SESUG Paper 146-2019

How to Write a Macro that Executes Other Macros

Kelly Fenn, Fulton Financial Corporation

ABSTRACT

The ability to create SAS® macro programs can make your life easier when you need to write a chunk of code that will be executed multiple times within a SAS program. Once you have mastered writing macro programs, however, you may find yourself in a situation where you need to call the macro several times, with a number of different parameters. This paper will introduce you to a technique for writing a macro program that will call another macro program repeatedly over a list of parameters, allowing you to avoid manually calling the macro for each unique parameter.

INTRODUCTION

Good programmers are fundamentally lazy, and SAS macro programs are incredibly useful to SAS programmers looking to avoid typing the same code repeatedly. Typing the same chunk of code repeatedly is both a waste of effort and prone to errors in the repetition. Using a macro program allows you to sidestep both these pitfalls. If you have a data set that you need to break into chunks, with separate data sets for each distinct value that one of your variables takes – for example, if you have a data set with a year's worth of data that you want to turn into individual data sets for each of the months – and if you are comfortable writing macro programs, you likely recognize that you can write a macro that extracts a single month of data, then execute that macro for each month in the data set.

However, maybe you don't like the idea of re-typing the call to the macro 12 times, changing the month name in each call. This paper will show you how to use PROC SQL to create a list of macro variables for each distinct value in the relevant column, then write a second macro that calls the first for each macro variable.

THE SAMPLE DATA

The code below creates the sample data that this paper will use:

```
DATA balance_data;
      INPUT year_month $ account $ balance;
      INFILE DATALINES DLM='09'x;
      DATALINES;
201901      1     5061
201901      2     3902
201901      3     6291
201901      4     8443
201901      5     1897
201902      1     67
201902      2     94
201902      3     2235
201902      4     129
201902      5     8
201903      1     75
201903      2     4039
201903      3     3047
201903      4     14869
201903      6     1737
201904      7     615
```

```
201904      2      50012
201904      3      49668
201904      4      4176
201904      6      33781
201905      7      1390
201905      2      20213
201905      3      5
201905      4      1522
201905      6      10219
;
RUN;
```

The code in the rest of the paper will create a wide data set that has the monthly balances for each account still open as of May.

CREATING A WIDE DATA SET

GETTING THE RELEVANT ACCOUNTS WITH MAY BALANCES

Because we are only interested in the accounts that are still in the data as of May, we first want to select the accounts in May, along with the account balance that month. This code (using PROC SQL) does that:

```
PROC SQL;
      CREATE TABLE balances_201905 AS
      SELECT account, balance AS bal_201905
      FROM balance_data
      WHERE year_month = '201905'
      ;
QUIT;
```

The resulting data set:

account	bal_201905
7	1390
2	20213
3	5
4	1522
6	10219

Output 1. Subset of balances in May

JOINING APRIL BALANCES FOR THOSE ACCOUNTS

Now say you want to add the April balances for those accounts. The following code will do that:

```
PROC SQL;
      CREATE TABLE balances_201904 AS
      SELECT a.*, b.balance AS bal_201904
      FROM balances_201905 AS a
      LEFT JOIN
      (SELECT * FROM balance_data WHERE year_month = '201904')
            AS b
      ON a.account = b.account
      ;
QUIT;
```

You could then write a similar PROC SQL statement that would extract the March balances for the accounts you're interested in and join that to the table balances_201904, and so forth. Since you're experienced with macro programs in SAS, however, you recognize that this is a perfect spot in your program to write a macro. This macro:

```
%MACRO balances(year_month, next_year_month);

PROC SQL;
      CREATE TABLE balances_&year_month AS
      SELECT a.*, b.balance AS bal_&year_month
      FROM balances_&next_year_month AS a
      LEFT JOIN
            (SELECT * FROM balance_data WHERE year_month =
            "&year_month") AS b
      ON a.account = b.account
      ;
QUIT;

%MEND;
```

Can be called with the parameters below:

```
%balances(201904, 201905);
```

To create the balances_201904 data set that we created above. You could call it again like so:

```
%balances(201903, 201904);
```

And again twice more, until you created a balances_201901 data set that would have the correct accounts and their monthly balances from January to May. You might be asking yourself, though – is there a better way?

