Macros: Code Cutters or Code Generators?
Prashant Mittal, Muskie School of Public Service, Portland, ME

ABSTRACT

Macros have been revered and feared as a collection of eccentric and mystic symbols scattered all over the SAS® Editor window. On the contrary, they are a straightforward and most powerful set of tools. SAS programmers often have to write repetitive code, which includes the same PROCS and data steps written multiple numbers of times. This redundancy makes the code boring to read and cumbersome to implement. There are a few secret ways of using Macro-%do loops, global variables indirect referencing, and the _Null_ statement, whose proper usage could force SAS to cut more than thousands of lines of repetitive code to less than fifty lines of easy to understand statements.

INTRODUCTION

Somebody has very well said that “necessity is the mother of invention”. Almost four years ago, when I was trying to establish a relationship between Uniform and Normal distributions, I was running simulations on tens of thousands of datasets from both distributions. The whole process required a lot of my time and attention. I desperately needed an easier way out. It was then that I learnt Macros. I suddenly discovered a hidden world of code which worked like functions, and which also helped me generate SAS code without actually having to write it.

I define macros as “SAS code generators” or sometimes as “general solutions to many problems”. Unlike regular SAS code, macros can handle multiple similar tasks by using a small amount of code. It requires macro elements to make a general solution to a class of related problems. Very few people have realized that with the advent of macros, SAS has truly become an Object Oriented Programming (OOP) language. One factor which plays a major role in defining the final framework of SAS as an OOP is the portability of Macros. The same macro could be used as a function in two completely different programs. This involves simple referencing of a macro from a bigger pool or library of macros.

This paper is an attempt to educate and inform readers about the strengths, flexibilities and utilities of macros.

MACROS: A FIRST LOOK

Fig. 1 shows the general framework of a macro. Every macro must have a name which works as an identifier for that macro (or one may call it a function). An analogy is the sum function in SAS. Sum function adds up the non missing values of all its parameters or variables. Similar to the sum function (or any other function in general), it’s advisable to give a meaningful name to a macro, depending upon its functionality or purpose.

```
Basic framework of a macro

%macro macro_name;
...
(Regular SAS data steps, procedure statements etc)
...
%mend macro_name;
% macro_name;
r;
```

```
Macro with parameters

%macro xyz (a = , b = , c = );
data any;
set any1;
newvar1 = &a;
newvar3 = &b;
newvar3 = &c;
r;
%mend xyz;

%xyz (a = var1, b = var2, c = var3);
%xyz (a = var4, b = var5, c = var6);
%xyz (a = var7, b = var8, c = var9);
```

As Fig. 1 shows, a macro should always start with a %macro statement and should always end with %mend statement. Dotted lines symbolize regular SAS statements, for example data steps, procedures etc. Dotted lines actually define the purpose and functionality of this macro. Fig. 2 shows another example of a macro which has three parameters (or variables) a, b and c. Notice that the variables a, b, c are given the values var1 var2 var3 etc. when the macro xyz is run at the end of the program. This technique is called indirect referencing. As I mentioned before macros work as functions and functions have
parameters, which vary in values. Once a macro is defined, its parameters (a, b, c etc) can take values from other variables in a dataset.

MACRO VARIABLES and THEIR SCOPE

Tools that enable dynamic modification of the text in a SAS program through symbolic substitution are called macro variables. Macro variables are by default character and can be converted to numeric for calculations. Macro variables can be either user defined or automatic. Also their scope can be global or local. Global variables can be defined and used anywhere in the program, whereas local variables can only be used in a macro in which they are defined. To get descriptions of all macro variables (even if the programmer has not created a macro variable, there always exists a number of useful macro variables in SAS session) that exists at a particular time the following code works well.

%Put {_all_ | _automatic_ | _global_ | _local_ | _user_}; run;

To write instructions in SAS log, the following code could be used:

%Put %str (Check LOG carefully this point onwards)

The above code will print Check LOG carefully this point onwards in SAS log. This is a good way of adding instructions and comments along with the run code. A similar use of automatic variable would be:

%Put ‘Report for &sysday, &sysdate’ generates the following in SAS log: Report for Thursday 02APR04, assuming today’s date is April 2, 2004.

CREATING NEW MACRO VARIABLES

A new macro variable, global or local, can be created using the statement %let in the following manner.

%let macro_variable = This is it;
%put &macro_variable;

The above two lines will produce This is it in SAS log.

Here is another example of creating a global variable and using it in the data step. Remember global variables are always created in the open code (outside of a macro).

