The George Washington University
Department of Statistics

STAT 6197 – Spring 2020
Week 9 – March 13, 2020

Major Topic: Macro Facility Basics

Detailed Topics:

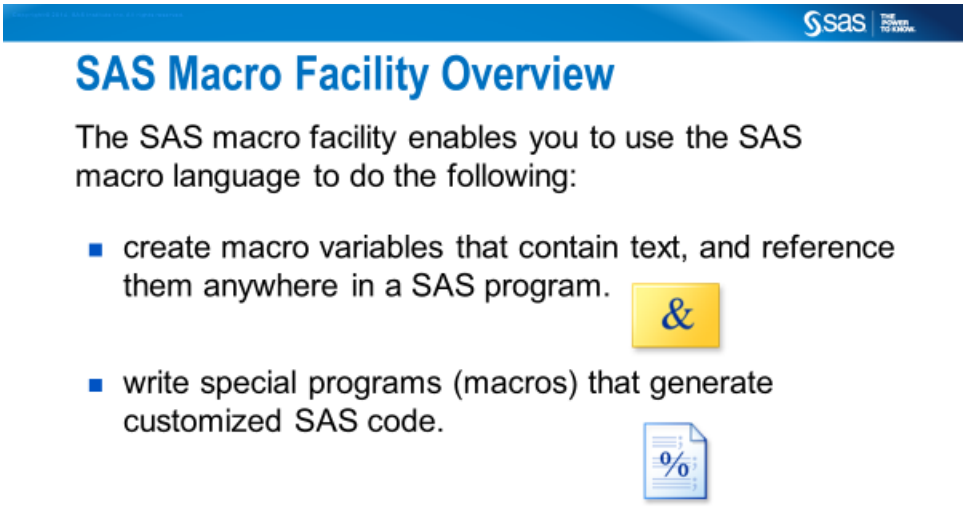
1) Creating, Referencing, and Displaying Macro Variables
2) Converting Macro Variables into Data Step Variables
3) Defining, Compiling, and Calling Macro Programs
4) Macro Loops
5) Creating Macro Arrays
6) Saving Macros and Accessing Them for Later Use
7) Deleting Macro Variables

Readings:

1. Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012

2. Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015

.



More specifically, the **macro facility** does the following:

- Dispalying system information
- Symbolic substitution within the SAS code
- Automated production of SAS code (automation for repetitive tasks)
- Conditional construction of SAS code
- Dynamic generation of SAS code (data-driven applications)

SAS Macro Facility = Macro Processor + Macro Language

Macro Processor: The Part of Base SAS that performs the macro activity**.**

Macro Language: The syntax that interacts with the macro processor

Levels of Learning of Macro Language

- Code Substitutions
- Macro Statement and Functions
- Dynamic Programming

(See Carpenter, A. 2016. Carpenter's Complete Guide to the SAS Macro Language. Third Edition)

Why Using Macro Language?

- To avoid same pieces of code repeatedly
- To minimize typing when writing the SAS program

The SAS macro language enables you generate the same piece of code (text) such as

- data set name
- data value
- variable name
- SAS statement
- part of the SAS statement

at different parts of your SAS program multiple times without you having to type the same code/text over and over again provided you assign that text to a macro variable [%LET DS=CLASS;] once and then reference it [&DS] repeatedly.

With the SAS macro language, you can

- re-use the same code multiple times after defining it once in your SAS program
- change the code once at a single place in your SAS program and reflect them at multiple parts of your SAS program
- make your program data driven by letting SAS decide what code to generate based on the data values.

You can create multiple macro variables that contain text and then reference them anywhere in a SAS program.

Words in SAS Language

In SAS, there are four basic types of words or tokens.

Name	Literal (i.e., characters enclosed in quotation marks)	Number	Special characters
• DATA • _var • FIRSTOBS • year_99 • descending • _n_ • Percent8.2	• 'Stat6197' • "1990-91" • 'Mary Delany' • "Final Exam"	• 3283 • 8.05 • 0b0x • -5 • 6.4E-1 • '04July18'd	• = • ; • ' • + • @ • /

What Are Macro Variables?

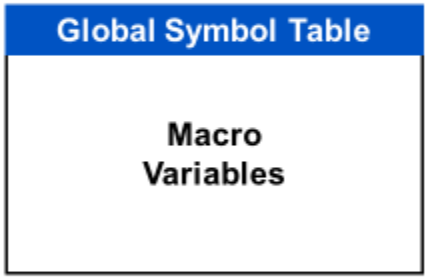
Macro variables store text, including the following:

- complete or partial SAS statements
- complete or partial SAS steps

5

How Are Macro Variables Stored?

Macro variables are stored in a memory area called the *global symbol table*.



6

Program Flow

When you submit a SAS program, it is copied to a memory location called the *input stack*.

- A SAS program can be any combination of the following:
  - DATA steps and PROC steps
  - global statements
  - Structured Query Language (SQL)
  - SAS Component Language (SCL)
  - SAS macro language

38

%LET Statement Examples

Use macro variables to select ***date*** values.

```
%let date1=25may2011;
%let date2=15jun2011;

proc print data=orion.Order_Fact;
   where Order_Date between "&date1"d and "&date2"d;
   var Order_Date Product_ID Quantity;
   title "Orders between &date1 and &date2";
run;
```

Partial SAS Log

```
NOTE: There were 11 observations read from the data set
      ORION.ORDER_FACT.
      WHERE (Order_Date>='25MAY2011'D and Order_Date<='15JUN2011'D);
```

Orders between 25may2011 and 15jun2011

Obs	Order_ Date	Product_ID	Quantity
526	05JUN2011	240100100679	2
527	06JUN2011	230100400007	1

89

m102d04

The %LET statement is used to create two macro variables (date1 and date2) manually and assign a value to each of them (25may2011 and 15jun2011, respectively) in open code. Macro variable references begin with an & followed by a macro variable name (e.g., &date1 and &date2). Double quotes surrounding the macro variable references are required for the macro processor to resolve them into macro variable values.

