CHAPTER 7. The Macro Language

a. Introduction
b. Macro variables
c. Macros
d. Conditional execution, macro functions, and other topics
e. DATA step interface: CALL SYMPUT

AN INTRODUCTION TO THE SAS MACRO LANGUAGE

What is the SAS macro language?

- A tool for extending and customizing the SAS System by enabling you to do the following:
  - Dynamically create SAS code
  - Reduce redundant code
  - Pass information between SAS steps
  - Conditionally execute SAS steps
  - Write special analysis routines to share with other people
- Thus, it can help you to produce better, more efficient, and more effective programs and applications.

The macro language is all about TEXT SUBSTITUTION.

The macro facility allows you to work with two new objects:

- Macro variables (identified by the ampersand &)
- Macros (identified by the percent sign %)

Before regular SAS compilation occurs, the SAS macro processor parses your submitted code and "resolves" any macro references – that is, it substitutes the appropriate text. Thus, macro coding in effect creates regular SAS code.

Macro Variables

- A macro variable is a string of text identified by a name.

- You use a macro variable by putting its name in your code wherever you want the string of text to be substituted.

- Some macro variables are built into SAS – these are called system-defined macro variables.

- For example, &sysdate, used in the standard BIOS511 footnote, is a system-defined macro variable that makes the date of the current SAS session available to your program.

- Other system-defined macro variables include

  - SYSTIME time of SAS invocation
  - SYSSCP operating system being run
  - SYSVER version of SAS being run
  - SYSERR return code set by last DATA or PROC step
  - SYSLAST name of the most recently created SAS data set in the form of libref.name. If no data set was created, the value is _NULL_ .

- Note that the values of &SYSDATE and &SYSTIME are not SAS date or time values, they are simply character strings.

- You can also create user-defined macro variables in several ways: by using the %LET statement, by using the CALL SYMPUT statement, or by defining a macro with parameters.

- Macro variables have what is called "scope". System-defined macro variables and macro variables created with %LET or CALL SYMPUT have "global" scope, which means they are available anywhere in your program. Macro variables that are parameters to a macro or are created within a macro have "local" scope, which means they are available only within that macro.

User-Defined Macro Variables and %LET

Anywhere in a SAS program, you can use the %LET statement to create a macro variable and assign it a value. Typically, this is done outside of a PROC step or a DATA step.

The general form of the %LET statement is

```
%LET variable = value;
```

Often, %LET is used at the beginning of a program to create a macro variable for storing a value that is used several times in the program or that is frequently changed when the program is rerun.

For example, see this ODS-related example that references the same folder repeatedly:

```
ODS RTF FILE="c:\bios511\odsoutput\list.rtf";
PROC PRINT DATA=bios511.class;
RUN;
ODS RTF CLOSE;

ODS PDF FILE="c:\bios511\odsoutput\stats.pdf";
PROC MEANS DATA=bios511.class;
     VAR ht wt;
RUN;
ODS PDF CLOSE;
```

If you use %LET to store the folder location in a macro variable at the beginning of the program, then you don't have to keep retyping that complicated path. Also, if you decide to store output somewhere else, you only have to change the information in one place in your code. Imagine the maintenance advantage if this program wrote out 10 or 20 ODS files!

```
%LET odsdir=c:\bios511\odsoutput;

ODS RTF FILE="&odsdir\list.rtf";
PROC PRINT DATA=bios511.class;
RUN;
ODS RTF CLOSE;

ODS PDF FILE="&odsdir\stats.pdf";
PROC MEANS DATA=bios511.class;
     VAR ht wt;
RUN:
ODS PDF CLOSE;
```

Macro variable names follow SAS naming conventions and can be up to 32 characters long.

A macro variable value can be any string, and these values have the following properties:

- The case of value is preserved.
- The maximum length of a value is 32K characters (really long!).
- The minimum length of a value is 0 characters (null value).
- All values, including what appear to be numbers, are stored as character strings.
- Mathematical expressions are not evaluated (unless you apply the macro function %EVAL).
- If you specify a value that includes quotes, the quotes are stored as part of the value.
- Leading and trailing blanks are removed as values are stored.
- If the macro variable already exists, you can assign it a new value and the new value replaces the current value.

Examples:

```
%let name = Joe Smith;

%let name = 'Joe Smith';

%let start = ;

%let total = 0;

%let total = 3+4;

%let total = %eval(3+4);
```

Referencing Macro Variables

- To substitute the value of a macro variable in your program, you must reference it.

- A macro variable reference

  - is made by preceding the macro variable name with an ampersand (&).