<table>
<thead>
<tr>
<th>Obs</th>
<th>i</th>
<th>var</th>
<th>newvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.83883</td>
<td>Example</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.82309</td>
<td>Example</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.20670</td>
<td>Example</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.48918</td>
<td>Example</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.56022</td>
<td>Example</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.44423</td>
<td>Example</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.83851</td>
<td>Example</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.36627</td>
<td>Example</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0.18782</td>
<td>Example</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.91411</td>
<td>Example</td>
</tr>
</tbody>
</table>

Fig. 3      Fig. 4
Program in Fig. 3 generates a global macro variable X which is equal to a text string ‘Example’. This program generates a dataset called new. This data set contains 10 observations and three new variables: i, var and newvar. The program then generates random numbers between 0 and 1 for the variable var. But most importantly, it uses indirect referencing to give a value ‘Example’ to the variable newvar for all 10 observations. This can be seen in Fig. 4.

The following example shows another way of creating macro variables by using call symput statement.

<table>
<thead>
<tr>
<th>data team2;</th>
<th>Newly created macro variables are: batter</th>
</tr>
</thead>
<tbody>
<tr>
<td>input position : $12. player $12.;</td>
<td>, pitcher , frstbase</td>
</tr>
<tr>
<td>call symput ('POS'</td>
<td></td>
</tr>
<tr>
<td>cards;</td>
<td>19:26 Thursday, July 29, 2004</td>
</tr>
<tr>
<td>batter Ann</td>
<td>Obs position player new</td>
</tr>
<tr>
<td>pitchet Tom</td>
<td>1 batter Ann batter</td>
</tr>
<tr>
<td>frstbase Bill</td>
<td>2 pitcher Tom batter</td>
</tr>
<tr>
<td>;</td>
<td>3 frstbase Bill batter</td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data team2;</th>
<th>Fig. 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>set team2;</td>
<td></td>
</tr>
<tr>
<td>new = &quot;&amp;pos1&quot;;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>proc print data = team2;</td>
<td></td>
</tr>
<tr>
<td>Title &quot;Newly created macro variables are: &amp;pos1,</td>
<td></td>
</tr>
<tr>
<td>&amp;pos2, &amp;pos3&quot;;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

**REPETITIVE CODE USING MACROS**

Programmers often have to write repetitive code (multiple PROCS, Data steps etc.). This is usually done to understand the nature and complexity of a dataset or to write various reports. This includes procedures and data steps written multiple times, making the code redundant and boring to read. With proper usage of macros, a long program of repetitive code can be cut down dramatically by forcing SAS to generate code by itself.

**Case I:**

Here is the simplest example of all. Consider running multiple PROC MEANS on various variables using multiple class statements as shown below:

<table>
<thead>
<tr>
<th>Proc means n min max mean sum median std P50 data = any;</th>
<th>How wonderful would it be to use a do loop to handle these repetitive procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var charg1 Paid1;</td>
<td></td>
</tr>
<tr>
<td>Class elig1;</td>
<td></td>
</tr>
<tr>
<td>Run;</td>
<td></td>
</tr>
<tr>
<td>Proc means n min max mean sum median std P50 data = any;</td>
<td></td>
</tr>
<tr>
<td>Var charg2 Paid2;</td>
<td></td>
</tr>
<tr>
<td>Class elig2;</td>
<td></td>
</tr>
<tr>
<td>Run;</td>
<td></td>
</tr>
<tr>
<td>Proc means n min max mean sum median std P50 data = any;</td>
<td></td>
</tr>
<tr>
<td>Var charg3 Paid3;</td>
<td></td>
</tr>
<tr>
<td>Class elig3;</td>
<td></td>
</tr>
<tr>
<td>Run;</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7
Fig. 7 shows a classic example of a cumbersome and repetitive code. The program above is trying to get descriptive statistics on variables like charg1, paid1, charg2, paid2 by elig1 and elig2 respectively. (Here charg, paid and elig variables can be defined as Medicaid charges, actual payments and whether somebody is eligible for Medicaid or not). This list of Proc Means statements sometimes goes on for quite long. An easier and much more efficient way of handling this or a similar situation would be to use a “do-loop”. This is a great misconception among programmers that do-loops are restricted only to data steps. Instead, do-loops can always be written in the open code using a macro. Here is how it is achieved:

```sas
%macro easy;
%do i = 1 %to 15;
   proc means n mean sum median std data = any;
   var charg&i Paid&i;       * Indirect referencing;
   class elig&i;
   run;
%end;
%mend;
%easy;
run;
```

Fig. 8

The macro “Easy” in Fig. 8 uses a do loop to handle 15 Proc Means statements, similar to the ones in Fig. 7. It would take more than 60 lines of code to write an equivalent program in regular SAS. Notice that ‘do’, ‘to’ and ‘end’ keywords start with the % sign in a macro. Also & sign is used for indirect referencing. For example, to give a different value each time to ‘i’ an & sign is used with i. Let’s look at another similar example with a little more usage of indirect referencing.

