Paper 258-2011

SAS® Macro Programming for Beginners

Susan J. Slaughter, Avocet Solutions, Davis, CA
Lora D. Delwiche, University of California, Davis, CA

ABSTRACT

Macro programming is generally considered an advanced topic. But, while macros certainly can be challenging, it is also true that the basic concepts are not difficult to learn. This paper is designed for people who know the basics of SAS programming, but know nothing about SAS macro programming. We explain how the macro processor works, and how to use macros and macro variables. Using these techniques you can create flexible, reusable code that can save you time and effort.

WHY USE MACROS?

Because macro code takes longer to write and debug than standard SAS code, you generally won't use macros in programs that will be run only a few times. But if you find yourself writing similar code over and over again, then macros may make your job easier.

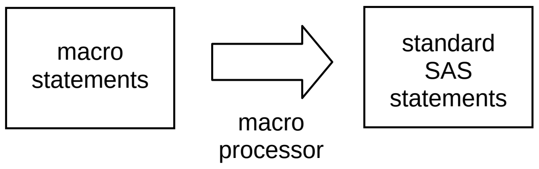
Macros can help in several ways. First, with macros you can make one small change in your program and have SAS echo that change throughout your program. Second, macros can allow you to write a piece of code and use it over and over again in the same program or in different programs. Third, you can make your programs data driven, letting SAS decide what to do based on actual data values.

THE MACRO PROCESSOR

The most important concept to keep in mind whenever you write macro code is that

You are writing a program that writes a program.

Here is how it works.



When you submit a standard SAS program, SAS compiles and then immediately executes it. But when you write macro code, there is an extra step. Before SAS can compile and execute your program, SAS must pass your macro statements to the macro processor which then "resolves" your macros generating standard SAS code. Because you are writing a program that writes a program, this is sometimes called meta-programming.

MACROS VS. MACRO VARIABLES

As you construct macro programs, you will use two basic building blocks: macros and macro variables. You can tell them apart because the names of macro variables start with an ampersand (&), while the names of macros start with a percent sign (%).[1]

A macro variable is like a standard data variable except that it does not belong to a data set and has only a single value which is always character. The value of a macro variable could be a variable name, a numeral, or any text you want substituted in your program.

A macro, however, is a larger piece of a program that can contain complex logic including complete DATA and PROC steps, and macro statements such as %IF-%THEN/%ELSE and %DO-%END. Macros often —but not always— contain macro variables.

THINK GLOBALLY AND LOCALLY

Macro variables come in two varieties: either local or global. A macro variable's scope is local if it is defined inside a macro. Its scope is global if it is defined in "open code" which is everything outside a macro. You can use a global macro variable anywhere in your program, but you can use a local macro variable only inside its own macro. [2]

If you keep this in mind as you write your programs, you will avoid two common mistakes: trying to use a local macro variable outside its own macro, and accidentally creating local and global macro variables having the same name.

YOU MAY QUOTE ME ON THAT

Another caveat to keep in mind is that the macro processor doesn't check for macros inside single quotes. To get around this, simply use double quotes for titles or other quoted strings that contain macro code.

SUBSTITUTING TEXT WITH %LET

%LET is a straightforward macro statement that simply assigns a value to a macro variable. Despite its simplicity, by learning this one statement you will greatly increase the flexibility of your programs.

Suppose you have a program that you run once a month. Every time you have to edit the program so it will select data for the correct month and print the correct title. This is time-consuming and prone to errors. You can use %LET to create a macro variable. Then you can change the value of the macro variable in the %LET statement, and SAS will repeat the new value throughout your program.

The general form a %LET statement is

```
%LET macro-variable-name = value;
```

where *macro-variable-name* is a name you make up following the standard rules for SAS names (32 characters or fewer in length; starting with a letter or underscore; and containing only letters, numerals, or underscores). *Value* is the text to be substituted for the macro variable name, and can be up to 64,000 characters long. The following statements each create a macro variable.

```
%LET iterations = 5;

%LET winner = Lance Armstrong;
```

Notice that there are no quotation marks around *value* even when it contains characters. Blanks at the beginning and end will be trimmed, and everything else between the equal sign and semicolon will become part of the value for the macro variable.

