

# AN INTRODUCTION TO MACRO VARIABLES AND MACRO PROGRAMS

## Mike Zdeb, University@Albany School of Public Health

### INTRODUCTION

There are a number of SAS® tools that you may never have to use. Why? The main reason is that you can usually reach the same result without them. Examples of such tools are arrays, macros, and quite possibly PROC SQL. However, though you may be able to reach the same result without resorting to these tools, the amount of SAS code needed to reach that result can often be seriously lessened by using the SAS tools you may have avoided. In some situations, you might even come to the conclusion that you just cannot achieve a desired goal with the collection of SAS tools you commonly use. Hopefully this paper will show you that a little investment of time and effort in learning about macros will allow you to attain previously unreachable goals with SAS and also lessen the amount of SAS code you currently use.

If you have resorted to trying to learn about macros by reading an introductory paper, you may have already encountered one of the factors that makes the topic of macros appear more difficult than other areas of SAS, i.e. there is a separate manual devoted to just the topic of macros (other topics sharing this distinction are PROC SQL and PROC TABULATE). A while ago, I adopted a very simple criterion for categorizing a SAS feature as being more complicated than others. If the topic deserves its own manual, it is, by my definition, more complicated. The topic may not necessarily be more complicated. It might just be that the number of features associated with the topic just make it seem more complicated. You can do a lot with macros and many of the things you can do are quite straightforward and no more complicated than SAS features you already use. So, rather than immediately introduce you to 'macro referencing environments' or 'double and triple ampersands' (&& and &&&), this paper will start slowly and give you enough examples that show the worth of knowing about macros, enough to make you want to know more.

### WHAT CAN YOU DO WITH MACROS

Though this section is titled ...what can you do with macros..., it could also have a subtitle of ...that you cannot already do without macros... Though there could be a longer list, for now it should be enough of an incentive to learn about macros if you know that they allow you to...

- avoid repetitious SAS code
- pass information from one part of a SAS job to another
- conditionally execute data steps and PROCs
- dynamically create code at execution time

Before illustrating how you can use macros to accomplish the tasks in the above lists, it is time to learn a little terminology. Exactly what is meant by ... a macro? The answer is ... it depends. At the low end of the macro ladder is the *macro variable*. A macro variable is a short hand way of referring to a string of stored text. You already know about *FILENAME* and *LIBNAME* statements. With either of them, you define a shortcut (a *fileref* or *libref*) that refer to a data file or data library respectively. In either case, you can use the short cut in lieu of the full file or data set location. At the high of the macro ladder are complete programs written as *macros*.

### MACRO VARIABLES

What if you had a number of different data sets and wanted to list the contents of each using PROC CONTENTS, then print the first five observations using PROC PRINT. If the first data set is named LABS, your SAS might look as follows:

```
proc contents data=labs;  
run;  
  
title 'DATA SET LABS';  
  
proc print data=labs (obs=5);  
run;
```

If the next data set is HOSPITAL, you would replace LAB in three lines of SAS code. The repetitious text is tedious, but necessary. The repetition of the task, changing three lines of code, is not necessary if you use a macro variable as a shortcut to represent the data set name...

#### Example 1...simple macro variable

```
%let dsn=LABS; ❶  
  
proc contents data=&dsn; ❷  
run;  
  
title "DATA SET &dsn"; ❸
```

```
proc print data=&dsn(obs=10);
run;
```

The statement starting with %LET ❶ assigns the text on the right (LAB) to the macro variable on the left (DSN). You should notice several features about the above SAS code...

- %LET is a GLOBAL statement. It can occur anywhere within a SAS job, i.e. it is not restricted to data steps (or PROCs). After executing the %LET statement, you can use the macro variable &DSN ❷ as a substitute for LAB.
- Once the macro variable is defined, you must precede the name of the macro variable with an ampersand (&) each time it is used. A long time ago, you learned that the names of SAS variables and data sets must start with a letter or underscore. A new item to remember is that, once a macro variable is defined, it must start with an & each time it is used.
- The title statement ❸ now uses double quotes instead of single quotes. Normally, either type of quote can be used in a title statement. If a macro variable appears within a quoted string (e.g title, footnote, note), double quotes must be used to ensure that the macro variable 'resolves' to its assigned value. Resolves is the SAS jargon used in place of "...is converted to...". Without the double quotes, the title in this example would be DATA SET &DSN, not DATA SET LAB.

