PLAYING WITH MACROS:
TAKE THE WORK OUT OF LEARNING TO DO MACROS

Arthur L. Carpenter

ABSTRACT

The macro language can add an additional level of complexity to programs written in SAS®. Macro statements are often complicated and may use logic that appears to defy the rules used by all proper SAS programmers. Regardless of the complexities, learning to use the fundamentals of the macro language need not be difficult.

This workshop will take users familiar with the basics of the SAS System and introduce them not only to the basics of the macro language, but also to a process for writing their own macros.

The course will start with macro variables and will examine their differences and similarities to data set variables. Existing code will then be converted to macros and named and positional parameters will be introduced. Examples will include discussion on the resolution of macro variables and the use of the double ampersand (&&). Several macro programming statements, such as, %IF will be introduced along with selected macro functions.

This workshop has been designed to take the work out of learning the SAS macro language.

Terminology

Macro
Stored text that contains SAS statements and macro language statements.

Text
Treated as character strings in the macro language. It consists of variable names, data set names, or any piece of SAS statements.

Macro Variable
Consists of a named single-value. It is treated as a text string.

Macro Program Statement
Controls what actions take place inside the macro processor. They are always preceded by a percent sign (%).

Macro Expression
One or more macro variable names, text, and/or macro functions combined together by one or more operators and/or parentheses.

Macro Function
Predefined routines for processing text in macros and macro variables.

Operators
Symbols that are used for comparisons, logical operation, or arithmetic calculations.

Automatic Macro Variable
Special-purpose SAS Institute provided system variables. These are reserved and should not be used for anything other than their intended purpose.

MACRO VARIABLES

Defining Macro Variables
Macro variables (symbolic variables) are very different from SAS data set variables. Data set variables reside as part of the Program Data Vector associated with a given DATA or PROC step. Macro variables are independent of all data sets and do not depend on either the data set or the observation being processed.

One of the easiest ways to define a macro variable is through the %LET statement: [Macro language statements always start with a %].

%LET dsn = clinics;
Using Macro Variables

PROC CONTENTS DATA=&dsn;
    TITLE 'DATA SET &dsn';
RUN;
PROC PRINT DATA=&dsn (OBS=10);
RUN;

Macro variables appearing inside of a quoted string will not be resolved unless double quotes (" ) are used.

The value of a macro variable may be changed at any time by issuing a new %LET statement. The most recent definition will be used at any given time.

References to Macro Variables

Suffix - A macro variable may be appended to SAS code including variables and data set names.

%LET SEX=MALE;
DATA ONLY &SEX;
SET CLASS.PATIENTS;
WHERE SEX="&SEX";
RUN;

This resolves to:
DATA ONLY MALE;
SET CLASS.PATIENTS;
WHERE SEX="MALE";
RUN;

Prefix - A macro variable may also precede portions of SAS code.

%LET DSN=CLINICS;
%LET DSN1=OLDDATA;
DATA &DSN;
    SET &DSN1 &DSN1;
RUN;

This resolves to:
DATA CLINICS;
    SET CLINICS1 OLDDATA;
RUN;

A double .. (period) is used when a . (period) is desired in the text.

%LET LIBREF=CLASS;
DATA &LIBREF..CLINICS;
    ...
RUN;

This will resolve to:
DATA CLASS..CLINICS;
    ...
RUN;

• More than one macro parameter may be joined to form a single result.

%LET DSN=CLINICS;
%LET N=5;
%LET DSN5=FRED;

Combination Resolves To
&DSN&N CLINICS5
&&DSN&N FRED

USING MACROS

Defining A Macro
Macros are defined using two macro language statements: %MACRO and %MEND. Often the easiest way to create a macro is to surround existing code with these two statements. Every macro definition must begin and end with these two statements, they come in pairs!

%MACRO LOOK;
PROC CONTENTS DATA=CLINICS;
    TITLE 'DATA SET CLINICS';
RUN;
PROC PRINT DATA=CLINICS (OBS=10);
RUN;
%MEND LOOK;

Invoking A Macro
Macros are invoked or called by placing a % in front of the macro name.

%LOOK
Debugging a macro can be, under the best of conditions, difficult. The LOG is often very cryptic with its isolation of error messages. Several options specifically designed for use with macros may be useful during the writing and processing of macros.