To use the macro variable, you simply add the ampersand prefix (&) and stick the macro variable name wherever you want its value to be substituted.

```
DO i = 1 to &iterations;

TITLE “First: &winner”;
```

After being resolved by the macro processor, these statements would become

```
DO i = 1 to 5;

TITLE “First: Lance Armstrong”;
```

Example

A company that manufactures bicycles maintains a file listing all their models. For each model they record its name, class (Road, Track, or Mountain), list price, and frame material. Here is a subset of the data:

```
Black Bora   Track     796 Aluminum
Delta Breeze Road      699 CroMoly
Jet Stream   Track    1130 CroMoly
Mistral      Road     1995 Carbon Comp
Nor'easter   Mountain  899 Aluminum
Santa Ana    Mountain  459 Aluminum
Scirocco     Mountain 2256 Titanium
Trade Wind   Road      759 Aluminum
```

This DATA step reads the raw data from a file named Models.dat creating a SAS data set named MODELS.

```
DATA models;
   INFILE 'c:\MyRawData\Models.dat' TRUNCOVER;
   INPUT Model $ 1-12 Class $ Price Frame $ 28-38;
RUN;
```

Frequently, the sales people want a list showing all the models of one particular class. Sometimes they want a list of road bikes, but other times they want mountain or track bikes. You could write three different programs—one for each type—or you could write one program with a macro variable. This program uses a %LET statement to create a macro variable named &BIKECLASS. To run the program, you type in the class of bike you want at the beginning of the program, and then SAS echoes that value throughout the program.

```
%LET bikeclass = Mountain;

* Use a macro variable to subset;
PROC PRINT DATA = models NOOBS;
   WHERE Class = "&bikeclass";
   FORMAT Price DOLLAR6.;
   TITLE "Current Models of &bikeclass Bicycles";
RUN;
```

Note that the macro variable &BIKECLASS will be global because it is created outside of a macro in open code. When you submit the program with a value of “Mountain”, then the macro processor will resolve the macro variable and create this standard SAS code.

```
* Use a macro variable to subset;
PROC PRINT DATA = models NOOBS;
   WHERE Type = "Mountain";
   FORMAT Price DOLLAR6.;
   TITLE "Current Models of Mountain Bicycles";
RUN;
```

Here are the results:

```
       Current Models of Mountain Bicycles

     Model         Class        Price      Frame

     Nor'easter    Mountain     $899    Aluminum
     Santa Ana     Mountain     $459    Aluminum
     Scirocco      Mountain   $2,256    Titanium
```

This was a short program, with only two occurrences of the macro variable. But imagine if you had a long program with dozens of occurrences of the macro variable sprinkled throughout. You could save a lot of time and trouble by changing the macro variable only once at the beginning.

CREATING MODULAR CODE WITH MACROS

Anytime you find yourself repeating the same program statements over and over, you might want to consider creating a macro instead. Macros are simply a group of SAS statements that have a name. And, anytime you want that group of statements in your program, you use the name instead of the re-typing all the statements.

The general form of a macro is

```
%MACRO macro-name;
   macro-text
%MEND macro-name;
```

The %MACRO statement tells SAS that this is the beginning of the macro and the %MEND statement signals the end of the macro. *Macro-name* is a name you make up for your macro. The name must follow standard SAS naming conventions (start with a letter or underscore; contain only letters, numerals or underscores; and can be up to 32 characters in length). You don’t need to specify the *macro-name* in the %MEND statement, but it will make your programs easier to understand if you include it. *Macro-text* represents the SAS statements that you want in your macro.

After you define your macro, you need to invoke it when you want to use it. Do this by adding a percent sign in front of the macro-name like this:

```
%macro-name
```

While you do not need to end this statement with a semicolon, adding one generally does no harm.

Example
Looking again at the bicycle models data, the sales personnel like to have lists of models sorted both by model name and by price. The following program creates a macro named PRINTIT that has the PROC PRINT statements you need for the report. Then the program calls the macro twice; first without sorting the data (it is already sorted by model name), and then after executing a PROC SORT by Price.

```
%MACRO printit;
PROC PRINT DATA = models NOOBS;
  TITLE 'Current Models';
  VAR Model Class Frame Price;
  FORMAT Price DOLLAR6.;
RUN;
%MEND printit;

%printit

PROC SORT DATA = models;
  BY Price;
%printit
```

Here are the standard SAS statements that are generated by the macro processor. The first PROC PRINT is generated by the first call to the PRINTIT macro, the PROC SORT comes from the original program and the second PROC PRINT is generated by the second call to the PRINTIT macro.

```
PROC PRINT DATA = models NOOBS;
  TITLE 'Current Models';
  VAR Model Class Frame Price;
  FORMAT Price DOLLAR6.;
RUN;

PROC SORT DATA = models;
  BY Price;
PROC PRINT DATA = models NOOBS;
  TITLE 'Current Models';
  VAR Model Class Frame Price;
  FORMAT Price DOLLAR6.;
RUN;
```