**Case II:**

```sas
proc sort data = any;
   by elig1;
run;
proc freq data = any;
   table chg1*paid1;
   by elig1;
   title' Cross tabulation between chg1 and paid1';
run;
proc sort data = any;
   by elig2;
run;
proc freq data = any;
   table chg2*paid2;
   by elig2;
   title' Cross tabulation between chg2 and paid2';
run;
.
.
```

Fig. 9

```sas
The previous repetitive code of more than 150 lines could be written as follows:

%macro easy;
%do i = 1 %to 15;
   proc sort data = any;
   by elig&i;
   proc freq data = any;
   table chg&i*paid&i;
   by elig&i;
   title' Cross tabulation between chg&i and paid&i';
   run;
%end;
%mend;
%easy;
run;
```

Fig. 10

Figure 9 shows a similar situation where multiple cross tabulations are performed by different class variables. Unfortunately, Proc Freq does not allow class statement; otherwise this would have been exactly same case as the Proc Means example above. Also in this case, the title statement is to be updated every time a new pair of variables is cross tabbed. If coded the regular way it will take approximately 150 lines of code to complete this program. An easier solution is given in Fig. 10. Through indirect referencing the title statement is updated every time the do loop runs a new iteration. Also notice the use of & sign each time an indirect reference is done.
Case III:

The previous two examples had one common characteristic. All variables used were sequential and ended in a number. That made them extremely easy to use in a do loop. In day to day programming, we don’t always deal with such easy and clear situations. Most of the time, variable names are not sequential and have nothing common in their names. There are two ways of handling such situations. The first one is to create new sequential variables using an array in a data step and go ahead with the techniques described above. The second way which is more advanced and easy to use is as follows:

Suppose one wants to have descriptive statistics for the variables in line 1 by variables in the line 2 given below. Clearly the variable names are not ending in a number and so are not sequential.

<table>
<thead>
<tr>
<th>Line 1: A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 2: V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

The regular and the macro way of coding are shown in figures 13 and 14 respectively.

```sas
Proc means mean median std data = any;
Var A;
Class V;
Title 'Summary statistics of A by V';
Run;
Proc means mean median std data = any;
Var B;
Class W;
Title 'Summary statistics of B by W';
Run;
Proc means mean median std data = any;
Var C;
Class X;
Title 'Summary statistics of C by X';
Run;
Proc means mean median std data = any;
Var D;
Class Y;
Title 'Summary statistics of D by Y';
Run;
Proc means mean median std data = any;
Var E;
Class Z;
Title 'Summary statistics of E by Z';
Run;.
```

```sas
%macro easy (var1 =, var2 =);
Proc means mean sum median std data = any;
Var &var1;
Class &var2;
Title 'Summary statistics of &VAR1 by &VAR2';
Run;
%mend;
%easy(var1 = A, Var2 = V);
%easy(var1 = B, Var2 = W);
%easy(var1 = C, Var2 = X);
%easy(var1 = D, Var2 = Y);
%easy(var1 = E, Var2 = Z);
run;
```

This example is different from the other two in several ways. Notice that in Fig. 12, macro name easy is defined as a function for the first time. This macro (or function) easy has two parameters var1 and var2. These two parameters work as dummy variables and are later used for indirect referencing. In other words, these two variables can be used to feed values from many other variables. Once a macro is written (its purpose and functionality are defined) and run, one does not need to run the macro statements again. Only the referencing of the macro name along with the new variables is enough. For example in the above case, running the following code is enough to force SAS to create code by itself.

```sas
%easy (var1 = A, Var2 = V); %easy (var1 = B, Var2 = W); %easy (var1 = C, Var2 = X);
```

Readers who are comfortable with the concept of functions will find similar macros very helpful and easy to use.
NULL_STATEMENT AND MACROS

Programmers often encounter situations where they actually need only a small portion of the output for their reports and other purposes, out of the oodles of output SAS generates from each procedure (PROC). Consider an example where the experimenter is looking to get only the chi-square values of cross tabulations of a variable say gender, with 10 other variables (B C D E F G H I J K). The final goal is to get a table of chi-sq values and p-values along with the variables information, which looks somewhat like the following:

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>PCHI</em></th>
<th>DF_PCHI</th>
<th>P_PCHI</th>
<th>vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.78754</td>
<td>1</td>
<td>0.18123</td>
<td>gender-B</td>
</tr>
<tr>
<td>2</td>
<td>0.66222</td>
<td>1</td>
<td>0.41578</td>
<td>gender-C</td>
</tr>
<tr>
<td>3</td>
<td>4.02022</td>
<td>1</td>
<td>0.04496</td>
<td>gender-D</td>
</tr>
<tr>
<td>4</td>
<td>0.04271</td>
<td>1</td>
<td>0.83627</td>
<td>gender-E</td>
</tr>
</tbody>
</table>

Here _Pchi_ is the Pearson’s chi-square coeffient, DF_PCHI is the degrees of freedom, P-PCHI is the p-value for the chi-square and Vars is the information on two variables which are cross tabbed.

