are less emphasized here in the five steps though will be included in the complete code in Step 5 at the end of the paper.

To access the correct data sets from the directory I will need to SET the separate scale data sets together and then MERGE the "n", "mean" and "estimate/confidence interval" components together. The basic DATA STEP code to complete that task is below:

```
libname c30auc "/project/data/stats/output/PRO/C30/AUC";

data N;
  set  C30AUC.NAPPETITE C30AUC.NCONSTIPATION C30AUC.NDIARRHOEA C30AUC.NEMOTIONAL
       C30AUC.NFATIGUE C30AUC.NGLOBAL C30AUC.NINSOMNIA C30AUC.NNAUSEA C30AUC.NPAIN
       C30AUC.NPHYSICAL C30AUC.NROLE;
run;

data M;
```

```
  set  C30AUC.MAPPETITE C30AUC.MCONSTIPATION C30AUC.MDIARRHOEA C30AUC.MEMOTIONAL
       C30AUC.MFATIGUE C30AUC.MGLOBAL C30AUC.MINSOMNIA C30AUC.MNAUSEA C30AUC.MPAIN
       C30AUC.MPHYSICAL C30AUC.MROLE;
run;

data D;
  set  C30AUC.DAPPETITE C30AUC.DCONSTIPATION C30AUC.DDIARRHOEA C30AUC.DEMOTIONAL
       C30AUC.DFATIGUE C30AUC.DGLOBAL C30AUC.DINSOMNIA C30AUC.DNAUSEA C30AUC.DPAIN
       C30AUC.DPHYSICAL C30AUC.DROLE;
run;

proc sort;
  by scale cohort;
run;

data all;
  merge n(rename=(frequency=n)) m(rename=(estimate=mean)) d;
  by scale cohort;
run;
```

The code to SET and MERGE the data is simple enough, but what I'm really interested in doing is using SAS to figure out the names of the data sets in the directory and generate that list dynamically onto the SET statement so we don't have to know in advance which data sets are in the directory.

Below is the basic DATA step code used to select the required data sets from the directory and to generate two variables (N_DATA and M_DATA) that contain the complete list of data sets to be used on the SET statement. Please don't feel concerned or overwhelmed if you don't fully understand what all of the code is doing as the most important thing to notice from the code below is the similarity in each section; only the data set names are slightly different. Noting these differences will help when we begin to add parameters to our macro.

For beginning macro programmers (and even for the more experienced) it can be really helpful to type everything out explicitly at first even if it seems long and redundant because it makes it much clearer where the variations are located. We could begin macrotizing this section right away after getting the "N" section set-up but would then run the risk of missing some of the places that macro parameters could be added thus costing us extra time and confusion in debugging due to the additional complexity.

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~
  GET "N" (Population) DATA
~~~~~~~~~~~~~~~~~~~~~~~~~~;
proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
  by libname memname;
run;

data vstable;
  set vstable;
  if substr(memname,1,1) ne "N" then delete;
run;

data N_data;
    length N_data $200;
  set vstable;
  by libname memname;
  retain N_data;

  if first.libname then do;
    N_data=trim(left(libname))||"."||trim(left(memname));;
  end;
  else do;
    N_data=trim(left(N_data))||" "||trim(left(libname))||"."||trim(left(memname));
  end;

  if last.libname;
run;
```

```
*~~~~~~~~~~~~~~~~~~~~~~~~
  GET "M" (Mean) DATA
~~~~~~~~~~~~~~~~~~~~~~~~~;
proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
  by libname memname;
run;

data vstable;
  set vstable;
  if substr(memname,1,1) ne "M" then delete;
run;

data M_data;
    length M_data $200;
  set vstable;
  by libname memname;
  retain M_data;

  if first.libname then do;
    M_data=trim(left(libname))||"."||trim(left(memname));;
  end;
  else do;
    M_data=trim(left(M_data))||" "||trim(left(libname))||"."||trim(left(memname));
  end;

  if last.libname;
run;
```

The PROC PRINT of the newly created N_DATA and M_DATA variables containing the data set lists are below:

N_data

```
C30AUC.NAPPETITE C30AUC.NCONSTIPATION C30AUC.NDIARRHOEA C30AUC.NEMOTIONAL C30AUC.NFATIGUE
C30AUC.NGLOBAL C30AUC.NINSOMNIA C30AUC.NNAUSEA C30AUC.NPAIN C30AUC.NPHYSICAL C30AUC.NROLE
```

M_data