If you want to use the same PROCs with data set HOSPITAL, you only have to change one line of your SAS job...

```
%let dsn=HOSPITAL;
```

If you want your title to be all uppercase, make sure that the macro variable &DSN contains uppercase text, i.e. LAB or HOSPITAL (though this is not absolutely necessary as is shown later in the paper in the discussion of macro functions using the %upcase macro function). Given that there is a data set named HOSPITALS, the LOG file that results from running the SAS job in example 1 is...

```
21  options symbolgen; ❶
22  %let dsn=HOSPITAL;
SYMBOLGEN: Macro variable DSN resolves to HOSPITAL ❷
23  title "DATA SET &dsn";
24  proc contents data=&dsn;
SYMBOLGEN: Macro variable DSN resolves to HOSPITAL
25  run;
NOTE: The PROCEDURE CONTENTS used 0.05 seconds.
26  proc print data=&dsn(obs=5);
SYMBOLGEN: Macro variable DSN resolves to HOSPITAL
27  run;
NOTE: The PROCEDURE PRINT used 0.05 seconds.
```

Notice that there is an extra statement in line 21, OPTIONS SYMBOLGEN ❶. That statement causes the SYMBOLGEN statements in the LOG ❷. Each time a macro variable is encountered, you will see its value in the LOG. This option is useful when you first start to use macros or when you are debugging SAS macro code. You will probably run most of you SAS run jobs with the option set to NOSYMBOLGEN.

If you have not noticed yet, a macro variable is merely stored text. Hopefully that takes some of the mystery out of the word macro. The rules associated with macro variables are similar to those you already know about normal variables...

- The name of a macro variable can be from one to thirty-two characters.
- The name must begin with a letter or an underscore.
- Only letters, numbers, or underscores can follow the first letter.
- The content of macro variable can be up to 64K.
- No macro variable can begin with begin with SYS.
- No macro variable can have the same name as a SAS-supplied macro or macro function.

Now that you know that a macro variable can hold up to 64K characters, %LET DSN=LAB must seem pretty paltry. Let's start using more of that 64K. What if you had two groups of statements that you commonly used to put observations into groups based on a variable named AGE.

### Example 2...using a macro function

```
%let age4=%str( ❶  
    if age < 20 then group = '<20  ';  
else if age < 35 then group = '20-34';  
else if age < 45 then group = '35-44';  
else  
    group = '45+' ❷);  
%let age2=%str(  
if age < 35 then group = '<35';  
else  
    group = '35+';❷);
```

Notice that macro function %STR ❶ is used to create the macro variables &AGE4 and &AGE2. The stored text contains semi-colons. Without the %STR function, the text assigned to each macro variable would end at the first semi-colon. With the %STR function, all the text with the parentheses is assigned to the macro variable. Whenever you use &AGE4 or &AGE2, SAS will substitute the stored text associated with the macro variables, for example...

```
data test;  
input age @@;  
*** use a macro variable to insert age edits;  
&age4;  
datalines;  
19 50 10 34 44 15 30  
;  
run;  
  
proc print data=test;  
run;
```

OBS	AGE	GROUP
1	19	<20
2	50	45+
3	10	<20
4	34	20-34
5	44	35-44
6	15	<20
7	30	20-34

In addition to SYMBOLGEN, there is a macro programming statement that allows you to selectively write the contents of a macro variable to the SAS LOG. The %PUT statement is the macro equivalent to the normal PUT statement. Here is your first macro test, and don't be surprised if you get the wrong answer. How many lines will be written to the LOG file using the PUT and %PUT statements in the following data step?

### Example 3...when is macro code 'resolved'

```
data test;  
input x @@;  
put 'THE VALUE OF X IS: ' x; ❶  
%put TODAY IS &SYSDATE; ❷  
%put 'THE VALUE OF X IS: ' x; ❸  
  
datalines;  
1 2 3 ❹  
;  
run;
```

Was your answer 9? That seems logical in that SAS makes three passes through the data step (there are three values of x to read ❹) and there are three PUT or %PUT statements ❶. Three times three, yes that is still nine. It is also wrong. The answer is five. Here is the LOG...

```
450 data test;  
451 input x @@;  
452 put 'THE VALUE OF X IS: ' x; ❶  
453 %put TODAY IS &SYSDATE; ❷  
TODAY IS 13OCT05 ❸  
454 %put 'THE VALUE OF X IS: ' x; ❷  
'THE VALUE OF X IS: ' x ❸
```

```
455
456 datalines;
```

```
THE VALUE OF X IS: 1 ④
THE VALUE OF X IS: 2
THE VALUE OF X IS: 3
```

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

NOTE: The data set WORK.TEST has 3 observations and 1 variables.