Options MPRINT MLOGIC SYMBOLGEN;

- MPRINT prints the macro code, one statement per line, with macro variable references resolved.
- MLOGIC traces the macro logic and follows the pattern of execution. %IF statements are resolved to true or false. Macro invocation, start and finish are noted. This option is especially useful for nested macros.
- SYMBOLGEN causes a message to be printed in the LOG whenever a macro variable is resolved. This option is very useful when tracing macro variable references with multiple & e.g. &&DAT&I.

MACRO PARAMETERS

Defining Parameters
Macros can become more flexible by adding the ability to pass parameters directly into the macro. This avoids getting bogged down in a series of %LET statements. There are two types of parameters positional and keyword (or named).

Adding two positional parameters to the macro %LOOK:

%MACRO LOOK(dsn,obs);
PROC CONTENTS DATA=&dsn;
TITLE "DATA SET &dsn"; 
RUN;

PROC PRINT DATA=&dsn (OBS=&obs);
TITLE2 "FIRST &obs OBSERVATIONS";
RUN;
%MEND LOOK;

Passing Parameter Values
The macro call for %LOOK could be:

%LOOK(CLINICS,10)

All parameters need not be given a value. Alternative invocations of the LOOK macro might include:

%LOOK()
%LOOK(CLINICS)
%LOOK(,10)

Macro variables that are not assigned a value resolve to a null string. Thus the %LOOK(,10) would resolve to:

PROC CONTENTS DATA=;
TITLE "DATA SET ";
RUN;

PROC PRINT DATA=(OBS=10);
TITLE2 "FIRST 10 OBSERVATIONS";
RUN;

The resolved code contains syntax errors and will not run. Care must be taken to construct macro code that will resolve to valid SAS code.

The following macro sorts a data set with one to three BY variables. The macro would not be needed or used if there was not at least one BY variable.

%MACRO SORTIT(DSN,BY1,BY2,BY3);
PROC SORT DATA=&DSN;
   BY &BY1 &BY2 &BY3;
RUN;
%MEND SORTIT;

The macro call

%SORTIT(CLINICS,LNAME,FNAME)
resolves to:

PROC SORT DATA=CLINICS;
   BY LNAME FNAME;
RUN;

Keyword parameters may be named in the macro definition. Keyword parameters may be used in any order and may be assigned default values.

%MACRO LOOK(dsn=CLINICS,obs=); 
PROC CONTENTS DATA=&dsn; 
   TITLE "DATA SET &dsn"; 
RUN; 
%MEND LOOK;

%MACRO LOOK(dsn=CLINICS,obs=); 
PROC CONTENTS DATA=&dsn; 
   TITLE "DATA SET &dsn"; 
RUN; 
%MEND LOOK;
Macro variables that are not assigned a value resolve to their default value or to a null string when no default has been specified. Thus the %LOOK(OBS=10) would resolve to:

```sas
PROC CONTENTS DATA=CLINICS;
  TITLE "DATA SET CLINICS";
RUN;

PROC PRINT DATA=CLINICS (OBS=10);
  TITLE2 "FIRST 10 OBSERVATIONS";
RUN;
```

**PROGRAM CONTROL THROUGH MACROS**

*Macros That Invoke Macros*

It is often advantageous to nest macro calls. Consider the two macros %LOOK and %SORTIT, there is one common parameter (&dsn), and it might be nice to have a utility to do both steps at once.

```sas
%MACRO DOBOTH;
  %SORTIT(CLINICS,LNAME,FNAME)
  %LOOK(OBS=10)
%MEND DOBOTH;

%MACRO LOOK(dsn=CLINICS,obs=);
  PROC CONTENTS DATA=&dsn;
  TITLE "DATA SET &dsn";
RUN;

  PROC PRINT DATA=&dsn (OBS=&obs);
  TITLE2 "FIRST &obs OBSERVATIONS";
RUN;

%MENDLOOK;

%MACRO SORTIT(DSN,BY1,BY2,BY3);
  PROC SORT DATA=&DSN;
  BY &BY1 &BY2 &BY3;
RUN;
%MEND SORTIT;
```

The macro call %DOBOTH(CLINICS,10,LNAME,FNAME) resolves to:

```sas
PROC SORT DATA=CLINICS;
  BY LNAME FNAME;
RUN;

PROC CONTENTS DATA=CLINICS;
  TITLE "DATA SET CLINICS";
RUN;

PROC PRINT DATA=CLINICS (OBS=10);
  TITLE2 "FIRST 10 OBSERVATIONS";
RUN;
```

**Conditional Execution of Macros**

The %IF-%THEN and %ELSE statements are similar to the IF-THEN-ELSE statements used in the data step, except the macro %IF statement is not constrained to a data step.

