Using Macros for Repetitive Tasks

(APMisc1 *)

Talk prepared by

Wendy Bergerud

February 2004

Research Branch, B.C. Ministry of Forests
P. O. Box 9519 Stn. Prov. Govt.
Victoria, B.C., V8W 9C2
PH: 250-387-5676 FAX: 250-387-0046

e-mail: Wendy.Bergerud@gems8.gov.bc.ca
http://www.for.gov.bc.ca/hre/topics/biomet.htm

* Annotated Programs - Miscellaneous topics - Document #1

Running similar SAS code many times

- Copy the code many times.
- Change a few variable and/or dataset names;

OR

- Set up a simple macro.
- Use macro variables to represent the changing variable and/or data set names.
- Run the macro many times.

Macro Definition

- **`%macro macroname;`**
  - starts the macro definition, giving the name for the macro.

  Sas code including macro variables.

- **`%mend;`** or **`%mend macroname;`**
  - ends the macro, the macroname is optional.

Macro Variables 1

- **`%let` *`macrovar = text values;`***
  - Defines the value to be used for the macro variable.
  - The values are treated as text. Working with numbers as numbers requires additional macro statements and functions.
  - Macro variables retain their value until assigned a new value.

Macro Variables 2

- **`%let macrovar = text values;`**
  - A null value will be assigned if there is no text after the equals sign.
  - Macro variables inside quotes must be inside double quotes to be resolved properly.
  - Macro variables will work outside of defined macros.

Calling the Macro

- ***%macroname;***
  - calls the macro and runs the code after resolving any macro variables within the macro.

- You may have noticed that macro statements can be identified by the fact that they start with a percent (%) sign.

Example 1: Producing printer plots for different pairs of variables

- Want several plots of various y-variables against different x-variables.
- The following program plots ***yvar*** versus ***xvar*** using the first digit of the treat values as the plotting symbol.
- The names of the variables are included in a title statement so that they will appear at the top of the graphical printout.

Example 1: Producing printer plots for different pairs of variables

```
/** Macro for plotting pairs of variables **/

%macro plots;    *<< Starting the macro defn;

title "For &yvar against &xvar";

proc plot;
  plot &yvar*&xvar = treat; run;

%mend;                              *<< Macro end;
```

Example 1: Producing printer plots for different pairs of variables

```
** Calling the macro three times;

*** plots y1 vs y2;
%let yvar = y1; %let xvar = y2;  %plots;

*** plots y1 vs x1;
                %let xvar = x1;  %plots;

*** plots y2 vs x1;
%let yvar = y2;                  %plots;
```

Example 2: Producing printer plots for different subsets of the data

- Suppose that we want to produce plots as in example 1, but we want to do this for subsets of the data.

- Use another variable to define the different subsets, e.g. zone.

Example 2: Producing printer plots for different subsets of the data

```
/** Macro for plotting pairs of variables **/

%macro plots;     *<< Starting the macro defn;

title  "For &yvar against &xvar";
title2 "For zones: &zonevals";
*** Must use double quotes in the titles!;

proc plot; where zone in(&zonevals);*by zone;
  plot &yvar*&xvar = treat; run;

%mend plots;                          *<< Macro end;
```

Example 2: Producing printer plots for different subsets of the data

```
** Calling the macro twice;

*** plots y1 vs y2 for zones 2 & 3;
%let yvar = y1; %let xvar = y2;
%let zonevals = 2,3;                    %plots;

*** plots y1 vs x1 for zones 3, 4 & 5;
                  %let xvar = x1;
%let zonevals = 3 4 5;                  %plots;
```

Example 5: Reading in many tables from an ACCESS database

- We want to convert data from a number of tables in an ACCESS database into permanent SAS datasets.
- First use the import wizard to create a simple program using PROC IMPORT.
- Build macro around that code.
- Program allows speedy recreation of SAS datasets as errors in original data require starting again.

Example 5: Reading in many tables from an ACCESS database

```
/** Importing individual tables
                          from Access Database **/
libname wildlife "H:\Projects\Data";

%macro import;

PROC IMPORT OUT= WILDLIFE.&name DBMS=ACCESS97
  REPLACE DATATABLE= "&name";
DATABASE="H:\Projects\Data\wtppdb1f_d5.mdb";
RUN;

%mend;
```

Example 5: Reading in many tables from an ACCESS database

```
** Calling the macro many times;

%let name = AppCompInfo;          %import;
%let name = block;                %import;
%let name = plot;                 %import;
%let name = reserve;              %import;
%let name = reserveDetail;        %import;
%let name = reserveSample;        %import;
%let name = reserveTree;          %import;
%let name = sample;               %import;
%let name = tree;                 %import;
```

Example 6: Reading and concatenating many data files into one SAS dataset

- Want to read in several data files with very similar layouts and merge into one large SAS data set.
- Need to be able to add or delete one variable name in the input for each data file depending upon data file.
- Create the final SAS data set with the first file read in, and then concatenate following data files as they are read in.

Example 6: Reading and concatenating many data files into one SAS dataset

```
%macro readin;
data &dsn;
  infile "&inf" dsd firstobs = 2 missover;
  input inst $ blk plt splt can &inpt lay $
    spp $ percov modht ;
run;

%if &flag = 1 %then %do; *<< First data set;
   data all; set &dsn;     run; %end;
%else %do;              *<< subsequent datasets;
   data all; set all &dsn; run; %end;
%mend;
```

Example 6: Reading and concatenating many data files into one SAS dataset

```
** Calling the macro many times;
%let flag = 1; *<< For first data set;
%let dsn = b1a; %let inf = aaa.csv;
  %let inpt = ;                          %readin;

%let flag = 2; *<< For subsequent datasets;
%let dsn = b1b; %let inf = bbb.csv;
  %let inpt = regen $;                   %readin;
%let dsn = b1c; %let inf = ccc.csv;
  %let inpt = ;                          %readin;
%let dsn = b1d; %let inf = ddd.csv;
  %let inpt = regen $ ;                  %readin;
```

Example 6a: Reading and concatenating many data files into one SAS dataset

```
** Calling the macro many times;
%let flag = 1; *<< For first data set;
%let dsn = b1a; %let inf = aaa.csv;
  %let inpt = ;                          %readin;

%let flag = 2; *<< For subsequent datasets;
                 %let inf = bbb.csv;
  %let inpt = regen $;                   %readin;
                 %let inf = ccc.csv;
  %let inpt = ;                          %readin;
                 %let inf = ddd.csv;
  %let inpt = regen $ ;                  %readin;
```

Dealing with Errors

- If possible, start with ordinary and working SAS code.
- Go slow - build macro in small pieces.
- Use a **`%put`** statement to print values of macro variables to the SAS Log.
  - These are also useful for monitoring the progress of a long SAS job.
- List all macro variables to SAS Log with **`%put _user_;`**

Use SAS debugging options

- SYMBOLGEN – the values of the macro variables are written to the SAS log as the program is run.
- MPRINT – as the SAS statements are generated by the macro, they are written to the SAS log.
  - To turn on: `options symbolgen mprint;`
  - To turn off: `options nosymbolgen nomprint;`
- Can greatly increase the size of the SAS log.

The End