NOTE: DATA statement used (Total process time):

```
real time      0.21 seconds
cpu time       0.04 seconds
```

```
458 ;
459 run;
```

Normal data step operation consists of a syntax check/compile, then execution. If macro variables are present, there is an extra step and it occurs first, macro resolution. All the macro resolution occurs before any statements are executed. The PUT statement ① wrote three lines to the log during the execution step ④. However, the two %PUT statements ② write to the LOG prior to execution not during execution of the data step ③. In the first %PUT, a SAS-supplied macro variable used, &SYSDATE (the stored text is the current date). Notice that each %PUT wrote exactly what was requested, with the second %PUT even writing the quotes. The

Just as PUT can be used with SAS-supplied variables...

```
put _numeric_;
put _character_;
put _all_;
```

%PUT can be used with SAS-supplied macro variables (you've just seen &SYSDATE)...

```
%put _all_;
%put _automatic_;
%put _user_;
%put _local_;
%put _global_;
```

\_ALL\_ is self-explanatory (if not, it is all macro variables). \_AUTOMATIC\_ represents all the SAS-supplied macro variables (e.g. &SYSDATE). \_USER\_ is the list of the macro variables that you create. For now, it is not worth discussing \_LOCAL\_ and \_GLOBAL\_ (not that local versus global environments is not an important concept, but not right now).

Since macro variables are merely stored text, you can join macro variables with other macro variables (you are just combining text strings) or with plain text (again, you are just combining text).

#### Example 4...macro variable as a suffix

```
data all;
input gender : $1. @@;
datalines;
M F M F M F M M M
;
```

```
%let gender=M;
```

```
data just_&gender;
set all;
where gender eq "&gender";
run;
```

The LOG...

```
477 data all;
478 input gender : $1. @@;
479 datalines;
```

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

NOTE: The data set WORK.ALL has 9 observations and 1 variables.

NOTE: DATA statement used (Total process time):

```
real time      0.04 seconds
cpu time       0.04 seconds
```

```
481 ;
482 run;
483
484 %let gender=M;
485
486 data just_&gender; ❶
487 set all;
488 where gender eq "&gender"; ❷
489 run;
```

NOTE: There were 6 observations read from the data set WORK.ALL.

```
WHERE gender='M';
```

NOTE: The data set WORK.JUST\_M has 6 observations and 1 variables. ❸

NOTE: DATA statement used (Total process time):

```
real time      0.09 seconds
cpu time       0.10 seconds
```

When a macro variable is used as a suffix, you merely append it to the end of any text in your SAS code ❶. Notice the name of the new data set, it is JUST\_M ❸. The text stored in the macro variable gender was concatenated (joined) with JUST\_. There are six observations in JUST\_M since the six of nine observations in data set ALL with a value of M are accepted by the where statement ❷ (M is substituted for &GENDER, and notice that double quotes are used in the where statement option).

#### Example 5...macro variable as a prefix

```
%let month=JAN;
%let year=96;
data &month&year (keep=x y z);
infile "e:\19&year\all_&year..dat";
input (month x y z) ($3. 3*3.);
if month eq "&month";
run;
```

The LOG...

```
1083 %let month=JAN;
1084 %let year=96;
1085 data &month&year (keep=x y z); ❶
1086 infile "e:\19&year\all_&year..dat"; ❷
1087 input (month x y z) ($3. 3*3.);
1088 if month eq "&month";
1089 run;
```

NOTE: The infile "e:\1996\all\_96.dat" is:

```
FILENAME=e:\1996\all_96.dat,
RECFM=V,LRECL=256
```

NOTE: 961 records were read from the infile "e:\1996\all\_96.dat". ❸

```
The minimum record length was 21.
```

```
The maximum record length was 21.
```

NOTE: The data set WORK.JAN96 has 193 observations and 3 variables. ❹

NOTE: The DATA statement used 0.33 seconds.