Regular code will be written the following way:

```sas
proc freq data = any;
table gender*( B C D E F G H I J K) /CHISQ EXACT;
run;
```

This will generate 10 pages of output and we only need three or four numbers out of each page. Each page looks like the output given in Fig. 13.

Notice that only the shaded part in the output attached is required. This whole process will involve picking out four to five numbers from each page and manually filling in on a spreadsheet to make a report table. With the help of Macros along with the _Null_ dataset, SAS could easily be forced to generate this kind of summary table by itself. The regular way of generating this table without using a macro would be:

```sas
data final;
set _null_
;
proc freq data = any nocprint;
table gender*B/ chisq;
output PCHI out = a ;
data a;
set a;
vars = "gender-B";

data final;
set final a;
run;
```

The Chi-square values of various cross tabulations in a dataset
The FREQ Procedure
Table of GENDER by B
GENDER(GENDER) B(B)

<table>
<thead>
<tr>
<th>Col Pct</th>
<th>Less than 1000</th>
<th>1000+</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>217</td>
<td>79</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>44.47</td>
<td>16.19</td>
<td>60.66</td>
</tr>
<tr>
<td></td>
<td>73.31</td>
<td>26.69</td>
<td>58.97</td>
</tr>
<tr>
<td></td>
<td>58.97</td>
<td>65.83</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>151</td>
<td>41</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>30.94</td>
<td>10.80</td>
<td>39.34</td>
</tr>
<tr>
<td></td>
<td>78.65</td>
<td>21.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>41.03</td>
<td>34.17</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>368</td>
<td>120</td>
<td>488</td>
</tr>
<tr>
<td></td>
<td>75.41</td>
<td>24.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frequency Missing = 512

Statistics for Table of GENDER by B

<table>
<thead>
<tr>
<th>Statistic</th>
<th>DF</th>
<th>Value</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-Square</td>
<td>1</td>
<td>1.7875</td>
<td>0.1812</td>
</tr>
<tr>
<td>Likelihood Ratio Chi-Square</td>
<td>1</td>
<td>1.8094</td>
<td>0.1786</td>
</tr>
<tr>
<td>Continuity Adj. Chi-Square</td>
<td>1</td>
<td>1.5714</td>
<td>0.2189</td>
</tr>
<tr>
<td>Mantel-Haenszel Chi-Square</td>
<td>1</td>
<td>1.7839</td>
<td>0.1817</td>
</tr>
<tr>
<td>Phi Coefficient</td>
<td></td>
<td>-0.0605</td>
<td></td>
</tr>
<tr>
<td>Contingency Coefficient</td>
<td></td>
<td>0.0604</td>
<td></td>
</tr>
<tr>
<td>Cramer’s V</td>
<td></td>
<td>-0.0605</td>
<td></td>
</tr>
</tbody>
</table>

Fisher’s Exact Test

| Cell (1,1) Frequency (F) | 217 |
| Left-sided Pr <= F      | 0.1090 |
| Right-sided Pr >= F     | 0.9264 |
| Table Probability (P)   | 0.0064 |
| Two-sided Pr <= F       | 0.1017 |

Effective Sample Size = 488
Frequency Missing = 512
WARNING: 51% of the data are missing.
Now once the algorithm is ready, it can easily be converted to a concise macro program as follows.

```sas
data final;
set final a;
run;
```

Now once the algorithm is ready, it can easily be converted to a concise macro program as follows.

```sas
data final;
set _null_

%macro chisq(var1 =);
proc freq data = any noprint;
table gender*&var1/chisq;
output PCHI out = a;
data final;
set final a;
vars = "gender- &var1";
run;
%mend;

%chisq(var2 = B); %chisq(var2 = C); %chisq(var2 = D); %chisq(var2 = E); %chisq(var2 = F); %chisq(var2 = G);
%chisq (var2 = H); %chisq (var2 = I); %chisq (var2 = J); %chisq (var2 = K);
run;
```

Fig. 14

CONCLUSION
The discussion started with the question of whether a Macro is a code cutter or a code generator. Readers, by now must have understood that a macro is actually both. It cuts the amount of code written by generating the code itself. Macro is in fact a way of programming, a habit and a great tool which becomes easier to use with practice.

ACKNOWLEDGMENTS:
SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.
SAS Macro Language, Reference, Version 8
SAS Online Doc, Versions 8 and 9.

CONTACT INFORMATION
Author Name: Prashant Mittal
Institution: Muskie School of Public Service
Address: 509 Forest Ave.,
         Muskie School of Public service
City State Zip: Portland, ME-04106
Work Phone: 207-228-8145
Email: p mittal@usm.maine.edu
       prashant_mittal@hotmail.com

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.