CMU MSP 36-602: SAS Macros II

Howard Seltman, April 3, 2019

1) ***Writing your own macro*** is equivalent to defining a new macro function.

   a. **Syntax:**

```
%MACRO myMacroName[(myArgName1[=[myDflt1]][,myArgName2[=[myDflt2]][…]])];
  ---any text lines---
%MEND [myMacroName];
```

   Use of "=" defines **keyword** as opposed to **positional parameters**. The former may be entered in any order, but must include the name and "=" at invocation. All positional parameters must precede all keyword parameters. Default values are optional. The parameters are "local" macro variables that can be accessed with &.

   The ***purpose*** of every macro is to ***generate text*** that will be submitted to the SAS system, generally DATA and PROC steps. Smaller text outputs, e.g., a set of variable names are also possible. Side effects, e.g., printing information may also be part of the goal of a given macro.

   b. **Macro Example 1** (using the "prepost" dataset from Macros1.pdf):

   /* simple macro "function" to compute means for one variable */

```
%MACRO periodMean(period, preOrPost);
  PROC MEANS DATA=prepost;
    VAR &preOrPost&period;
  RUN;
%MEND periodMean;
%periodMean(Q2, pre)     /* note no need for a semicolon */
%periodMean(H1, post)
```

   c. **Macro Example 2:**

   /* regression with EDA using data set "ds" and specified x and y variables */

```
%MACRO regPlus(ds=, xvar=X, yvar=Y);
  TITLE "Regression of &yvar on &xvar";
  PROC SGPLOT DATA=&ds;
    REG X=&xvar Y=&yvar;
  RUN;
  PROC GLM DATA=&ds PLOTS=(DIAGNOSTICS);
    MODEL &yvar = &xvar;
  RUN;
  QUIT; /* because REG is an interactive PROC */
  TITLE; /* turn title off */
%MEND regPlus;
%regPlus(ds=prePost, yvar=preH2, xvar=preQ2)
```

d. **Macro Example 3:**

```
/* Demonstration that invoking macros just substitutes their text */
%MACRO dsname;
  prePost
%MEND dsname;
PROC PRINT DATA=%dsname;
RUN;
```

Or more confusingly:

```
%MACRO dsnameSem;
  prePost;
%MEND dsnameSem;
PROC PRINT DATA=%dsnameSem
RUN;
```

2) **Saving macros across sessions**

   a. **Macros loaded into a session in the usual way are lost at the end of the session** and need to be reloaded if you want to use them in a future session.

      i. Note that you can use `%INCLUDE "myFile.sas";` at any time to load code, including macros, from a file.

      ii. Such macros are stored in the "work" library, which is automatically deleted at the end of the session

      iii. You can open the "work" library in the SAS Explorer window, where you will see the "Sasmacr" folder if any macros are defined. Open this folder to see the names (but not contents) of the currently defined macros.

   b. **Method 1: Storing macros in a macro library in text form (autocall library)**

      i. Important code safety tip: Use the `%LOCAL var1 [var2…];` statement to avoid clashes with user-defined macro variable names or across macros. Typically, the first line of a macro is `%LOCAL var1 var2 ...;` listing all macro variables created in the macro (except the macro arguments, which are local by default). This assures that variables defined elsewhere are not accidentally used or changed.

      ii. **Store each macro in a separate plain text file whose name matches the macro name plus the extension ".sas".**

      iii. To ***setup*** to access the code in the macro text files for any SAS session, use code like this:

```
FILENAME myMacDir "myDirName";  /* not LIBNAME! */
OPTIONS SASAUTOS = (myMacDir) MAUTOSOURCE;
```

      As usual, the `'myMacDir'` does not need to be the same across sessions, but the directory name must match where the macro file(s) are stored. Note that any number of directories containing any number of .sas files can be included inside the parentheses.

      iv. Now just ***call*** any of the macros in your "myDirName" directory as if they had been defined in the current session.

      v. This method has some overhead: each macro needs to be "compiled" the first time it is used in each SAS session.

      vi. Including the keyword "SASAUTO" inside the parentheses will give you access to some built-in macros (search help for "Selected Autocall Macros").