In this example, macro variables are used as both a prefix and a suffix. In line 1085 of the LOG, the macro variable &MONTH is a prefix while &YEAR is a suffix ❶. Later in the LOG, you can see the data step created data set JAN96 ❹ (&MONTH&YEAR). Macro variable &YEAR is used again in line 1086, both as a prefix and suffix ❷. Just after the E:\19 and ALL\_, &YEAR is added to form E:\1996 and ALL\_96 respectively ❸. Notice that two periods are used to complete the file name in the infile statement. SAS interprets a period at the end of a macro variable name (&YEAR.) as a concatenation operator (just like the || used to join character variables/text).

If the INFILE statement had been...

```
infile "e:\19&year\all_&year.dat";
```

without the extra period to join text to the macro variable &YEAR, the LOG would contain...

```
ERROR: Physical file does not exist, e:\1996\all_96dat.
```

SAS interprets one period as concatenation and there is no period left to insert between ALL\_96 and the text DAT. There are instances when you must make SAS 'understand' when the name of a macro variable ends. That is when to use a PERIOD as a 'macro concatenation operator'.

#### Example 6...macro concatenation operator

```
%let yr=96;
data data&yr.m ❶
  datab&yr.f;
set all (where=(year eq "19&yr"));
if gender eq 'M' then output data&yr.m; ❶
else          output data&yr.f;
run;
```

The LOG...

```
1188 %let yr=96;
1189 data data&yr.m data&yr.f;
1190 set all (where=(year eq "19&yr"));
1191 if gender eq 'M' then output          data&yr.m;
1192 else          output          data&yr.f;
1193 run;
NOTE: The data set WORK.DATA96M has 150 observations and 4 variables.
NOTE: The data set WORK.DATA96F has 43 observations and 4 variables.
NOTE: The DATA statement used 0.59 seconds.
```

In this example, a period is used each time reference is made to a data set ❶. The period tells SAS that the name of the macro variable being used is &YR, not &YRM or &YRF. Without the period (using DATA&YRM and DATA&YRF, not DATA&YR.M and &DATA&YR.F), a portion of the LOG would show...

```
1195 %let yr=96;
WARNING: Apparent symbolic reference YRM not resolved.
1196 data data&yrm data&yrf;
          -          -
          200      200
WARNING: Apparent symbolic reference YRF not resolved.
1197 set all (where=(year eq "19&yr"));
1198 if gender eq 'M' then output          data&yrm;
          -
          200
WARNING: Apparent symbolic reference YRM not resolved.
1199 else output data&yrf;
          -
          200
WARNING: Apparent symbolic reference YRF not resolved.
1200 run;
```

SAS is looking for macro variables &YRM AND &YRF and they do not exist.

#### MACRO PROGRAMS

Thus far, we have stored text in macro variables. You can also store text in macro programs. Macro programs offer more programming flexibility than macro variables in that you can pass information to parts of the macro program using macro parameters. If that is a little confusing right now, you'll see what it really means using some examples.

Starting slowly, let's take the age edits that were shown in example 2 and change them from macro variables to macro programs.

### EXAMPLE 7...simple macro programs

```
%macro age4; ❶  
    if age < 20 then group = '<20  ';  
    else if age < 35 then group = '20-34';  
    else if age < 45 then group = '35-44';  
    else  
        group = '45+' ;  
%mend; ❷
```

```
%macro age2; ❶  
    if age < 35 then group = '<35';  
    else  
        group = '35+';  
%mend; ❷
```

Instead of creating macro variables &AGE4 and &AGE2, we have created two macro programs, i.e. %AGE4 and %AGE2 ❶. The creation of a macro program starts with a %MACRO statement and ends with a %MEND ❷ statement. At a minimum, the %MACRO statement also contains the name of the macro program being created. Once the macro programs are created and submitted (run) once, they can be invoked during the SAS session using the name of the macro program preceded by a %. Comparing example 2 and example 7, we can insert the edits in a data step by either using macro variables (&AGE4, &AGE2) or using macro programs (%AGE4, %AGE2)...

```
options mprint; ❶  
  
data test;  
input age @@;  
*** use a macro program to insert age edits;  
%age4;  
datalines;  
19 50 10 34 44 15 30  
;  
run;
```