And here are the results of the two PROC PRINTs: one before sorting and one after sorting by Price.

```
                   Current Models

Model          Class      Frame            Price

Black Bora     Track      Aluminum          $796
Delta Breeze   Road       CroMoly           $699
Jet Stream     Track      CroMoly         $1,130
Mistral        Road       Carbon Comp     $1,995
Nor'easter     Mountain   Aluminum          $899
Santa Ana      Mountain   Aluminum          $459
Scirocco       Mountain   Titanium        $2,256
Trade Wind     Road       Aluminum          $759
```

```
                   Current Models

Model          Class      Frame            Price

Santa Ana      Mountain   Aluminum          $459
Delta Breeze   Road       CroMoly           $699
Trade Wind     Road       Aluminum          $759
Black Bora     Track      Aluminum          $796
Nor'easter     Mountain   Aluminum          $899
Jet Stream     Track      CroMoly         $1,130
Mistral        Road       Carbon Comp     $1,995
Scirocco       Mountain   Titanium        $2,256
```

This program is fairly simple with only two calls to the macro, so you wouldn't save much effort by putting the PROC PRINT statements in a macro. But if you had many calls to the macro, you could simplify your program a great deal by using it. And, if you need to make a change to any of the PROC PRINT statements, if they are in a macro, then you would only need to make the change once instead of trying to find every occurrence of the PROC PRINT and risking making mistakes.

ADDING PARAMETERS TO MACROS

Macros allow you to execute a set of SAS statements with just one statement, and while this alone can be helpful, macros are even more powerful when you add parameters to them. Parameters are macro variables whose values are set when you invoke the macro. Parameters give your macros flexibility.

To add parameters to your macro, simply list the macro-variable names followed by an equal sign in parentheses after the macro name:

```
%MACRO macro-name (parameter-1=,  parameter-2=, . . . parameter-n=);
   macro-text
%MEND macro-name;
```

For example, if you have a macro named %MONTHLYREPORT that has parameters for the month and region, it might start like this:

```
%MACRO monthlyreport (month=, region=);
```

Then when you invoke the macro, specify the values for the macro variables after the equal signs:

```
%monthlyreport (month=May, region=West)
```

Because these macro variables are defined in the macro, they are, by default, local to the macro and you cannot use them in any SAS statements outside the macro.

Example

The PRINTIT macro from the previous example prints the desired reports, but you have to sort the data between calls to the macro and the title does not reflect the sort order of the report. This new macro, SORTANDPRINT includes the PROC SORT statements and adds parameters so that you can vary both the sort variable and the sort sequence (descending or not). Then the macro variables are added to a TITLE statement, so the report states how it is sorted. There are two calls to the macro, producing two reports.

```
* For sortvar enter the variable for sorting the report.  For sortseq use either
Descending or do not enter a value for an ascending sequence;


%MACRO sortandprint(sortseq=, sortvar=);
PROC SORT DATA = models;
   BY &sortseq &sortvar;
PROC PRINT DATA = models NOOBS;
   TITLE 'Current Models';
   TITLE2 "Sorted by &sortseq &sortvar";
   VAR Model Class Frame Price;
   FORMAT Price DOLLAR6.;
RUN;
%MEND sortandprint;

%sortandprint(sortseq=Descending, sortvar=Price)

%sortandprint(sortseq=, sortvar=Class)
```

The first call to the macro gives the &SORTSEQ macro variable the value DESCENDING and the &SORTVAR macro variable the value Price. So the statements generated by this call will sort the data by descending values of the variable Price. The second call to the macro gives &SORTSEQ no value and &SORTVAR the value Class. So this call will produce statements that sort the data by ascending (the default) Class.