  - causes the macro processor to search for the named variable and return the value if the variable exists.

- For example, if the macro variable AMOUNT has a value of 800 :

```
 where cost > &amount ;
```

generates

```
where cost > 800 ;
```

- If you need to reference a macro variable within a literal, enclose the literal in double quotes.

- For example, if the macro variable CITY has a value of Dallas :

```
where c = "&city" ;
```

generates

```
where c = "Dallas" ;
```

```
where c = '&city' ;
```

generates

```
where c = '&city' ;
```

- To monitor the values that are substituted when macro variables are referenced, you can specify the **SYMBOLGEN** system option.

Macro Referencing Examples

Say that you commonly need to generate random numbers between 1 and 100. Sometimes you need 10 random numbers and sometimes you need 45. You could use this program to generate any number of random numbers in that range.

```
%LET n=10;

DATA randoms;
      DO num=1 TO &n;
            x = INT(100*RANUNI(0) + 1);
            OUTPUT;
      END;
RUN;

TITLE "Data set with &n random numbers";
PROC PRINT DATA=randoms NOOBS;
RUN;
```

Data set with 10 random numbers

num	x
1	60
2	15
3	91
4	62
5	99
6	62
7	31
8	7
9	65
10	65

Or say that you wanted a program that could be used to print a list of any type of car (sedan, SUV, hatchback, etc.) available in the United States in 2011.

```
%LET which=Minivan;

PROC PRINT DATA=bios511.cars2011;
     WHERE Type="&which";
     VAR Type Make Model BaseMSRP CityMPG;
     TITLE "&which cars available in the US in 2011";
RUN;
```

```
103  %LET which=Minivan;
104
105  PROC PRINT DATA=bios511.cars2011;
106      WHERE Type="&which";
107      VAR Type Make Model BaseMSRP CityMPG;
108      TITLE "&which cars available in the US in 2011";
109  RUN;

NOTE: There were 7 observations read from the data set BIOS511.CARS2011.
      WHERE Type='Minivan';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Minivan cars available in the US in 2011

Obs	Type	Make	Model	Base MSRP	City MPG
54	Minivan	Chrysler	Town and Country	29995	17
59	Minivan	Dodge	Grand Caravan	22995	17
84	Minivan	Honda	Odyssey	28075	18
113	Minivan	Kia	Sedona	24595	18
179	Minivan	Nissan	Quest	27750	19
219	Minivan	Toyota	Sienna	25060	19
230	Minivan	Volkswagen	Routan	26930	17

From these examples, you can see that some advantages of macro variables result when:

- You can use the same macro variable in several places.
- You can change your code in many places at once by changing the value of a macro variable in one place.
- Use of a macro variable makes the code more flexible and more easily re-usable.

Macros

- Macros are more complicated objects than macro variables.

- Like a macro variable, the end result of running a macro is text that is included as part of your regular SAS program.

- But unlike a macro variable, macros can include executable statements that manipulate information and perform logical control.

- Macros and executable statements that are a part of the macro language are all identified by the percent character % .

- The %LET statement mentioned above is actually an executable macro statement that can appear any place in a SAS program, not just in a macro.

Defining and Calling Macros

- You define a macro by coding %MACRO as the first statement and %MEND as the last statement.

- You give the macro a name on the first statement and call it later by using that name.

For example, you could define a macro called %runstats that produced statistics on some variables in a data set.

```
%MACRO RUNSTATS ;

   OPTIONS NOLABEL;
   PROC MEANS DATA=bios511.cars2011;
        VAR CityMPG HwyMPG CurbWeight;
    RUN;

%MEND ;
```

Running this code would not actually produce the statistics – it would simply define the macro. The code would not actually run, and therefore the statistics would not be produced, until we also submitted this code:

```
%runstats
```

At that point we would see this in the SAS log:

```
67   %MACRO RUNSTATS ;
68
69      OPTIONS NOLABEL;
70      PROC MEANS DATA=bios511.cars2011;
71           VAR CityMPG HwyMPG CurbWeight;
72        RUN;
73
74   %MEND ;
75
76   %runstats

NOTE: There were 244 observations read from the data set
      BIOS511.CARS2011.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.40 seconds
      cpu time            0.04 seconds
```

And this in the SAS output window:

```
                          The SAS System

                        The MEANS Procedure

 Variable       N           Mean        Std Dev        Minimum        Maximum
 ------------------------------------------------------------------------------
 CityMPG      243     21.6995885      9.2121618     12.0000000    106.0000000
 HwyMPG       243     28.6625514      7.8441243     17.0000000     92.0000000
 CurbWeight   243        3864.30    908.9176077        1805.00        6325.00
 ------------------------------------------------------------------------------
```

Notice that by default, running a macro does not show us the actual code produced by the macro. To see that, we need to turn on the SAS option MPRINT.

Our SAS log after submitting