The LOG...

```
11  options mprint;  
12  data test;  
13  input age @@;  
14  *** use a macro program to insert age      edits;  
15  %age4;  
MPRINT(AGE4):  IF AGE < 20 THEN GROUP = '<20  ';  
MPRINT(AGE4):  ELSE IF AGE < 35 THEN GROUP = '20-34';  
MPRINT(AGE4):  ELSE IF AGE < 45 THEN GROUP = '35-44';  
MPRINT(AGE4):  ELSE GROUP = '45+' ;  
16  datalines;  
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.  
NOTE: The data set WORK.TEST has 7 observations and 2 variables.  
NOTE: The DATA statement used 0.59 seconds.  
18  ;  
19  run;
```

When we used macro variables, we used the SYMBOLGEN option to show the values of the macro variables in the LOG. The MPRINT option ❶ is used to display the contents of a macro program in the LOG when the program is used anywhere in a SAS job. Rather than creating two macro programs (one for each age group), we can create one and use a macro variable to select the number of age groups when the macro is used in a data step.

### EXAMPLE 8...macro program with one parameter

```
%macro age_edit(groups); ❶
  %if &groups eq 4 %then %do;
    if age < 20 then group = '<20  ';
  else if age < 35 then group = '20-34';
  else if age < 45 then group = '35-44';
  else
    group = '45+' ;
  %end;
  %else %do;
    if age < 35 then group = '<35';
    else
      group = '35+' ;
  %end;
%mend;
```

In addition to the macro name (AGE\_EDIT) specified in the %MACRO statement, we now also have a macro parameter, GROUPS ❶. Inside the macro program, the macro parameter is used as macro variable. As opposed to creating a macro variable using %LET in open SAS code, using a macro parameter to create a macro variable limits the use of the macro variable to the macro program, not the entire SAS job. Earlier in the paper, we mentioned LOCAL and GLOBAL macro variables. The macro variable &GROUPS in this example is LOCAL to the macro program AGE\_EDIT. In the earlier examples, %LET created macro variables that were GLOBAL, usable anywhere in a SAS job.

We can use the new macro program in a data step and specify either four or two age groups...

```
options mprint;
data test;
input age @@;
*** use a macro program with a parameter;
%age_edit(2);
datalines;
19 50 10 34 44 15 30
;
run;
```

The LOG...

```
33  options mprint;
34  data test;
35  input age @@;
36  *** use a macro program with a parameter;
37  %age_edit(2);
MPRINT(AGE_EDIT):  IF AGE < 35 THEN GROUP = '<35'; ❶
MPRINT(AGE_EDIT):  ELSE GROUP = '35+';
38  datalines;
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.TEST has 7 observations and 2 variables.
NOTE: The DATA statement used 0.32 seconds.
40  ;
41  run;
```

Even though the macro program had ten lines of SAS code, we see only two in the LOG ❶. Remember that the macro resolution (text insertion) occurs prior to execution. All the logic in the macro program is carried out before anything else happens in the data step. The logic says that if the macro parameter is anything other than a 4, the statements that put the variable age into two groups should be used. They are the only two statements you see in the LOG (using the MPRINT option). The macro programming statements %IF, %THEN, %ELSE, %DO, and %END look and work just like their non-macro counterparts.

If the AGE\_EDIT macro program is used mostly to put observations into four age groups, you can use a KEYWORD macro parameter...

#### EXAMPLE 9...macro program with one keyword parameter

```
%macro age_edit(groups=4);
  %if &groups eq 4 %then %do;
    if age < 20 then group = '<20 ' ;
  else if age < 35 then group = '20-34';
  else if age < 45 then group = '35-44';
  else
    group = '45+' ;
  %end;
  %else %do;
    if age < 35 then group = '<35';
    else
      group = '35+' ;
  %end;
%mend;
```

Now, if we use the macro program without specifying a value for the parameter groups, the value will default to 4.

```
data test;
input age @@;
*** use a macro program with(out)a keyword parameter;
%age_edit;
datalines;
19 50 10 34 44 15 30
;
run;
```

The LOG...

```
142 data test;
143 input age @@;
144 *** use a macro program with(out) a keyword parameter;
145 %age_edit; ❶
MPRINT(AGE_EDIT):  IF AGE < 20 THEN GROUP = '<20 ' ;
MPRINT(AGE_EDIT):  ELSE IF AGE < 35 THEN GROUP = '20-34';
MPRINT(AGE_EDIT):  ELSE IF AGE < 45 THEN GROUP = '35-44';
MPRINT(AGE_EDIT):  ELSE GROUP = '45+' ;
146 datalines;
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.TEST has 7 observations and 2 variables.
NOTE: The DATA statement used 0.33 seconds.
148 ;
149 run;
```

Since no parameter value was used in line 145 ❶, the value of groups defaulted to four and the logic of the macro program inserted four lines of SAS code into the data step.