```
C30AUC.MAPPETITE C30AUC.MCONSTIPATION C30AUC.MDIARRHOEA C30AUC.MEMOTIONAL C30AUC.MFATIGUE
C30AUC.MGLOBAL C30AUC.MINSOMNIA C30AUC.MNAUSEA C30AUC.MPAIN C30AUC.MPHYSICAL C30AUC.MROLE
```

Step 3: Look for patterns and variability.

Now that you've completed the first pass at the code and everything is working great you are now ready to start thinking about introducing some macro code to your programming by looking for patterns and variability. As previously stated, the reason for writing out seemingly repetitious code in its complete form is to begin to identify and view patterns and similarity that will help you structure your macro correctly.

QOL example: First, let's take a look at the annotated shell again where we will see some similarities in variable and data set naming conventions.

					Treatment Comparison		
						95% CI	
Scale	Treatment **cohortcd**	n	Mean Estimate	Compar-ison	Comparison Estimate	Lower	Upper
Appetite loss scale	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
Data:	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
<n/m/d>appetite	C	**n.n**	**m.estimate**				
Constipation scale	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
Data:	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
<n/m/d>constipation	C	**n.n**	**m.estimate**				

					Treatment Comparison		
						95% CI	
Scale	Treatment **cohortcd**	n	Mean Estimate	Compar-ison	Comparison Estimate	Lower	Upper
	A	**n.n**	**m.estimate**	A-C	**d.estimate**	**d.lower**	**d.upper**
Diarrhoea scale							
<n/m/d>diarrhoea	B	**n.n**	**m.estimate**	B-C	**d.estimate**	**d.lower**	**d.upper**
	C	**n.n**	**m.estimate**				
Additional scales to be included in subsequent pages…							

Page X of Y

Footnote 1
Footnote X

Program: xxx.sas
Output: xxx.rtf Source Data: QOL

An example of a pattern in this case would be the fact that the “mean estimate” data set variable is always named “estimate” in every data set and always comes from the “m” prefixed data set (eg. mdiarrhoea). Similarly, an example of variability in this shell is the varying data set names (eg. appetite, diarrhoea etc.) and their respective prefixes (eg. ndiarrhoea for the “n” values and ddiarrhoea for the “comparisons estimates” and “95% CI”).

Further, if we return to the partial code that generated our data set list in variables N_DATA and M_DATA in Step 2, we can start to highlight these patterns and variables:

```
*~~~~~~~~~~~~~~~~~~~~~~~~~
  GET "N" (Population) DATA
~~~~~~~~~~~~~~~~~~~~~~~~~;
proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
  by libname memname;
run;

data vstable;
  set vstable;
  if substr(memname,1,1) ne "N" then delete;
run;

data N_data;
    length N_data $200;
  set vstable;
  by libname memname;
  retain N_data;

  if first.libname then do;
    N_data=trim(left(libname))||"."||trim(left(memname));;
  end;
  else do;
    N_data=trim(left(N_data))||" "||trim(left(libname))||"."||trim(left(memname));
  end;

  if last.libname;
run;

*~~~~~~~~~~~~~~~~~~~~~~~~~
  GET "M" (Mean) DATA
~~~~~~~~~~~~~~~~~~~~~~~~~;
proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
  by libname memname;
run;

data vstable;
  set vstable;
```

```
  if substr(memname,1,1) ne "M" then delete;
run;

data M_data;
    length M_data $200;
  set vstable;
  by libname memname;
  retain M_data;

  if first.libname then do;
    M_data=trim(left(libname))||"."||trim(left(memname));;
  end;

  else do;
    M_data=trim(left(M_data))||" "||trim(left(libname))||"."||trim(left(memname));
  end;

  if last.libname;
run;
```

In the code above, we can clearly see that the variation of "N" and "M" data set prefixes would be a great place to include a macro parameter. We also know from our annotated shell that the variable names are consistent across data sets so once the individual scale data is SET together and then ultimately MERGED together we can manipulate and reformat all of the variables at once.

Step 4: Vary one thing at a time.

So the code is now working correctly and I've generated a great draft of the output. It's time to finally start writing the macro. It is usually best when just starting out with SAS macros that you vary only one parameter at a time and check your work before adding any additional parameters. If you try to vary more than one thing at a time you run the risk of getting confused very quickly as it will be difficult to determine which change caused the errors, warnings, etc. in your log. As you get better and better at developing macros you will likely be able to successfully vary a few things at a time. If you're a novice, however, it's best to stick with just one parameter at a time.

QOL example: See in the below code and corresponding log output for the "N" data where the data set prefixes are assigned via the macro variable DATA. Note how the previous SAS data set variables of M_DATA and N_DATA have been turned into macro variables of the same name using CALL SYMPUT:

```
%macro auc(data=);
proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
  by libname memname;
run;

data vstable;
  set vstable;
  if substr(memname,1,1) ne "&data" then delete;
run;

data &data._data;
    length &data._data $200;
  set vstable;
  by libname memname;
  retain &data._data;

  if first.libname then do;
    &data._data=trim(left(libname))||"."||trim(left(memname));;
  end;

  else do;
    &data._data=trim(left(&data._data))||"
              "||trim(left(libname))||"."||trim(left(memname));
  end;
  call symput("&data._data",&data._data);
  if last.libname;
```