```
OPTIONS MPRINT;
%runstats
```

would be

```
82   OPTIONS MPRINT;
83   %runstats
MPRINT(RUNSTATS):   OPTIONS NOLABEL;
MPRINT(RUNSTATS):   PROC MEANS DATA=bios511.cars2011;
MPRINT(RUNSTATS):   VAR CityMPG HwyMPG CurbWeight;
MPRINT(RUNSTATS):   RUN;

NOTE: There were 244 observations read from the data set BIOS511.CARS2011.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

The MPRINT option can be very helpful for debugging when you are having trouble getting a macro to work correctly.

But the macro above isn't very flexible. We can make it more flexible by adding parameters. For example, we might use macro parameters to control which data set and variables to analyze and even which statistics should be produced.

Macro Parameters

- Macro parameters are simply macro variables that are available within the macro and nowhere else.

- Macro parameters provide a way to pass values to macro variables used within a macro.

For example, suppose we wanted to modify the runstats macro to analyze variables in any data set.

```
%MACRO RUNSTATS(ds=,vars=);

   OPTIONS NOLABEL;
   PROC MEANS DATA=&ds;
        VAR &vars;
    RUN;

%MEND ;
```

After submitting this code to define the macro, we could call it like this:

```
%runstats(ds=bios511.class, vars=ht wt)

%runstats(ds=bios511.cars2011, vars=seating hwyMPG reliability)
```

The first call would generate this code:

```
OPTIONS NOLABEL;
PROC MEANS DATA=bios511.class;
     VARS ht wt;
RUN;
```

The order of parameter specification does not matter.

```
%runstats(vars=seating hwyMPG reliability, ds=bios511.cars2011)
```

would be equivalent to the second call above.

We can also give macro parameters default values so that they don't always need to be specified. For example, we could modify the runstats macro to accept a list of the statistics to be produced, but we could set up a default list in case the user didn't pass in any statistic names.

```
%MACRO RUNSTATS(ds=,vars=,stats=N NMISS MEAN MIN MAX);

   OPTIONS NOLABEL;
   PROC MEANS DATA=&ds &stats;
        VAR &vars;
    RUN;

%MEND ;
```

```
%runstats(ds=bios511.cars2011, vars=seating hwyMPG reliability)
```

would produce

```
OPTIONS NOLABEL;
PROC MEANS DATA=bios511.cars2011 N NMISS MEAN MIN MAX;
     VARS seating hwyMPG reliability;
RUN;
```

```
%runstats(ds=bios511.cars2011, vars=seating hwyMPG reliability,
          stats=N MEAN STD)
```

would produce

```
OPTIONS NOLABEL;
PROC MEANS DATA=bios511.cars2011 N MEAN STD;
     VARS seating hwyMPG reliability;
RUN;
```

Macro Processing

- Macro processing happens before the SAS compile phase.

- Macro processing is triggered by the characters & and % .

- When a & is found, the macro processor will search the macro variable table for the value of the variable in the string immediately following the & . This value is then substituted for the reference to the macro variable.

- When a % is encountered, the macro processor will interpret the following characters as a macro that you defined or as a macro function or keyword.

- So don't name your macros with the same name as a macro function or keyword!

- When a reference to a macro is found, the macro will be run, and the result will be inserted in place of the reference.

- After the macro processor has interpreted the value of a macro variable, a macro, or a macro function, the result will be scanned again until no more references with & or % exist.

- The final result is treated as if it were a part of your original SAS program.

Dots (.) and Double Dots (..)

Sometimes you will need to insert a . or even .. after a macro variable name to help SAS parse your code correctly.

```
%LET form=DEM;
PROC PRINT DATA=&form.A(OBS=10);
RUN;
```

The . here tells SAS to look for the value of macro variable form rather than looking for macro variable forma.

```
%LET which=cusco;
ODS PDF FILE="c:\projects\peru\&which..pdf";
```

The first . here tells SAS where the macro variable name ends, and the second . becomes the separator between the file name cusco and the file extension pdf.

Conditional Execution

- You can perform conditional execution at the macro level with

  %IF-%THEN and %ELSE statements

- The general form of %IF-%THEN and %ELSE statements is:

```
%IF expression %THEN text ;

%ELSE text ;
```

- *expression* can be any valid macro expression.

- The %ELSE statement is optional.