Let's go back to an earlier example and write another program, but with two keyword parameters. The first example in this paper showed how to use %LET to create a macro variable that was used to change the name of a data set in a title statement and two PROCs (CONTENTS and PRINT).

#### EXAMPLE 10...macro program with two keyword parameters

```
%macro printit(dsn=LAB,nobserv=5);
proc contents data=&dsn;
run;

title "DATA SET &dsn";

proc print data=&dsn(obs=&nobserv);
run;
%mend;
```

We can run PROC CONTENTS and print five observations in data set LAB by using...

```
%printit;
```

since, if no parameters are specified, the macro program will default to using data set LAB and 5 for the number of observations. We can also change the data set name, but still print only 5 observations if we use...

```
%printit(dsn=hospital);
```

Or, you can use both parameters to print a given number of observations in any data set...

```
%printit(dsn=hospital,nobserv=100);
```

In example 8, we wrote a macro program with one parameter. It was not a KEYWORD parameter since no default value was associated with the macro variable. It was a POSITIONAL parameter, though with only one parameter, it is not easy to see what POSITIONAL means. We can take the macro written in example 10 and use POSITIONAL rather than KEYWORD parameters (we will also modify the macro code to perform a slightly different task).

#### EXAMPLE 11...macro program with three positional parameters

```
%macro printit(dsn,first,howmany); ❶  
%let lastobs=%eval(&first + &howmany - 1); ❷  
title "DATA SET &dsn";  
proc print data=&dsn(firstobs=&first obs=&lastobs);  
run;  
%mend;
```

Let's use this macro before explaining what all the code means...

```
data lab;  
input x @@;  
datalines;  
888 348 38 216 47 5552 4563 6544 45 336 722 81 2223 3444 999  
;  
run;  
  
options mprint;  
  
%printit(lab,6,5);
```

A portion of the LOG and all the OUTPUT...

```
322 options mprint;  
323 %printit(lab,6,5);  
MPRINT(PRINTIT): TITLE "DATA SET lab";  
MPRINT(PRINTIT): PROC PRINT DATA=LAB(FIRSTOBS=6 OBS=10);  
MPRINT(PRINTIT): RUN;  
DATA SET lab  
OBS X  
6 5552  
7 4563  
8 6544  
9 45  
10 336
```

It is easier to see what POSITIONAL parameter means. The macro value specified first (labs) is used as the text for the first parameter (macro variable &DSN) ❶. The second (6) and third (5) values are used as the text for the second (&FIRST) and third (&HOWMANY) parameters respectively. The next line shows how to do integer math with macro variables ❷. As you already know, the data set options FIRSTOBS and OBS allow you to specify the first observation and last observation of a given data set to be used in a PROC or data step. It is easy to forget that OBS really means last observation, not the number of observations to be used (I have always wondered why it is not called LASTOBS...). The new macro allows you to specify the number of observations to be printed rather than the last observation.

The statement...

```
%let lastobs=%eval(&first + &howmany - 1);
```

computes the last observation for you, given that you specify the first (&FIRST) and number of observations (&HOWMANY) to be printed. Note, if we had written...

```
%let lastobs=&first + &howmany - 1;
```

the value of the macro variable &LASTOBS would be the text... 6 + 5 - 1, not the text 10. Without the macro function %EVAL, you are placing the text on the right side of the = into macro variable &LASTOBS. With the %EVAL, you are instructing SAS to perform integer math on the text within the parentheses. Remember that if you write a macro program using POSITIONAL parameters, a value for each parameter must be present when you use the macro (and in the proper location).

One last thing to look at is the title. Since we used lower case letters to specify the data set (labs) when using the %PRINTIT macro program, the data set name is in lower case in the title above the PROC PRINT output.

### ONE OTHER WAY TO CREATE A MACRO VARIABLE

So far, you have learned how to do several of the tasks cited earlier in the section *WHAT CAN YOU DO WITH MACROS*. One thing you have not learned how to do is to pass information from one part of a SAS job to another. You now know that you can store text in a macro variable using a %LET statement. However, if you want to store the value of a variable as the text of a macro variable, you can use a SAS CALL statement.

