The SAS Macro Language

Joy Reel

The purpose of the SAS macro language tutorial is to give an overview of the language and its capabilities, and to provide some possible uses pertinent to your environment. The last part of the tutorial is a small-scale interactive system made up of a series of macros to illustrate using the language to build user friendly systems for local use.

In general a macro language is a preprocessor. It does its work and passes the results to a compiler or another processor. A single macro is a set of statements that produces as its results some sort of textual output, for example, character strings.

The SAS macro language uses programming statements, keywords, and techniques which very closely resemble the basics of the SAS language itself. The major difference in the two are the identifying % and & used in the macro language. The results of a SAS macro are character strings which form SAS statements or pieces of SAS statements. The SAS macro language uses macro variables, also called symbolics, to alter the generated text from one execution of the macro to another or to provide needed information to the macro.

There are many possible uses of the macro facility in SAS. You can replace data set names from run to run by using symbolics or possibly to build a data set name based on different criteria. You can do these things through text replacement and/or text concatenation.

Many users have requested the capability to conditionally execute DATA and PROC steps within SAS, and the macro language offers that ability. The macro programming statements allow a user to test and select steps based on user defined or automatic macro variable values. The actual selection process is done before the code is sent to the SAS compiler and therefore no SAS rules are broken using this approach.

You can define parameters with the macro with default or null values. With symbolic substitution the macro execution can produce different results at different times.

With the ability of the macro language to talk to the user and accept his response from a terminal, user friendly interactive systems can be written for use by those who do not know SAS at all.
The core of the system is a selection of one of four possible reports - two using PROC PRINT and two using PROC CHART. RUNIT only allows a user one pass if he chooses correctly the first time, but other looping within the RUNIT macro is possible.

Note my use of %PUT as a debugging tool in the &choice=1 section. Note also that the SAS compiler sees only one procedure step. The condition is met within the macro and the macro results pass to compiler as only one PROC step.

This little mini system is very simple to construct, and uses very basic macros with only one function each. However, the idea can be expanded to write very sophisticated systems which can lead users with no SAS experience to construct SAS programs by responding to questions.
A MACRO

%MACRO LOOP;
%DO I = 1 %TO 10;
   DATA DATA&I;
   DO X = 1 TO 10;
      Y = &1 + X;
      OUTPUT;
   END;
%END;
%MEND LOOP;

THE CALL

%LOOP

THE RESULTS

SAS sees

DATA DATA1;
   DO X = 1 TO 10;
      Y = 1 + X;
      OUTPUT;
   END;
DATA DATA2;
   DO X = 1 TO 10;
      Y = 2 + X;
      OUTPUT;
   END;
DATA DATA3;
   DO X = 1 TO 10;
      Y = 3 + X;
      OUTPUT;
   END;

... 

DATA DATA10;
   DO X = 1 TO 10;
      Y = 10 + X;
      OUTPUT;
   END;

This is what a SAS macro definition looks like. The important words here are %MACRO and %MEND which enclose the macro definition. Notice the use of the same syntax in the DO loops. The macro programming language keywords are identical to SAS key words except that they begin with a %.

Note that the macro call uses a % in front of the macro name. It does not include a semicolon at the end.

The result of the execution of macro LOOP indicates that the SAS compiler receives ten data steps as defined by the %DO loop. Note also that if the upper limit had been 1000 instead of 10, the macro would have generated 1000 data steps to be compiled and executed by the SAS compiler. Be careful.
An important concept of the macro language is that of open code versus macro code. SAS statements themselves, macro calls, and some of the macro keywords such as %LET are used in open code. Most macro keywords are used in macro code only, that is, inside a macro definition. For example, %IF ... %THEN ...; may only be used inside a macro, not in open code.

Using a macro variable or symbolic statement in a TITLE statement is a common application of the macro facility. However, it is important to realize that symbolic substitution is done at compile time. In this example, failure to repeat the TITLE2 statement results in using the original value for &month. The actual value of month changed, but substitution had already been done in the TITLE2 statement itself.
/

THIS EXAMPLE ILLUSTRATES THE USE OF SYMBOLICS IN A TITLE STATEMENT. NOTE THAT THE TITLE STATEMENT USING THE SYMBOLIC MUST BE REPEATED WHEN THE VALUE OF THE SYMBOLIC IS CHANGED.

DATA TEST;
   DO X=1 TO 5;
      OUTPUT;
   END;
RUN;

%LET NUMBER=FIRST;
%LET PAGE=1;

TITLE1;
TITLE2 THIS IS MY &NUMBER EXAMPLE -- PAGE &PAGE;
RUN;

PROC PRINT;
RUN;

%LET NUMBER=SECOND;
%LET PAGE=%EVAL(&PAGE+1);

TITLE2 THIS IS MY &NUMBER EXAMPLE -- PAGE &PAGE;
RUN;

PROC PRINT;
RUN;

Old Style Macro

MACRO KEEPLIST NAME ADDRESS1 ADDRESS2 CITY STATE ZIP; %
Reference KEEPLIST

NEW WAY

%LET KEEPLIST=NAME ADDRESS1 ADDRESS2 CITY STATE ZIP;
Reference &KEEPLIST

The next example shows the correct way to handle the problem. Since TITLE2 is repeated the second PROC PRINT output will show the current value of the macro variables in the title.

Since many users have old macro definitions in current production code, there needs to be a simple way of converting to the new macro facility. The new macro facility is much more efficient and certainly more flexible for future enhancements to user code. The easiest way to convert from old to new is to use the %LET statement as in this example. If the old style macro consists of more than one statement, then the entire old macro can be enclosed inside a %STR statement so that a semicolon does not signal the end of the %LET.