- Text following the %THEN and %ELSE keywords can be:
  - a macro programming statement,
  - constant text,
  - an expression,
  - a macro variable reference,
  - a macro call.

❖ Use %DO and %END statements following %THEN or %ELSE to submit text that contains semicolons.

```
%IF expression %THEN %DO ;

       statement ;

       statement ; ....

%END ;

%ELSE %DO ;

       statement ;

       statement ; ....

%END ;
```

NOTE: The macro statements above can be used only inside a macro definition.

Conditional processing can make our runstats example much more interesting. For example, what if we wanted to run PROC MEANS for continuous variables and PROC FREQ for categorical variables? Also, you can set up the macro to use your preferred settings automatically.

```
%MACRO RUNSTATS2(ds=,cont=,cat=);

   %IF &cont ne %THEN %DO;
         OPTIONS NOLABEL;
         PROC MEANS DATA=&ds N NMISS MEAN STD MIN MAX MAXDEC=2;
               VAR &cont;
         RUN;
   %END;

   %IF &cat ne %THEN %DO;
         PROC FREQ DATA=&ds;
               TABLES &cat / MISSING;
         RUN;
   %END;

%MEND RUNSTATS2;
```

```
%runstats2(ds=bios511.class, cont=ht wt, cat=sex)
```

```
MPRINT(RUNSTATS2):   OPTIONS NOLABEL;
MPRINT(RUNSTATS2):   PROC MEANS DATA=bios511.class N NMISS MEAN STD MIN MAX MAXDEC=2;
MPRINT(RUNSTATS2):   VAR ht wt;
MPRINT(RUNSTATS2):   RUN;

NOTE: There were 6 observations read from the data set BIOS511.CLASS.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds


MPRINT(RUNSTATS2):   PROC FREQ DATA=bios511.class;
MPRINT(RUNSTATS2):   TABLES sex / MISSING;
MPRINT(RUNSTATS2):   RUN;

NOTE: There were 6 observations read from the data set BIOS511.CLASS.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The MEANS Procedure

Variable	N	N Miss	Mean	Std Dev	Minimum	Maximum
HT	6	0	50.00	26.42	12.00	74.00
WT	5	1	119.20	90.32	1.00	195.00

The FREQ Procedure

SEX	Frequency	Percent	Cumulative Frequency	Cumulative Percent
	1	16.67	1	16.67
F	1	16.67	2	33.33
M	4	66.67	6	100.00

Let's see how this macro works if we only have one type of variable to examine.

```
%runstats2(ds=bios511.cars2011, cat=country satisfaction)
```

```
MPRINT(RUNSTATS2):   PROC FREQ DATA=bios511.cars2011;
MPRINT(RUNSTATS2):   TABLES country satisfaction / MISSING;
MPRINT(RUNSTATS2):   RUN;
```

The FREQ Procedure

Country	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Germany	49	20.08	49	20.08
Great Britain	7	2.87	56	22.95
Italy	2	0.82	58	23.77
Japan	97	39.75	155	63.52
Korea	20	8.20	175	71.72
Sweden	12	4.92	187	76.64
USA	57	23.36	244	100.00

Satisfaction	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	77	31.56	77	31.56
1	21	8.61	98	40.16
2	77	31.56	175	71.72
3	51	20.90	226	92.62
4	14	5.74	240	98.36
5	4	1.64	244	100.00

The MLOGIC SAS system option can be useful for debugging macros that use conditional logic. If we turn on the MLOGIC option and rerun the macro call above, this SAS log results:

```
MLOGIC(RUNSTATS2):  Beginning execution.
MLOGIC(RUNSTATS2):  Parameter DS has value bios511.cars2011
MLOGIC(RUNSTATS2):  Parameter CAT has value country satisfaction
MLOGIC(RUNSTATS2):  Parameter CONT has value
MLOGIC(RUNSTATS2):  %IF condition &cont ne is FALSE
MLOGIC(RUNSTATS2):  %IF condition &cat ne is TRUE
MPRINT(RUNSTATS2):   PROC FREQ DATA=bios511.cars2011;
MPRINT(RUNSTATS2):   TABLES country satisfaction / MISSING;
MPRINT(RUNSTATS2):   RUN;

NOTE: There were 244 observations read from the data set BIOS511.CARS2011.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


MLOGIC(RUNSTATS2):  Ending execution.
```

You can turn off any of the macro-related SAS system options by submitting

```
OPTIONS NOSYMBOLGEN NOMPRINT NOMLOGIC;
```

Macro Functions

- Many macro applications require character manipulation.

- Below is a list of selected macro character functions:

FUNCTION NAME	DESCRIPTION
%UPCASE	translates letters from lower case to upper case
%LENGTH	determines the length of a character string
%SUBSTR	produces a substring of a character string
%SCAN	extracts a word from a character string
%INDEX	finds the position of the first occurrence of a character string within another string

%UPCASE

- Most macro comparisons are case sensitive.

- The %UPCASE function translates all letters in an argument to uppercase.

We could use the %UPCASE function to make one of our earlier examples more robust.