Here are the SAS statements that are generated by the macro processor:

```
PROC SORT DATA = models;
   BY Descending Price;
PROC PRINT DATA = models NOOBS;
   TITLE 'Current Models';
   TITLE2 "Sorted by Descending Price";
   VAR Model Class Frame Price;
   FORMAT Price DOLLAR6.;
RUN;

PROC SORT DATA = models;
   BY  Class;
PROC PRINT DATA = models NOOBS;
   TITLE 'Current Models';
   TITLE2 "Sorted by  Class";
   VAR Model Class Frame Price;
   FORMAT Price DOLLAR6.;
RUN;
```

Here are the results of the two PROC PRINTs, one sorted by descending Price, the other by Class.

```
                 Current Models
            Sorted by Descending Price

 Model          Class      Frame           Price

 Scirocco       Mountain   Titanium       $2,256
 Mistral        Road       Carbon Comp    $1,995
 Jet Stream     Track      CroMoly        $1,130
 Nor'easter     Mountain   Aluminum         $899
 Black Bora     Track      Aluminum         $796
 Trade Wind     Road       Aluminum         $759
 Delta Breeze   Road       CroMoly          $699
 Santa Ana      Mountain   Aluminum         $459
```

```
                  Current Models
                 Sorted by  Class

 Model          Class      Frame           Price

 Scirocco       Mountain   Titanium       $2,256
 Nor'easter     Mountain   Aluminum         $899
 Santa Ana      Mountain   Aluminum         $459
 Mistral        Road       Carbon Comp    $1,995
 Trade Wind     Road       Aluminum         $759
 Delta Breeze   Road       CroMoly          $699
 Jet Stream     Track      CroMoly        $1,130
 Black Bora     Track      Aluminum         $796
```

MPRINT system option

We have been showing you what SAS sees after the macro processor has resolved your program, but normally you won't see these statements. However if you specify the MPRINT system option in your program, then SAS will print the resolved statements from macros in the SAS log. This can be very useful for debugging purposes. To turn on the MPRINT option, submit an OPTIONS statement like this:

```
OPTIONS MPRINT;
```

Here is what the SAS log looks like with the MPRINT option turned on:

```
6    %MACRO sortandprint(sortseq=, sortvar=);
7    PROC SORT DATA=models;
8      BY &sortseq &sortvar;
9    PROC PRINT DATA=models NOOBS;
10     TITLE 'Current Models';
11     TITLE2 "Sorted by &sortseq &sortvar";
12     VAR Model Class Frame Price;
13     FORMAT Price DOLLAR6.;
14   RUN;
15   %MEND sortandprint;
16   %sortandprint(sortseq=Descending, sortvar=Price);

MPRINT(SORTANDPRINT):   PROC SORT DATA=models;
MPRINT(SORTANDPRINT):   BY Descending Price;

NOTE: There were 8 observations read from the data set WORK.MODELS.
NOTE: The data set WORK.MODELS has 8 observations and 4 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


MPRINT(SORTANDPRINT):   PROC PRINT DATA=models NOOBS;
MPRINT(SORTANDPRINT):   TITLE 'Current Models';
MPRINT(SORTANDPRINT):   TITLE2 "Sorted by Descending Price";
MPRINT(SORTANDPRINT):   VAR Model Class Frame Price;
MPRINT(SORTANDPRINT):   FORMAT Price DOLLAR6.;
MPRINT(SORTANDPRINT):   RUN;

NOTE: There were 8 observations read from the data set WORK.MODELS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


17   %sortandprint(sortseq=, sortvar=Class);
MPRINT(SORTANDPRINT):   PROC SORT DATA=models;
MPRINT(SORTANDPRINT):   BY Class;

NOTE: There were 8 observations read from the data set WORK.MODELS.
NOTE: The data set WORK.MODELS has 8 observations and 4 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


MPRINT(SORTANDPRINT):   PROC PRINT DATA=models NOOBS;
MPRINT(SORTANDPRINT):   TITLE 'Current Models';
MPRINT(SORTANDPRINT):   TITLE2 "Sorted by  Class";
MPRINT(SORTANDPRINT):   VAR Model Class Frame Price;
MPRINT(SORTANDPRINT):   FORMAT Price DOLLAR6.;
MPRINT(SORTANDPRINT):   RUN;

NOTE: There were 8 observations read from the data set WORK.MODELS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

18   RUN;
```

You can see that SAS has inserted into the regular SAS log the MPRINT lines. The statements generated by the macro processor are all labeled with the word MPRINT followed by the name of the macro that generated the statements—in this case SORTANDPRINT. By using the MPRINT system option it is easy to see the standard SAS statements your macro is generating.

CONDITIONAL LOGIC

With macros and macro variables, you have a lot of flexibility. You can increase that flexibility still more by using conditional macro statements such as %IF. Here are the general forms of statements used for conditional logic in macros:

```
%IF condition %THEN action;
   %ELSE %IF condition %THEN action;
   %ELSE action;


%IF condition %THEN %DO;
   action;
%END;
```

These statements probably look familiar because there are parallel statements in standard SAS code, but don't confuse these with their standard counterparts. These statements can only be used inside a macro, and can perform actions that would be completely impossible for standard SAS statements. With %IF, actions can include other macro statements or even complete DATA and PROC steps. Remember, the macro statements won't appear in the standard SAS code generated by the macro processor because you are writing a program that writes a program.