CREATING THE SECOND MACRO

PUTTING THE YEAR_MONTH VARIABLE INTO MACRO VARIABLES

One method for creating a macro variable for each distinct value that a variable can take is to use PROC SQL with INTO:. A benefit of this method is that you do not need to know how many distinct values the variable takes. The code to create these macro variables (called year_month1, year_month2, etc):

```
PROC SQL NOPRINT;
      SELECT DISTINCT year_month
      INTO :year_month1-
      FROM balance_data
      ORDER BY year_month DESC
      ;
QUIT;
```

Here, the year_month variable is sorted in descending order so that the most recent month in our data – May 2019 – will be stored in year_month1.

USING THE MACRO VARIABLES TO CALL THE BALANCES MACRO

To call the balances macro for each month, you need to create a second macro. Even though you will only be calling this second macro once, it uses a %DO loop, and the %DO statement is not valid in open SAS code, so it must be wrapped in a macro program. The code below will execute the balances macro for each month:

```
%MACRO balances_2;
%LET n = %EVAL(&sqlobs - 1);

%DO i = 1 %TO &n;
      %LET j = %EVAL(&i + 1);
      %balances(&&year_month&j, &&year_month&i);
%END;

%MEND;

%balances_2;
```

Sqlobs is an automatic macro variable that is set as the number of rows produced by the most recently-run PROC SQL statement. Because we have started by creating the first data set with the May balances, the balances macro will not need to run for the first macro variable year_month1, so we use %EVAL to create a macro variable n. The %DO loop will need to run n times, where n is the number of distinct values of year_month minus 1.

Because the macro program balances requires 2 parameters, we have to create an additional macro variable – here we use j – so that balances has the appropriate 2 year_month macro variables to use.

THE OUTPUT

The result of calling balances_2 on our data set balance_data is 4 data sets, where the final data set created is the one that we are interested in – the one containing every month of data. To see it, we can run:

```
PROC PRINT DATA=balances_201901; RUN;
```

And see the output:

Obs	account	bal_201905	bal_201904	bal_201903	bal_201902	bal_201901
1	2	20213	50012	4039	94	3902
2	3	5	49668	3047	2235	6291
3	4	1522	4176	14869	129	8443
4	6	10219	33781	1737	.	.
5	7	1390	615	.	.	.

Output 2. All balances of interest

MODIFYING THE CODE FOR RE-USABILITY

In our example, we knew that the final year_month in balance_data was called 201905 and the first was 201901. What if we wanted to be able to re-use the code with a different date range? Or didn't know exactly what months of data were represented, or what form the year_month variable was in? Luckily, macros and PROC SQL can also help us here. The code below creates two macro variables, one for the final month in the data and one for the first month:

```
PROC SQL NOPRINT;
      SELECT max(year_month), min(year_month)
      INTO :final_month, :first_month
      FROM balance_data
      ;
QUIT;
```

Running:

```
%PUT = &final_month;
%PUT = &first_month;
```

Will print the macro variables to the SAS log, allowing us to confirm that they are 201905 and 201901, respectively, as we expected. Modifying the code to create the initial data set like this:

```
PROC SQL;
      CREATE TABLE balances_&final_month AS
      SELECT account, balance as bal_&final_month
      FROM balance_data
      WHERE year_month = "&final_month"
      ;
QUIT;
```

Before running balances_2 and using the code below to view the final output:

```
PROC PRINT DATA=balances_&first_month; RUN;
```

Allows our pair of macro programs to be generalized.

CONCLUSION

Macro programs are already an incredibly powerful tool for SAS programmers, and one of the most important skills for a novice programmer to add to his or her SAS toolbox. Taking the next step and learning to create macro programs that can iteratively execute another macro program makes the macro programming tool that much more useful.

ACKNOWLEDGMENTS

I would like to thank my supervisors and colleagues in the Fulton Financial Corporation Marketing Department – including but not limited to Avi Patel, Amy Hartenstine, Leanne Terpak, Gotham Pasupuleti, Mimi Nguyen, Daniele Knight, and Martin Baker – for their insights, patience, encouragement, and mentorship.

RECOMMENDED READING

- *SAS® Macro Programming Made Easy*

CONTACT INFORMATION

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

Kelly Fenn
Fulton Financial Corporation
KFenn@Fult.com
https://www.linkedin.com/in/kellyf10/