```
%LET which=minivan;

PROC PRINT DATA=bios511.cars2011;
     WHERE UPCASE(Type)= "%UPCASE(&which)";
     VAR Type Make Model BaseMSRP CityMPG;
     TITLE "&which cars available in the US in 2011";
RUN;
```

Here, we use the regular UPCASE function to uppercase the value of the data set variable before comparison, and we use the macro %UPCASE function to uppercase the value of whatever the macro user passes in as the value of macro parameter &which. Thus, the two values will be comparable, allowing the user to pass in minivan or MINIVAN or Minivan or MiniVan or whatever, and get the desired results.

Notice that here we are using the %UPCASE function in the middle of regular SAS code, not within a macro. It would work within a macro definition as well.

%SUBSTR

- The general form of the %SUBSTR function is:

  %SUBSTR(argument,position,n) ;

- The first two arguments are required.
- The %SUBSTR function returns the portion of *argument* beginning at *position* for a length of *n*. If *n* is omitted, the function returns the string from *position* to the end of *argument*.
- *argument*, *position*, and *n* values can contain:
  - constant text,
  - macro variable references,
  - macro functions,
  - macro calls.

%SCAN

- The general form of the %SCAN function is:

  %SCAN(argument,n,delimiter) ;

- The first two arguments are required.
- The %SCAN function returns the *n*th word of *argument*, where words are strings of characters separated by one or more delimiters. The recognized delimiters include

  blank . < (+ | & ! $ *) ; ^ - / , % > \

- The function returns a null string if there are fewer than *n* words in *argument*.
- *argument* and *n* values can contain:
  - constant text,
  - macro variable references,
  - macro functions,
  - macro calls.

Example:

```
%MACRO freqout(ds=,vars=) ;

      %LET var1 = %SCAN(&vars,1) ;

      %LET var2 = %SCAN(&vars,2) ;

      PROC FREQ DATA=&ds;

         TABLE &var1 * &var2 / OUT=countsboth ;
         TABLE &var1 / out=counts1 ;
         TABLE &var2 / out=counts2 ;

      RUN ;

%MEND ;
```

`%freqout(ds=bios511.sales, vars=dept clerk)` would resolve to:

```
PROC FREQ DATA=bios511.sales;

TABLE dept * clerk / OUT=countsboth;
TABLE dept / OUT=counts1;
TABLE clerk / OUT=counts2;

RUN;
```

Writing Text to the SAS Log

❖ When errors or problems occur in macro processing, you may want to write your own messages to the SAS log.

❖ The %PUT statement writes messages to the SAS log.

❖ The general form of the %PUT statement is:

```
%PUT text ;
```

The %PUT statement:

- can be used inside or outside a macro definition
- writes to the SAS log only
- always writes to a new log line starting in column one
- resolves macro triggers in text before text is written
- does not require quotes around text.

❖ %PUT is often used to discover the value of a macro variable. Perhaps early on in a SAS session you used %LET to assign a value to the macro variable mymacrovar, but later on you can't remember what you set it to. You could find out the value with

```
%PUT &mymacrovar ;
```

You can also include plain text in what you write with %PUT.

```
140  %LET who=Jackie;
141  %PUT Hello &who;
Hello Jackie
```

Iterative Processing

- Many macro applications require iterative processing.
- With the iterative %DO statement (available only within a macro), you can
  - execute macro programming code
  - generate SAS code.

Example 1:

```
%MACRO doloop ;

     %DO i = 1 %TO 17 ;

          PROC FREQ DATA=medications ;
               WHERE '370000' <= DrugCode&i <= '380000' ;
               TABLES DrugCode&i ;
               TITLE "DrugCode&i";
          RUN;

     %END ;


%MEND ;
```

Submitting %DOLOOP generates:

```
PROC FREQ DATA=medications ;
       WHERE '370000' <= DrugCode1 <= '380000' ;
       TABLES DrugCode1 ;
       TITLE "DrugCode1" ;
RUN;
PROC FREQ DATA=medications ;
       WHERE '370000' <= DrugCode2 <= '380000' ;
       TABLES DrugCode2 ;
       TITLE "DrugCode2" ;
RUN;
```

and so forth all the way to:

```
PROC FREQ DATA=medications ;
       WHERE '370000' <= DrugCode17 <= '380000' ;
       TABLES DrugCode17 ;
       TITLE "DrugCode17" ;
RUN;
```

- The general form of the iterative %DO statement is:

```
%DO index-variable = start %TO stop %BY increment ;

            text ;

%END ;
```

- %DO and %END are valid only inside a macro definition.

- *index-variable* is a macro variable. The index-variable is created if it doesn't already exist.

- *start*, *stop*, and *increment* can be any valid expressions that resolve to integers.

Notice that this is a case where our macro didn't have any parameters, but the macro was useful anyway because we could use macro logic to do complex processing (iterative or based on conditional logic).

Example 2:

```
DATA old;
     x1=5; x2=27; x3=6; x4=17; x5=0;
RUN;

%MACRO renamevars;
     %DO i=1 %TO 5;
          x&i=y&i
     %END;
%MEND renamevars;

DATA new;
     SET old;
     RENAME
          %renamevars
     ;
RUN;
```

The crucial part of the generated SAS log is

```
291  DATA new;
292      SET old;
293      RENAME
294          %renamevars
MPRINT(RENAMEVARS):   x1=y1 x2=y2 x3=y3 x4=y4 x5=y5
295      ;
296  RUN;
```

data with old names

Obs	x1	x2	x3	x4	x5
1	5	27	6	17	0

data with new names

Obs	y1	y2	y3	y4	y5
1	5	27	6	17	0

Example 3:

```
%LET responses = CHD CVD CANCER ;

%MACRO rate ;

     %DO i=1 %TO 3;

                %LET response = %SCAN(&responses,&i) ;

                PROC LOGISTIC ;

                MODEL &response = age bmi sbp ;

                RUN;

     %END;

%MEND ;
```

The macro call %rate resolves to:

```
       PROC LOGISTIC ;

           MODEL CHD = age bmi sbp ;

       RUN;

       PROC LOGISTIC ;

           MODEL CVD = age bmi sbp ;

       RUN;

       PROC LOGISTIC ;

           MODEL CANCER = age bmi sbp ;

       RUN;
```

Example 4: Run a report for a specified range of years.

```
%MACRO report(start=,end=) ;

   %DO i=&start %TO &end %BY 1 ;

       PROC FREQ DATA=sashelp.prdsal2;

          TABLES prodtype product ;

          WHERE year = &i ;

          TITLE "Report for Year &i" ;

       RUN;

   %END ;

%MEND ;
```

The macro call `%report(start=1995,end=1997)` would run the report for years 1995, 1996, and 1997.

Conditional Loops

- You can perform conditional iteration in macros with %DO %WHILE and %DO %UNTIL statements.

- The general forms of these statements are:

```
%DO %WHILE(expression) ;
        text ;
 %END ;


 %DO %UNTIL(expression) ;
        text ;
 %END ;
```

- *expression* can be any valid macro expression.

%DO %WHILE loops

- Evaluate *expression* at the top of the loop
- Execute repetitively while *expression* is true

%DO %UNTIL loops

- Evaluate *expression* at the bottom of the loop
- Execute repetitively until *expression* is true
- Execute at least once

Example:

```
%LET responses = CHD CVD CANCER ;

%MACRO rate ;

     %LET i = 1;

     %DO %UNTIL (%SCAN(&responses,&i)=);

                %LET response = %SCAN(&responses,&i) ;

                PROC LOGISTIC ;

                MODEL &response = age bmi sbp ;

                RUN;

                %LET i=%EVAL(&i + 1);

     %END;

%MEND ;
```

Just as before, the macro call %rate resolves to:

```
PROC LOGISTIC ;

    MODEL CHD = age bmi sbp ;

RUN;

PROC LOGISTIC ;

    MODEL CVD = age bmi sbp ;

RUN;

PROC LOGISTIC ;

    MODEL CANCER = age bmi sbp ;

RUN;
```

Note: The %EVAL function enables the macro language to do arithmetic.

DATA Step Interface

- In many applications you need to create macro variables that are assigned values based on data values (values of regular SAS variables).

- You can use the **SYMPUT** routine to assign a value available to the DATA step to a macro variable.

- The general form of the SYMPUT routine is:

```
CALL SYMPUT('macro-variable', text) ;
```

where *text* is almost always the name of a regular (data set) variable

- *macro-variable* is assigned the character value of *text*.

- A macro variable created with CALL SYMPUT is not available in the same DATA step in which it is created – only after that DATA step.

CALL SYMPUT Example 1

Sometimes you would like to include information from a DATA step variable in a title. For example, you might want a title to include count information.

This example lists the sales of the clerk named Burley, and a count of his sales is included in the listing title. CALL SYMPUT is used to transfer the count information from a DATA step variable to a macro variable, and then the macro variable is used in the title.