```
run;

data &data;
  set &&&data._data;
run;

%mend auc;
%auc(data=N);
%auc(data=M);
```

From the SAS log (using MPRINT for debugging):

```
MPRINT(AUC):   proc sort data=sashelp.vstable out=vstable(where=(libname="C30AUC")) ;
MPRINT(AUC):   by libname memname;
MPRINT(AUC):   run;

MPRINT(AUC):   data vstable;
MPRINT(AUC):   set vstable;
MPRINT(AUC):   if substr(memname,1,1) ne "N" then delete;
MPRINT(AUC):   run;

MPRINT(AUC):   data N_data;
MPRINT(AUC):   length N_data $200;
MPRINT(AUC):   set vstable;
MPRINT(AUC):   by libname memname;
MPRINT(AUC):   retain N_data;
MPRINT(AUC):   if first.libname then do;
MPRINT(AUC):   N_data=trim(left(libname))||"."||trim(left(memname));
MPRINT(AUC):   end;
MPRINT(AUC):   else do;
MPRINT(AUC):   N_data=trim(left(N_data))||"
                      "||trim(left(libname))||"."||trim(left(memname));
MPRINT(AUC):   end;
MPRINT(AUC):   call symput("N_data",N_data);
MPRINT(AUC):   if last.libname;
MPRINT(AUC):   run;

MPRINT(AUC):   data N;
MPRINT(AUC):   set C30AUC.NAPPETITE C30AUC.NCONSTIPATION C30AUC.NDIARRHOEA
C30AUC.NEMOTIONAL C30AUC.NFATIGUE C30AUC.NGLOBAL C30AUC.NINSOMNIA C30AUC.NNAUSEA
C30AUC.NPAIN C30AUC.NPHYSICAL C30AUC.NROLE ;
MPRINT(AUC):   run;
```

A similar concept to varying things one at a time is to try to work from the “inside to the outside.” What does that mean exactly? You might find that you have smaller, “inner” macros contained within larger, “outer” macros (commonly referred to as “nested” macros). It is often to your advantage to work on the inner ones first before branching out to the outer macros for reasons similar to why varying one parameter at a time is useful: to reduce complexity and for easier debugging. Remember in this particular example that we need to generate three different tables with similar layouts:

Table 1.1 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ - C30
QOL Analysis Set

Table 1.2 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ – LC13
QOL Analysis Set

Table 1.3 Summary of Statistical Analysis of Time-Adjusted PRO AUC by Scale
EORTC QLQ – TSQM
QOL Analysis Set

The existing macro %AUC in the above code is varying the individual parameters within the table and is considered the “inner” macro whereas the next macro we’d like to develop is an “outer” macro that will generate additional outputs. Though only the partial code for accessing and compiling the data has been shown so far, we can assume that we’ve successfully generated the first table (the C30 version) in its entirely and we’d like to now add the outer macro that will generate the additional outputs (LC13 and TSQM). The basic structure of the program including both the inner and outer macros is shown below.

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  SET-UP LIBNAMES FOR DATA IN DIRECTORY
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
libname c30auc "/project/data/stats/output/PRO/C30/AUC";
libname lc13auc "/project/data/stats/output/PRO/LC13/AUC";
libname tsqmauc "/project/data/stats/stats/output/PRO/TSQM/AUC";

%macro pro(libn=,outfile=); /*OUTER MACRO*/

       %macro auc(data=); /*INNER MACRO*/

       SAS statements

       %mend auc;
       %auc(data=N);
       %auc(data=M);
       %auc(data=D);

       DATA ALL;

       Statements for reformatting data before outputting via PROC REPORT

       RUN;

       PROC REPORT data=all out=&outfile;
       .
       .
       RUN;

%mend pro;
%pro(libn=C30AUC,outfile=t_pro_auc_c30);
%pro(libn=LC13AUC,outfile=t_pro_auc_lc13);
%pro(libn=TSQMAUC,outfile=t_pro_auc_tsqm);
```

The complete code including the addition of the outer macro to generate additional outputs will be shown in step 5 when we put everything together. Not all macro development will include nested macros but in the case that yours does it is a good idea to work on the inner macros first.

Step 5: Put it all together.

The final step to macro writing is to put everything together, review the code and output and adjust as necessary. After you've completed the first four steps and have written both your "inner" and "outer" macros, run your program, review the results and adjust as needed. Specifically:

1. Do a complete run of your program and fix any errors, warnings, etc. in the log file.
2. Review results and compare back to the original assignment. Have you included all of the necessary components?
3. Identify additional places of variability and if applicable, adjust the code and start step #5 again.
4. Once you're convinced the output is what is desired, review the macro code and clean-up as necessary. Check for indentation for better readability and document with comments throughout the macro especially for large, nested macros.