### EXAMPLE 12...store the value of a variable as macro variable text

```
data mydata;
input x @@;
datalines; ❶
11 22 33 44 55 66 77 88 99 100 110 22
;
run;

data _null_;
call symput('observ',numobs); ❷
stop;
set mydata nobs=numobs; ❸
run;

title "FIRST 3 OF &observ OBSERVATIONS IN DATA SET MYDATA";
proc print data=mydata (obs=3);
run;
```

A portion of the LOG...

```
540 data _null_;
541 call symput('observ',numobs);
542 stop;
543 set mydata nobs=numobs;
544 run;
NOTE: Numeric values have been converted to character values at the places given by:
(Line):(Column).
      541:22
NOTE: The DATA statement used 0.05 seconds.
```

The OUTPUT...

```
FIRST 3 OF          12 OBSERVATIONS IN DATA SET MYDATA
OBS      X
  1     11
  2     22
  3     33
```

There are twelve values of X in the datalines portion of the data step ❶. A DATA \_NULL\_ step is used to determine the number of observations in the data set MYDATA. The option NOBS on the SET statement ❸ places the number of observations in variable NUMOBS prior to execution of the data step. During execution, the first executable statement is CALL SYMPUT ❷. This CALL routine is used to store the value of variable NUMOBS as the text of macro variable &OBSERV. There is a NOTE in the LOG ❹ telling you that a numeric value was converted to a character value in line 541, the location of

CALL SYMPUT. Remember that macro variables are merely stored text. If you try to use numeric data (the variable NUMOBS is numeric) as that stored text, SAS must convert the numeric data to character data prior to storing it as the value of the macro variable. Look at the title. When you let SAS perform numeric-to-character conversion, it is the same as if you had done the conversion with a PUT function as a BEST12. format...

```
char_var = put(num_var,best12.);
```

The number 12 in the title is preceded by 10 spaces, the result of SAS using the BEST12. format in the numeric to character conversion. You can get rid of the extra spaces if you make use of another macro function in the TITLE statement...

```
title "FIRST 3 OF %left(&observ) OBSERVATIONS IN DATA SET MYDATA";
```

You can eliminate the numeric-to-character conversion and the leading spaces on the text by using...

```
call symput('observs',trim(left(put(numobs,best.))));
```

or by using a new CALL statement (SYMPUTX), only available as of version 9...

```
call symputx('observ',numobs);
```

## STORING MACRO VARIABLES AND MACRO PROGRAMS

I'll share my favorite macro variable with you...

```
%let np=norow nocol nopercnt;
```

You might recognize the text on the right as what you must type to suppress all the percentages in PROC FREQ. I got tired of writing all that text, so I placed the above %LET statement in the AUTOEXEC.SAS file in the SAS root directory on my PC. Each time I start SAS, all SAS statements in the AUTOEXEC.SAS file execute. In a SAS session, I can use &NP (three characters) instead of the twenty-one characters on the right. If you use SAS on a PC and have some control over the SAS installation, the AUTOEXEC.SAS file is a convenient place to create commonly used macro variables (created using %LET).

Storing macro programs is a little more involved. Assume that you are working on a PC and store all your macro programs in a directory (or, if you prefer, folder) named C:\SAS\MYMACROS. In example 10, we created a macro program named PRINTIT. If you store that macro program in a file with the name PRINTIT.SAS, you can use the PRINTIT macro anytime during a SAS session if you place the following line in the file AUTOEXEC.SAS (or submit the line of SAS code during your SAS session)...

```
options sasautos=("c:\sas\mymacros",!sasroot\core\sasmacro");
```

The SASAUTOS option tells SAS where you have stored macro programs. The first directory in the list is where your self-written macros are stored. The second directory contains SAS-supplied macros. In both of these directories, each macro program is stored in a file with a name that is the same as the stored macro program and with a .SAS file extension. Once again, that means that the PRINTIT macro program would be stored in a file with the name PRINTIT.SAS. If there are files (macro programs) with the same name in the both of the listed directories, the one in MYMACROS would be run since that directory is first in the search list. Another way to refer to the collection of directories specified after SASAUTOS is the AUTOCALL library.

## FINAL THOUGHT

This paper is introductory, but hopefully you've read and learned enough to make you realize that macro variables and macro programs are very useful. If you do more reading about macros, make sure that you learn about...

- %SYSFUNC
- using PROC SQL to create macro variables using SELECT INTO

You can learn a lot about macros from two excellent books in the SAS books by users series (I give much credit to the introductory chapters in the Art Carpenter book for helping me to put this paper together.).

