Paper 192-28

Macro Power

Ian Whitlock, Westat, Rockville MD
Quentin McMullen, Brown University, Providence RI

Abstract

The basics of macro programming will be illustrated with many examples to execute and modify. We will start with the concept of a macro as a parameter-driven unit of code, and then add macro decisions, looping, and variables. In the end we will have macros getting help from other macros to solve significant problems.

The student should have a fair idea of what a SAS® program looks like, how to write DATA steps, how to execute base SAS programs in the SAS display manager on a PC, and how to work with multiple edit windows. Some understanding of SQL would be handy, but all code will be explained when used.

Introduction

The beginner often mistakenly gets the impression that SAS macro is sort of a super SAS programming language. It isn't. It is a language for manipulating text to make SAS programs. Through examples, the workshop emphasizes principles that help one to understand the role of macro in SAS programming and the nature of the problems that it can solve.

SAS has developed in a context where variable names are known and tied to specific data elements. This makes it easy to quickly develop small programs that can accomplish a lot. However, the programs are typically applicable to a very specific situation. When given a new dataset, the LIBNAME must change and the member name must change and all the usage of variable names may no longer be appropriate. In a word, SAS programs can be simple, but brittle. Many changes may be needed when the situation changes only slightly.

SAS programs may make good templates for particular problems, but they are not general solutions. They usually require some macro elements to make them general solutions to a class of related problems. The main significance of the macro language is that it can return some of the flexibility to the SAS language needed to make general programs. A macro may be thought of as a parameterized unit of SAS code. So this is where we begin. The authors started with the realization that when one begins teaching macro with macro variables, one has already started by teaching the student to emphasize global macro variables.

The role of macro, as a parameterized unit of SAS code, is then extended by introducing decision making and looping to give the unit more power.

Both authors see lists as a central concept to organizing programs, so lists, how to make them, how to manipulate them, and what they can do for you is a theme that runs throughout the course.

Both iterative and conditional looping are covered since conditional looping is required for handling lists.

To keep things simple as possible no procedures beyond PROC PRINT are used. However, the concept of generating a macro variable from the SELECT statement of PROC SQL is used in a simple form because it is the easiest way to assign a list to a macro variable. Dictionary files are introduced since they are an important way to get and use system information for project management. On the other hand, arrays of macro variables are not introduced. Instead %SCAN is used to get at the elements of a list.

Of course an hour and 15 minutes is much too short to cover even the minimum of our desires, so some of the material we developed will be left to the student for self study.

The Course

Lesson 1:

1) Macro as a parameterized package of SAS code
2) %MACRO and %MEND statements
3) Key word parameter
4) Parameter resolution
5) Macro comment statement
6) Option MPRINT to see generated SAS code

Macro: %tprint
Service: Synchronization of title statement and print

Lesson 2:

1) Enhanced functionality by addition of parameters
   - Control number of observations
   - Better title control
   - Control columns printed
2) Parameter as part of a SAS word
3) %PUT statement for debugging
   - _LOCAL_ as key word to %PUT

Macro: %tprint
New Features: More parameters
Reporting of _LOCAL_ macro variables

Lesson 3:

1) _AUTOMATIC_ variables
   - SYSLAST holds name of last created data set
2) %LET statement to reassign macro parameter value

3) Macro parameter references resolve inside double quotes
4) Macro modification

Macro: %tprint
New Features: Default title problem fixed

Lesson 4:

1) %IF statement to make macro execution decision
   - Debugging
   - Error reporting
2) %LENGTH function to test for existence of parameter value

Macro: %tprint
New Features: More parameters
Better control

Lesson 5:

1) %GOTO statement to control macro flow
2) %UPCASE() function
3) %GLOBAL statement
4) %LET statement to create global macro variables
5) MLOGIC and SYMBOLGEN options for debugging
6) %SYMDEL statement to delete global macro variables

Macro: %tprint
New Features: Test for execution mode

Lesson 6:

1) Macro function %EVAL
   - Does integer/boolean arithmetic
   - Used in %IF
   - Used wherever an integer is expected

Macro: %compare - illustrate use of %EVAL in a %IF statement

Lesson 7:

1) Iterative %DO-loop
   - Using a simple %PUT statement
2) %LOCAL statement
   - Local variables do not exist outside of an executing macro
3) Option MLOGIC

Macro: %loop - illustrate iterative %DO-loop

Lesson 8:

1) Iterative %DO-loop
   - Using PROC PRINT code

Macro: %PrintLoop - illustrate iterative %DO-loop

Lesson 9:

1) %SCAN function
2) %STR function

Lesson 10:

1) Conditional %DO-loop
   - %PUT statement in loop

Macro: %PutAll - illustrate conditional %DO-loop

Note: Use conditional %DO-loops to parse lists. Iterative %DO-loops are less appropriate for such tasks. The structure of the macro code in examples 10, 11, and 12 is identical. This is standard "shell code" for constructing a conditional %DO-loop. The only significant difference among the examples is the code inside of the loop.

Lesson 11:

1) Conditional %DO-loop
   - PROC PRINT code in loop

Macro: %PrintList - illustrate conditional %DO-loop

Lesson 12:

1) Conditional %DO-loop
   - Invoke %tprint in loop
2) Call a helper macro within a macro
3) Pass parameters from outside macro to inside macro

Macros: %PrintList [NEW]- illustrate conditional %DO-loop
%tprint [OLD]

Lesson 13:

1) List creation with PROC SQL INTO: SEPARATED BY
2) SQL Dictionary files

Lesson 14:

1) Supervisor macro %PrintAll to generate list of datasets and then pass the list to %PrintList

Macros: %PrintAll [NEW]
%PrintList [OLD]
%tprint [OLD]
New Features: "Supervisor macro" reads in a control dataset, and invokes another macro

Lesson 15:

1) Example of developing a macro solution for a common task: Converting character variables to numeric.

Macros: %rename - generate a rename list

%CharToNum - generate a list of assignment statements converting data step variables from character to numeric
%DSCharToNum - convert all character variables in a dataset to numeric

Conclusion

Macro is a new programming language for the SAS programmer. It is easy to get off to a bad start. We hope this class will help some to avoid that fate.

The authors may be contacted as follows:

Ian Whitlock
Westat
1650 Research Boulevard
Rockville, MD 20850

Email: IanWhitlock@westat.com

Quentin McMullen
Brown University
169 Angell Street
Box G-S2
Providence, RI 02912

Email: quentin_mcmullen@brown.edu