QOL example: The code to generate the three individual quality of life tables in its entirety:

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  SET-UP LIBNAMES FOR DATA IN DIRECTORY
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
libname c30auc "/project/data/stats/output/PRO/C30/AUC";
libname lc13auc "/project/data/stats/output/PRO/LC13/AUC";
libname tsqmauc "/project/data/stats/stats/output/PRO/TSQM/AUC";

*~~~~~~~~~~~~~~~~~~~~~~~~
  BEGIN MACRO CODE
~~~~~~~~~~~~~~~~~~~~~~~~;
```

```
%macro pro(libn=,outfile=); /*OUTER MACRO - generates each individual output*/

      %macro auc(lib=,data=); /*INNER MACRO - generates statistics within each table*/

      *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      GET "N" (Ns), "M" (Mean), "D" (Estimates and 95% CI) DATA
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;

      proc sort data=sashelp.vstable out=vstable(where=(libname="&lib")) ;
        by libname memname;
      run;

      data vstable;
        set vstable;
        if substr(memname,1,1) ne "&data" then delete;
      run;

      data &data._data;
          length &data._data $200;
        set vstable;
        by libname memname;
        retain &data._data;

      if first.libname then do;
         &data._data=trim(left(libname))||"."||trim(left(memname));
      end;

      else do;
        &data._data=trim(left(&data._data))||"
        "||trim(left(libname))||"."||trim(left(memname));
      end;

      /*Dynamically generated list of the data sets in the directory*/
      call symput("&data._data",&data._data);
      if last.libname;
      run;

      data &data;
        set &&&data._data;
      run;

      proc sort;
        by scale cohort;
      run;

      %mend auc;
      %auc(lib=&libn,data=N);
      %auc(lib=&libn,data=M);
      %auc(lib=&libn,data=D);

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  MERGE, MANIPULATE AND REFORMAT DATA
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;

data all;
  merge n(rename=(frequency=n)) m(rename=(estimate=mean)) d;
  by scale cohort;

  treatment=substr(cohort,5,1);

  if treatment='A' then comparison='A-C';
  else if treatment='B' then comparison='B-C';

  array old(*)     mean   estimate ;
  array new(*) $10 mean_c estimate_c ;

  array ci1(*)     lower   upper;
```

```
  array ci2(*) $10 lower_c upper_c;

  do i=1 to dim(old);
    if old(i) ne . then new(i)=trim(left(put(round(old(i),.0001),12.2)));
    else if old(i)=. then new(i)='';
  end;

  do i=1 to dim(ci1);
    if ci1(i) ne . then ci2(i)=trim(left(put(round(ci1(i),.00001),12.3)));
    else if ci1(i)=. then ci2(i)='';
  end;
run;

*~~~~~~~~~~~~~~~~~~~~~~~~~~~
  GENERATE RTF FILE
~~~~~~~~~~~~~~~~~~~~~~~~~~~;
proc report data=all rtf=&outfile;
.
.
run;

%mend pro;
%pro(libn=C30AUC,outfile=t_pro_auc_c30);
%pro(libn=LC13AUC,outfile=t_pro_auc_lc13);
%pro(libn=TSQMAUC,outfile=t_pro_auc_tsqm);
```

Conclusion

SAS macro language can be intimidating to the new SAS programmer though it doesn't need to be if one is willing to learn a few basic concepts related to the SAS macro facility. In this paper I have identified five simple steps for designing and writing macros that should make the process easier and more logical to the beginning macro programmer. By including a real world example of the process the programmer should be able to take these techniques and adapt them to their own macro-related tasks.

References

Slaughter, Susan J. and Delwiche, Lora D. (2004), SAS Macro Programming for Beginners, *Proceedings of the 29th SAS User's Group Conference:* http://www2.sas.com/proceedings/sugi29/243-29.pdf

SAS Institute, Inc., SAS Online Documentation, Version 9.1.3, "SAS Macro Language: Reference," http://support.sas.com/onlinedoc/913/docMainpage.jsp

Phillips, Stacey D. and Klein, Gary (2004), "Let SASHELP Help You: Easy Ways SAS Can Help You with Your Dynamic Programming," *Proceedings of the 15th Midwest SAS User's Group Conference*: http://www.lexjansen.com/mwsug/2004/Pharmaceutical/P2_Phillips_Klein.pdf

Contact Information

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

Stacey D. Phillips
i3 Statprobe
3071 Cimarron Trl.
Madison, WI 53719
Phone: 608-271-6042
Fax: 608-271-6042
E-mail: stacey.phillips@i3statprobe.com
Web: www.i3statprobe.com