```
DATA count;
      SET bios511.sales END=lastrec;
      RETAIN nburley 0;
      IF UPCASE(clerk)='BURLEY' THEN nburley = nburley + 1;
      IF lastrec THEN
            CALL SYMPUT('burleycount',nburley);
RUN;
PROC PRINT DATA=bios511.sales NOOBS;
      WHERE UPCASE(clerk)='BURLEY';
      TITLE "The &burleycount Sales by Mr. Burley";
RUN;
```

```
                The            9 Sales by Mr. Burley

          DEPT     CLERK      PRICE     COST     WEEKDAY    DAY

          FURS     BURLEY    599.95    180.01      THR        5
          SHOES    BURLEY     69.95     34.93      THR        5
          FURS     BURLEY    800.00    240.00      MON        9
          SHOES    BURLEY     75.00     36.31      WED       11
          FURS     BURLEY    499.95    200.01      SAT       14
          FURS     BURLEY    700.00    210.00      THR       19
          SHOES    BURLEY     59.95     31.78      WED       25
          SHOES    BURLEY     54.95     30.00      FRI       27
          SHOES    BURLEY     55.00     30.02      FRI       27
```

We can use the macro function %LEFT to left justify the macro variable value, thus getting rid of the extra blanks before the count. Also, note that it is important to use double quotes around the title so that the macro variable is resolved.

```
PROC PRINT DATA=bios511.sales NOOBS;
      WHERE UPCASE(clerk)='BURLEY';
      TITLE "The %left(&burleycount) Sales by Mr. Burley";
RUN;
```

```
                     The 9 Sales by Mr. Burley

          DEPT     CLERK      PRICE     COST     WEEKDAY    DAY

          FURS     BURLEY    599.95    180.01      THR        5
          SHOES    BURLEY     69.95     34.93      THR        5
          FURS     BURLEY    800.00    240.00      MON        9
          SHOES    BURLEY     75.00     36.31      WED       11
          FURS     BURLEY    499.95    200.01      SAT       14
          FURS     BURLEY    700.00    210.00      THR       19
          SHOES    BURLEY     59.95     31.78      WED       25
          SHOES    BURLEY     54.95     30.00      FRI       27
          SHOES    BURLEY     55.00     30.02      FRI       27
```

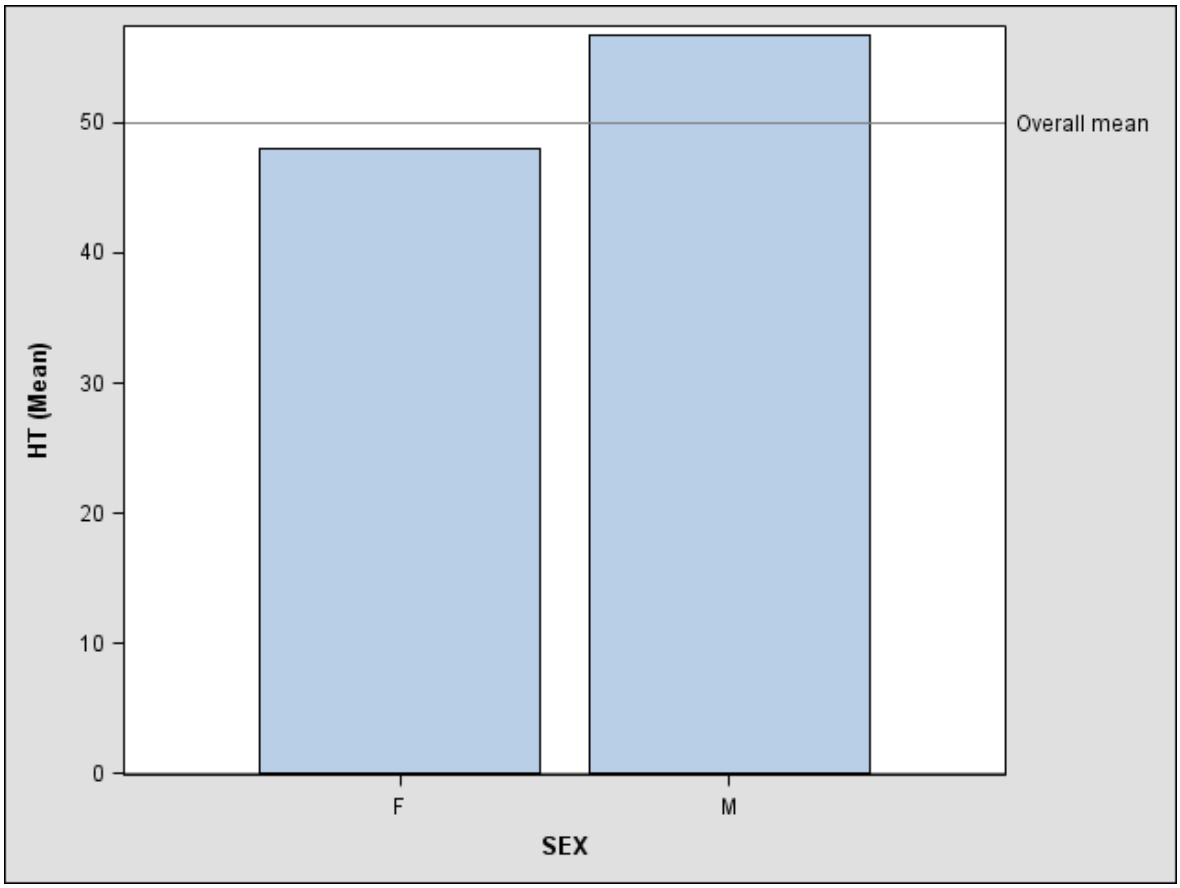
CALL SYMPUT Example 2

In the next example, we use CALL SYMPUT to transmit a value to use as a reference line location on a graph.