Automatic macro variables

Every time you invoke SAS, the macro processor automatically creates certain macro variables. You can use these in your programs. Two of the most commonly used automatic variables are

Variable Name	Example Value	Description
&SYSDATE	31OCT11	Character value of the date that job or session began
&SYSDAY	Friday	Day of the week that job or session began

Example

The company maintains a file with information about every order they receive. For each order, the data include the customer ID number, date the order was placed, model name, and quantity ordered. Here are the data:

```
287 15MAR11 Delta Breeze 15
287 15MAR11 Santa Ana    15
274 16MAR11 Jet Stream    1
174 17MAR11 Santa Ana    20
174 17MAR11 Nor'easter    5
174 17MAR11 Scirocco      1
347 18MAR11 Mistral       1
287 21MAR11 Delta Breeze 30
287 21MAR11 Santa Ana    25
```

This DATA step reads the raw data from a file named Orders.dat creating a SAS data set named ORDERS.

```
DATA orders;
   INFILE 'c:\MyRawData\Orders.dat';
   INPUT CustomerID $ 1-3 +1 OrderDate DATE7. Model $ 13-24 Quantity;
RUN;
```

Every Monday the president of the company wants a detail-level report showing all the current orders. On Wednesday the president wants a report summarized by customer. You could write two separate programs to run these reports, or you could write one program using macros.

Combining conditional logic with the automatic variable &SYSDAY, the following program creates a macro named REPORTS that prints a detail report if you run it on Monday, and a summary report if you run it on Wednesday (and no report on other days).

```
%MACRO reports;
   %IF &SYSDAY = Monday %THEN %DO;
   PROC PRINT DATA = orders NOOBS;
      FORMAT OrderDate DATE7.;
      TITLE "&SYSDAY Report: Current Orders";
   %END;

   %ELSE %IF &SYSDAY = Wednesday %THEN %DO;
   PROC TABULATE DATA = orders;
      CLASS CustomerID;
      VAR Quantity;
      TABLE CustomerID ALL, Quantity;
      TITLE "&SYSDAY Report: Summary of Orders";
   %END;
%MEND reports;
RUN;

%reports
RUN;
```

If you run this program on Monday, the macro processor with generate the following standard SAS code.

```
PROC PRINT DATA = orders NOOBS;
    FORMAT OrderDate DATE7.;
    TITLE "Monday Report: Current Orders";
```

And the output will look like this:

```
         Monday Report: Current Orders

     Customer    Order
        ID        Date    Model         Quantity

       287      15MAR11    Delta Breeze    15
       287      15MAR11    Santa Ana       15
       274      16MAR11    Jet Stream       1
       174      17MAR11    Santa Ana       20
       174      17MAR11    Nor'easter       5
       174      17MAR11    Scirocco         1
       347      18MAR11    Mistral          1
       287      21MAR11    Delta Breeze    30
       287      21MAR11    Santa Ana       25
```

If you run this program on Wednesday, the macro processor with generate the following standard SAS code.

```
PROC TABULATE DATA = orders;
   CLASS CustomerID;
   VAR Quantity;
   TABLE CustomerID ALL, Quantity;
   TITLE "Wednesday Report: Summary of Orders";
```

And the output will look like this:

```
            Wednesday Report: Summary of Orders

         ----------------------------------
         |                  | Quantity    |
         |                  |-------------|
         |                  |    Sum      |
         |------------------+-------------|
         |CustomerID        |             |
         |------------------|             |
         |174               |        26.00|
         |------------------+-------------|
         |274               |         1.00|
         |------------------+-------------|
         |287               |        85.00|
         |------------------+-------------|
         |347               |         1.00|
         |------------------+-------------|
         |All               |       113.00|
         ----------------------------------
```

DATA-DRIVEN PROGRAMS

This is where your macro programs begin to take on a life of their own. Using the CALL SYMPUT macro routine you can let a macro program look at the data and then decide for itself what to do. CALL SYMPUT takes a value from a DATA step and assigns it to a macro variable which you can then use later in your program.

CALL SYMPUT can take many forms, but to assign a single value to a single macro variable, use CALL SYMPUT with this general form:

```
   CALL SYMPUT("macro-variable", value);
```

where *macro-variable* is the name of a macro variable, either new or old, and is enclosed in quotes. *Value* is the name of a variable from a DATA step whose current value you want to assign to that macro variable.