vii. Example of use of autocall macros:

```
/* Assume hello.sas is in the folder pointed to by mLoc */
OPTIONS SASAUTOS = (mLoc) MAUTOSOURCE;

%hello(Howard, 1954)
```

c. **Method 2: Storing macros in a macro library in compiled form (a type of SAS catalog)**

i. Important code safety tip: Use the %LOCAL var1 [var2…]; statement to avoid clashes with user-defined macro variable names or across macros.

ii. The ***setup*** to either compile and store macros or to use them is the same:

```
LIBNAME myMacDir "&wd/myMacros"; /*LIBNAME, not FILENAME*/
OPTIONS MSTORED SASMSTORE=myMacDir;
```

To ***compile and store a macro*** into the SAS catalog called “sasmacr.sas7bcat” in a specified directory use code like this:

```
%MACRO myMacro / STORE SOURCE DES="My macro's description";
  my macro's code
%MEND myMacro;
```

The “SOURCE” and “DES” are optional, but strongly recommended. If you forget to include SOURCE and don’t store your source code somewhere else, there is no way to retrieve the source code in the future!

Note that any number of macros may be stored in a catalog, but there can only be one catalog per directory, and only one macro catalog may be in use at a time.

Example:

```
%MACRO addRatio(ds, num, den, ratio) / STORE
         SOURCE DES="Add ratio of num and den to ds";
  DATA &ds;
    SET &ds;
    &ratio = &num/&den;
  RUN;
%MEND addRatio;

/* Now you can quit SAS and come back */
LIBNAME macs "&wd/myMacros";
OPTIONS MSTORED SASMSTORE=macs;

DATA stuff;
  LENGTH ID $2.;
  INPUT id x y @@;
  DATALINES;
  AK 1 2  CM 3 4  MZ 10 20  BS 30 40
RUN;
%addRatio(stuff, x, y, rat)
```

iii. To ***run a stored macro***, just use it (assuming the above `OPTIONS` command has been run in this session).

iv. The stored macro system does the compilation step when the macro is stored, rather than once per session for the `AUTOCALL` method, which can be a major time saver, especially for large, complex macros.

v. To ***retrieve the source code of a macro*** (created with the `STORE` option) use code like this:
```
%COPY myMacro / SOURCE;  /* Code is written to the log */
```
or use code like this to put the code in a text file:
```
%COPY myMacro / SOURCE OUTFILE = "myMacroFile.sas";
```

vi. To ***see a listing of all of the macros in a stored macro catalog***, use code like this:
```
PROC CATALOG CATALOG=macs.SASMACR;
  CONTENTS;
RUN;
```
where the name “SASMACR” is constant (a keyword).
Or, in the Explorer window, navigate to the library you are using, and open “Sasmacr” by double clicking on it.

►At this point you should try storing macros across sessions as “Homework”, but there won’t be anything to turn in.

3) Connection between SQL and the macro system

a. Stats

```
PROC SQL NOPRINT;
  SELECT COUNT(x), MEAN(rat) INTO :nObs, :meanRat
  FROM stuff;
QUIT;

%PUT stuff has &Nobs rows and the mean of rat is &meanRat;
```

```
stuff has        4 rows and the mean of rat is    0.625
```

```
%PUT stuff has %SYSFUNC(TRIM(&Nobs)) rows and the mean of rat
 is %SYSFUNC(TRIM(&meanRat));
```

```
stuff has 4 rows and the mean of rat is 0.625
```

```
PROC SQL NOPRINT;
  SELECT COUNT(x) FORMAT 1., MEAN(rat) FORMAT 4.2
  INTO :nObs, :meanRat
  FROM stuff;
QUIT;

%PUT stuff has &Nobs rows and the mean of rat is &meanRat;
```

```
stuff has 4 rows and the mean of rat is 0.63
```

b. Create a macro variable from multiple observations in a table

```
DATA states;
  LENGTH id $2;
  input id x @@;
  DATALINES;
  PA 4.2  VA 5.9  NJ 3.1  AK 9.2  AZ 3.8
RUN;

PROC SQL NOPRINT;
  SELECT id INTO :idGt5 SEPARATED BY '", "'
  FROM states WHERE x>5;
QUIT;
%PUT _USER_;
… IDGT5  VA", "AK

TITLE "Mean ratio for x>5";
PROC MEANS DATA=states(WHERE=(id in ("&idGt5")));
  VAR x;
RUN;
```

```
N      Mean    Std Dev   Minimum   Maximum
2 7.5500000 2.3334524 5.9000000 9.2000000
```