In addition all references to the macro must be changed from the name alone to the name preceded by &.
There is not yet available the ability to format a symbolic variable with a SAS or user defined format. This example illustrates a method of assigning a formatted value to a macro variable so that it will display in the desired format.

There is probably a better example in the little interactive system at the end of the tutorial.

/* THE SORTIT MACRO PERFORMS A SORT WITH THE BY VARIABLE SPECIFIED ON THE MACRO CALL */

%MACRO SORTIT(KEY);

PROC SORT;
  BY &KEY;
%MEND SORTIT;

We are going to look at some very simple macro definitions to get the feel of the macro language. The SORTIT macro is about as simple as they get, but it does illustrate the use of a positional parameter. It could be used if a user needed to sort a data set several times using a different BY variable each time. Even here we can see that it is easier to key %SORTIT(NAME) or %SORTIT(ID) than the whole PROC step, and there is less chance for user error.
The CHOOSE macro illustrates the use of keyword parameters defined with specific default values. It also shows error checking of a passed parameter value. The macro call without parameters

\%CHOOSE

would use the default values as defined on the \%MACRO statement.

There are a number of useful automatic symbolic variables supplied with the macro facility to facilitate its use. The SETUP macro checks the SYSSCP variable to determine if the user is running under CMS or CS. If neither is the case, a message is written to the SAS log. Another similar and perhaps more useful symbolic is the SYSENV which tells whether one is in foreground(interactive) or background(batch). See the SAS USERS GUIDE: BASICS for more of these. All the automatic symbolics are global variables.
/** This macro writes a number of blank lines to the terminal screen depending on the value of the lines parameter. */

%do i=1 %to &lines;
    %put %str( );
%end;

%MEND SPACES;

/** This macro checks for a null data set. If it finds one, it sets a flag and stops. If the data set is not empty, it sets the flag to a different value and creates a symbolic to be used later in a title. */

data _null_
    do i=1;
        if _n_=1 and nobs=0 then do;
            call symput('eof','y');
            stop;
            end;
        else do;
            call symput('eof','n');
            set input.&which point=i nobs=nobs;
            call symput('date',put(month,date7.));
            stop;
            end;
    end;
run;
%MEND CHECK;

The SPACES macro has a very trivial function. It writes a literal blank to the SAS log. Because of the single %PUT and the literal blank, it actual writes a blank line. The number of blank lines it writes is determined by the value of the position parameter LINES. We will use this later on to make screen as pretty as possible.

The CHECK macro looks at a data set to see if it is empty. It uses the data step function SYMPUT to set a flag with the appropriate value. If the data set is not empty, it also sets up a symbolic for later use in a title.
%MACRO ERROR;
  %* This macro is called if the user does not enter a valid selection from the list. If it is his first mistake, he gets another chance. Otherwise, he is told to try later.
  %SPACES(3);
  %IF &time=1 %THEN %DO;
    %PUT You are not paying attention.;
    %PUT Perhaps you should try again later.;
    %LET bad=1;
    %END;
  %ELSE %DO;
    %LET bad=0;
    %PUT You did not follow instructions.;
    %PUT Would you like to try again? Y or N? ;
    %INPUT again;
    %IF %UPCASE(&again)=Y %THEN %RUNIT;
    %ELSE %LET bad=1;
    %END;
%MEND ERROR;

%MACRO BUFFER;
  %* This macro checks the automatic symbolic variable SYSBUFFER to see if the user entered more information than the prompting macro requested. If so, a message is displayed and execution continues.
  %SPACES(3);
  %IF &SYSBUFFER ~ = %THEN %DO;
    %PUT Extra information on input ignored.;
    %PUT Execution will continue.;
    %PUT Please be careful to follow instructions.;
    %END;
%MEND BUFFER;

%MACRO BAD_RTCD;
  %* This macro tests for a good return code after execution of a macro or the macro language service routines. In this example it is used to make sure a file allocation was successful.
  %SPACES(3);
  %IF &SYSRC ~ = 0 %THEN %DO;
    %PUT The file you have chosen is not currently available.;
    %LET bad=1;
    %END;
  %ELSE %LET bad=0;
%MEND BAD_RTCD;
%MACRO TALKIT;

%* This macro displays the primitive menu for the user to make his selection and accepts his choice.:;

%SPACES(3)
%PUT The following monthly reports are available for you to run;

%SPACES(3)
%PUT 1. A chart of monthly sales by salesman.;

%SPACES(2)
%PUT 2. A year-to-date chart of sales by salesman.;

%SPACES(2)
%PUT 3. A regular monthly report of sales by salesman.;

%SPACES(2)
%PUT 4. A regular monthly report of year-to-date sales by salesman.;

%SPACES(3)
%PUT Please enter the number of your choice.;

%INPUT choice;

%MEND TALKIT;

The TALKIT macro calls the SPACES macro to write blank lines to the SAS log and gives the user information from which to select certain SAS steps. It provides a rather primitive menu. It also accepts the user's choice from the terminal. Please note that %INPUT does not work in batch.

%MACRO RUNIT;

%* The RUNIT macro combines all the macros to produce the desired results;

%GLOBAL time eof bad choice date;

%TALKIT

%BUFFER

%TSO ALLOC f(INPUT) da('JOY.MYLIB') SHR;

%BAD_RTCD

%IF &bad=1 %THEN %GOTO ENDIT;

The RUNIT macro is the driver macro which puts together the last six examples to build a simple interactive system. The flags which are used in several of the macros are gloaled and each macro is called according to its function. The flags are tested in the RUNIT macro so that processing can be halted if something is not right by jumping to a label at the end of the macro.