The macro %DOBOTH in the previous example will fail with a syntax error if there aren't any BY variables passed. It would be nice if the %SORTIT macro was only
executed if at least one BY variable is not blank.

%MACRO DOBOTH(d,o,b1,b2,b3);
%IF &B1 ^= %THEN
  %SORTIT(&d,&b1,&b2,&b3);
%LOOK(&d,&o)
%MEND DOBOTH;

Dynamic Code Building With Macros
The %IF can be used to conditionally insert SAS code into a program.

%MACRO DASTEP(STATE);
DATA SUBHOSP;
SET
%IF &STATE=CA %THEN CAHOSP,
%ELSE AZHOSP;
;
WHERE DATE>'19JUN91'D;
RUN;
%MEND DASTEP;

For %DASTEP(AZ) this resolves to:

DATA SUBHOSP;
SET
AZHOSP
;
WHERE DATE>'19JUN91'D;
RUN;

Macro DO Groups
The %DO and iterative %DO statements in the macro language are very similar to the corresponding statements used in the data step. Like the %IF, however, these statements are not confined to the data step. They can be used anywhere inside of a macro. Each form of the %DO statement must be matched with an %END.

The simplest form is the %DO block.

%MACRO DATASET(VALUE);
%IF &VALUE = 1 %THEN %DO;
  PROC SORT DATA=PATIENTS;
  BY SYMPTOM;
  RUN;
  PROC PRINT DATA=PATIENTS;
  BY SYMPTOM;
  RUN;
%END;
%ELSE %DO;
  PROC FREQ DATA=ICECREAM;
  TABLES TYPE;
  RUN;
%END;
%MEND DATASET;

The macros %SORTIT and %LOOK could easily be combined into %DOBOTH by using conditional processing with the %DO block.

%MACRO DOBOTH(dsn,obs,by1,by2,by3);
%IF &BY1 ^= %THEN %DO;
  PROC SORT DATA=&DSN;
  BY &BY1 &BY2 &BY3;
  RUN;
%END;
%MACRO ALLYR(START,STOP);
%DO YEAR = &START %TO &STOP;
  DATA TEMP;
  SET YR& YEAR;
  YEAR = 1900 + &YEAR;
  RUN;
%MEND ALLYR;

Iterative %DO
The form of the iterative %DO is similar to the DO statement except; the WHILE and UNTIL clauses cannot be added and irregular increments are not allowed. The incremental variable is not a SAS variable but rather a new macro variable.

%MACRO ALLYR(START,STOP);
%DO YEAR = &START %TO &STOP;
  DATA TEMP;
  SET YR& YEAR;
  YEAR = 1900 + &YEAR;
  RUN;
%MEND ALLYR;
PROC APPEND BASE=ALL YEAR
  DATA=TEMP;
RUN;
%END;
%MEND ALL YR;

The call %ALL YR(88,90) produces:

DATA TEMP;
SET YR88;
YEAR = 1900 + 88;
RUN;
PROC APPEND BASE=ALL YEAR
  DATA=TEMP;
RUN;

DATA TEMP;
SET YR89;
YEAR = 1900 + 89;
RUN;
PROC APPEND BASE=ALL YEAR
  DATA=TEMP;
RUN;

DATA TEMP;
SET YR90;
YEAR = 1900 + 90;
RUN;
PROC APPEND BASE=ALL YEAR
  DATA=TEMP;
RUN;

SUMMARY

Although there is a lot to learn in the macro language, one
does not need to know how to %do it all in order to %put
%macros to use. Start simple, %ifyou have the chance
try new statements, and above all %else have fun.

Much of the material in this paper is taken from a book
on the SAS macro language which is being written by Art
Carpenter and is due to be published in 1998.

ABOUT THE AUTHOR

Art Carpenter’s publications list includes the book Quick Results with SAS/GRAPH® Software, two chapters in Reporting from the Field, and over two dozen papers and posters presented at SUGI and WUSS. Art has been using SAS since 1976 and has served as a steering committee chairperson of both the Southern California SAS User's Group, SoCalSUG, and the San Diego SAS Users Group, SANDS; a conference cochair of the Western Users of SAS Software regional conference, WUSS; and Section Chair at the SAS User’s Group International conference, SUGI.

Art is a SAS Quality Partner™ and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

Art Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199

(760) 945-0613
72212.211@compuserve.com

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries. ™ indicates USA registration.