CALL SYMPUT is often used in IF-THEN statements, for example

```
   IF Place = 1 THEN
      CALL SYMPUT("WinningTime", Time);
```

This statement tells SAS to create a macro variable named &WINNINGTIME and set it equal to the current value of the variable TIME when the value of the variable Place is 1.

Be careful. You cannot create a macro variable with CALL SYMPUT and use it in the same DATA step. Here's why: When you submit macro code, it is resolved by the macro processor, and then compiled and executed. Not until the final stage—execution—does SAS see your data. CALL SYMPUT takes a data value from the execution phase, and passes it back to the macro processor for use in a later step. That's why you must put CALL SYMPUT in one DATA step, but not use it until a later step.

Example

The marketing department wants a report showing all the orders placed by their best customer. The following program finds the customer with the single largest order and then prints all the orders for that customer.

```
*Sort by Quantity;
PROC SORT DATA = orders;
   BY DESCENDING Quantity;

*Use CALL SYMPUT to find biggest order;
DATA _NULL_;
   SET orders;
   IF _N_ = 1 THEN
     CALL SYMPUT("biggest", CustomerID);
   ELSE STOP;

*Print all obs for customer with biggest order;
PROC PRINT DATA = orders NOOBS;
   WHERE CustomerID = "&biggest";
   FORMAT OrderDate DATE7.;
   TITLE "Customer &biggest Had the Single Largest Order";
RUN;
```

This program has three steps. First PROC SORT sorts the data by descending Quantity so that the first observation is the largest single order.

Then a DATA step reads the sorted data, and in the first iteration of the DATA step (when _N_ = 1), uses CALL SYMPUT to assign the current value of CustomerID (287) to the macro variable &BIGGEST. The keyword _NULL_ in the DATA statement tells SAS not to bother creating a new SAS data set, and the STOP statement tells SAS to stop after the first iteration. The DATA _NULL_ and STOP are not necessary but they make the program more efficient by preventing SAS from reading and writing observations that won't be used.

The final step takes the macro variable &BIGGEST and inserts its value into the PROC PRINT so that SAS selects just the orders for customer number 287. Here are the standard SAS statements that SAS will see after the macro processor has resolved the final step:

```
*Print all obs for customer with biggest order;
PROC PRINT DATA = orders NOOBS;
   WHERE CustomerID = "287";
   FORMAT OrderDate DATE7.;
   TITLE "Customer 287 Had the Single Largest Order";
RUN;
```

The output will look like this:

```
     Customer 287 Had the Single Largest Order

   Customer      Order
      ID          Date        Model      Quantity

     287        21MAR11   Delta Breeze     30
     287        21MAR11   Santa Ana        25
     287        15MAR11   Delta Breeze     15
     287        15MAR11   Santa Ana        15
```

CONCLUSIONS

There is no doubt that macros can be very complicated, but don't let that scare you off. Learning to use macro variables and incorporating simple macros into your programs can also make your work a lot easier. Get your feet wet by starting with adding macro variables to your programs. Then when you get the hang of macro variables, start building simple macros.

To avoid mangling your macros, always write them one piece at a time. First write your program in standard SAS code. When that is working and bug-free, then add your %MACRO and %MEND statements. When that is working, then add your parameters, if any, one at a time. If you make sure that each macro feature you add is working before you add another one, then debugging will be vastly simplified.

REFERENCES

Delwiche, Lora D. and Susan J. Slaughter (2008). *The Little SAS Book: A Primer, Fourth Edition*. SAS Institute, Cary, NC.

SUGGESTED READING

Carpenter, Art (2004). *Carpenter's Complete Guide to the SAS Macro Language*. SAS Institute, Cary, NC.

Burlew, Michele (2006). SAS Macro Programming Made Easy. SAS Institute, Cary, NC.

ABOUT THE AUTHORS

Susan Slaughter and Lora Delwiche are also the authors of *The Little SAS Book: A Primer* published by SAS Institute, and may be contacted at:

Susan J. Slaughter	susan@avocetsolutions.com
Lora D. Delwiche	llddelwiche@ucdavis.edu

[1] There are a few exceptions. %INCLUDE, for example, is not part of the macro facility despite its percent sign prefix. And it is possible, though rather confusing, to write macros that do not start with a percent sign.

[2] It is possible to change local macro variables into global macro variables and vice versa using the %GLOBAL and %LOCAL statements.