***Carpenter's Complete Guide to the SAS Macro Language (2nd ed)***, by Art Carpenter (SAS Publication #59224)

***SAS Macro Programming Made Easy***, by Michele Burlew (SAS Publication #56516)

Most of the SAS code (plus data sets) used in both books is available at the SAS Institute web site...

<http://www.sas.com/service/doc/code.samples.htm>

If you are a SAS/GRAPH user, you might also want to look at... **SAS System for Statistical Graphics**, by Michael Friendly (SAS Publication #56143)

It contains a large number of macros written to augment and enhance what you can already do with SAS/GRAPH procedures. The macros in Michael Friendly's book are also posted on his web site...

<http://www.math.yorku.ca/SCS/friendly.html>

Any questions about this paper can be directed to the author at... [msz03@albany.edu](mailto:msz03@albany.edu)

## APPENDIX

The following are modifications of EXAMPLE11. The following macro adds default values to the positional parameters.

### EXAMPLE\_A...default values for positional parameters

```
%macro printit(dsn,first,howmany);
%if &dsn      eq   %then %let dsn=&syslast; ❶
%if &first    eq   %then %let first=1; ❷
%if &howmany  eq   %then %let howmany=5; ❸

%let lastobs=%eval(&first+&howmany-1);

title "data set &dsn";
proc print data=&dsn (firstobs=&first obs=&lastobs) obs="observation number";
run;
%mend;

data lab; ❹
input x @@;
datalines;
888 348 38 216 47 5552 4563 6544 45 336 722 81 2223 3444 999
;
run;

%printit; ❺
%printit(,6); ❻
%printit(,,10); ❼
%printit(all,20,10); ❽
```

Default values were added previously using keyword parameters, not positional. If no data set is specified, the macro will print the last created data set (&SYSLAST) ❶. If no starting observation (FIRSTOBS) is specified, the starting observation will be the first in the data set ❷. If no number of observations to be printed is specified, five observations will be printed ❸.

Data set LAB is created ❹. The first use of the macro specifies no parameters ❺, thus all the defaults will be used and the first 5 observations in data set LAB (the most recently created data set) are printed. One parameter is specified in the next use ❻ and observations 6 through 10 in data set LAB are printed. The next use results in the first 10 observations in data set LAB being printed ❼. Finally, the last use of the macro prints observations 20 through 29 in data set ALL ❽.

The next version allows a user to select a simple random sample rather than consecutive observations.

#### EXAMPLE\_B...add SIMPLE RANDOM SAMPLING to EXAMPLE\_A

```
%macro printit(dsn,first,howmany);
%if &dsn      e      %then %let dsn=&syslast;
%if &first    eq     %then %let first=1;
%if &howmany  eq     %then %let howmany=5;

%if %upcase(&first) eq R %then %do; ❶
  proc surveyselect data=&dsn out=_tempdata_  sampsize=&howmany outall noprint; ❷
  run;

  title "random sample from data set &dsn";
  proc print data=_tempdata_ obs="observation number";
  where selected eq 1; ❸
  by selected; ❹
  run;

  proc datasets lib=work nolist; ❺
  delete _tempdata_;
  quit;
%end;
%else %do; ❻
  %let lastobs=%eval(&first+&howmany-1);

  title "data set &dsn";
  proc print data=&dsn (firstobs=&first obs=&lastobs) obs="observation number";
  run;
%end;
%mend;
```

A user of the macro is now allowed to specify an 'r' or and 'R' as the second macro parameter to print a simple random sample(SRS) of a data set ❶. If an 'r' is specified, the %UPCASE function will convert it to uppercase prior to checking if the value is equal to 'R'. PROC SURVEYSELECT is used to take a SRS ❷. All other parameters remain the same. If an SRS is taken, the number of observations selected is specified by the last parameter (either the default of 5, or user-selected). All observations in the original data set are in the sample, with the sample flagged by having a new variable in the data set, SELECTED, have a value of 1. This is the result of using the OUTALL option in PROC SURVEYSELECT. A where statement asks PROC PRINT to only print the sample portion of the data set ❸. The BY statement takes the variable SELECTED out of the listing of observations (repeated many times) and places it at the top of the listing (only once) ❹. The sample data set is deleted after it is printed ❺. If a user has not asked for a SRS, the macro works as shown in the previous example ❻.