```
PROC MEANS DATA=bios511.class NOPRINT;
    VAR ht;
    OUTPUT OUT=means MEAN=meanht;  /* the mean here is exactly 50, by the way */
RUN;

DATA _NULL_;
    SET means;
    CALL SYMPUT('macmeanht',put(meanht,best.));
RUN;

PROC SGPLOT DATA=bios511.class;
    VBAR sex / RESPONSE=ht STAT=MEAN;
    REFLINE &macmeanht / AXIS=Y LABEL='Overall mean';
RUN;
```



CALL SYMPUT Example 3

Here, we use CALL SYMPUT to provide a better denominator for percentages in our counting hobbies example from Chapter 6. The preferred denominator for these percentages is the number of people surveyed, not the total number of hobbies.

```
DATA hobbies;
    SET bios511.hobbies END=eof;

    KEEP name hobby;

    i=1;
    DO UNTIL (SCAN(hobbies,i)=' ');
        Hobby=SCAN(hobbies,i);
        OUTPUT;
        i=i+1;
    END;

    RETAIN n 0;
    n=n+1;
    IF eof THEN CALL SYMPUT('nobs',PUT(n,BEST.));
RUN;

%PUT &nobs;       /* not needed, but let's make sure nobs is what we expect */

PROC FREQ DATA=hobbies NOPRINT;
    TABLES hobby / OUT=freqout;
RUN;

DATA add_denominator;
    SET freqout(DROP=percent);
    Percent=100*(count/&nobs);
    LABEL Percent='Percent of class members having this hobby'
          Count  ='Number of class members having this hobby';
RUN;

PROC PRINT LABEL DATA=add_denominator NOOBS;
    VAR Hobby Count Percent;
    TITLE 'Report on hobbies of class members';
    FORMAT Percent 8.2;
RUN;
```

Hobby	Number of class members having this hobby	Percent of class members having this hobby
baseball	2	33.33
eat	3	50.00
primp	1	16.67
read	2	33.33
swim	2	33.33
travel	2	33.33

Compare these percentages with the ones on page 47 of Chapter 6.

Multistep Macro Example

```
/* set up conditions */
LIBNAME perm "J:\ENRICHD\SC\SASDATA";
%LET retrieval_date=0911;


/* define the macro */
%MACRO process_form(form= );
PROC IMPORT DATAFILE="J:\ENRICHD\DM\&form..xls" OUT=temp DBMS=EXCEL REPLACE;
      RANGE="Sheet1$";
      GETNAMES=YES;
      MIXED=YES;
RUN;

DATA perm.&form&retrieval_date;
      SET temp;
      LENGTH Form $3;
      Form="&form";
      DROP x: ;
RUN;

TITLE "Preliminary check of perm.&form&retrieval_date";

PROC CONTENTS DATA=perm.&form&retrieval_date;
RUN;

PROC PRINT DATA=perm.&form&retrieval_date(OBS=10);
RUN;
%MEND process_form;


/* call the macro for each form */
%process_form(form=DEM)
%process_form(form=CFD)
%process_form(form=MBR)
```