Automatic Macro Variables

When a SAS job or session begins, the global symbol table is created and initialized with automatic macro variables.

Automatic Variables

Global Symbol Table	
SYSDATE	06JAN12
SYSDATE9	06JAN2012
SYSDAY	Friday
SYSTIME	10:47
SYSUSERID	joeuser
.	.
.	.

7

Automatic Macro Variables

The following are true for automatic macro variables:

- system-defined
- created at SAS invocation
- global in scope (always available)
- assigned values by SAS
- can be assigned values by the user in some cases

16

Automatic Macro Variables

Some automatic macro variables have fixed values that are set at SAS invocation.

Name	Description
SYSDATE	Date of SAS invocation (06JAN12)
SYSDATE9	Date of SAS invocation (06JAN2012)
SYSDAY	Day of the week of SAS invocation (Friday)
SYSTIME	Time of SAS invocation (10:47).
SYSSCP	Operating system abbreviation (WIN, OS, HP 64)
SYSVER	Release of SAS software (9.3)
SYSUSERID	Login or user ID of current SAS process

- The macro variables **SYSDATE**, **SYSDATE9**, and **SYSTIME** store text, not SAS date or time values.

17

Automatic Macro Variables

Some automatic macro variables have values that change automatically based on activity during your SAS session.

Name	Description
SYSLAST	Name of the most recently created SAS data set in the form *libref.name*. If no data set has been created, the value is _NULL_.
SYSPARM	Value specified at SAS invocation.
SYSERR	SAS DATA or PROC step return code (0=success).
SYSLIBRC	LIBNAME statement return code (0=success).

18

Automatic Macro Variables

The _AUTOMATIC_ argument in the %PUT statement writes the names and values of all automatic macro variables to the SAS log.

Partial SAS Log

```
12   %put _automatic_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 3000
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 06JAN12
AUTOMATIC SYSDATE9 06JAN2012
```

%PUT _AUTOMATIC_;

19

2.02 Quiz – Correct Answer

Submit the following statement:

```
%put _automatic_;
```

What is the value of **SYSTIME**?
Does it match the current time?

SYSTIME represents the time of SAS invocation.
It does not necessarily match the current time.

21

User-Defined Macro Variables

User-defined macro variables can be added to the global symbol table.

Global Symbol Table

	.	.
	.	.
Automatic Variables	**SYSTIME**	10:47
	SYSUSERID	joeuser
	.	.
	.	.
User-Defined Variables	OFFICE	Sydney
	DATE1	25may2012
	UNITS	4

The global symbol table is deleted at the end of your SAS job or session.

8

Macro Variable References

Macro variable references begin with an ampersand (&) followed by a macro variable name.

Macro variable references

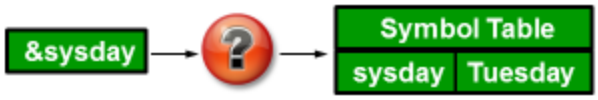
- are also called *symbolic references*
- can appear anywhere in a program
- are not case sensitive
- represent macro triggers
- are passed to the macro processor.

26

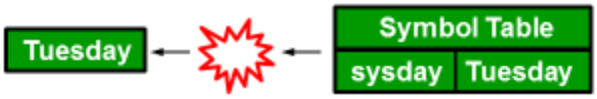
Macro Variable References

When the macro processor receives a macro variable reference, it does the following:

- searches the symbol table for the macro variable



- resolves the macro variable by substituting its value



28

Macro Variable References

Example: Write the day of the week to the SAS log.

SAS Log

```
14    %put Today is &sysday;
Today is Friday
15    %put &=sysday;
SYSDAY=Friday
```

- The form of the %PUT statement that is shown on line 15 is new in SAS 9.3.

30

Macro Triggers

During word scanning, two token sequences are recognized as *macro triggers*:

&*name-token*	a macro variable reference
%*name-token*	a macro statement, function, or call

- The word scanner passes macro triggers to the *macro processor*. The macro processor will compile and execute macro triggers.

32

User-Defined Macro Variables

User-defined macro variables have the following characteristics:

- Macro variable names follow SAS naming conventions.
- If the macro variable already exists, its value is overwritten.
- If the variable or value contain macro triggers, the triggers are evaluated before the %LET statement executes.

```
%let name=Ed Norton;
```

69

User-Defined Macro Variable Values

Macro variable values have the following characteristics:

- Minimum length is 0 characters (null value).
- Maximum length is 65,534 characters (64K).
- Number tokens are stored as text strings.
- Mathematical expressions are not evaluated.
- Case is preserved.
- Leading and trailing blanks are removed.
- Quotation marks, if any, are stored as part of the value.

```
%let name=Ed Norton;
```

70

%LET Statement Examples

The macro variable's name resolves to **varlist**.

```
%let name= Ed Norton ;
%let name2=' Ed Norton ';
%let title="Joan's Report";
%let start=;
%let sum=3+4;
%let total=0;
%let total=&total+&sum;
%let x=varlist;
%let &x=name age height;
```

Global Symbol Table

name	Ed Norton
name2	' Ed Norton '
title	"Joan's Report"
start	
sum	0+3+4
total	0
x	varlist
varlist	name age height

80

Displaying Macro Variables

The _USER_ argument in the %PUT statement writes the names and values of all user-defined macro variables to the SAS log.

SAS Log

```
175  %put _user_;
GLOBAL OFFICE Sydney
GLOBAL DATE1 25may2011
GLOBAL DATE2 15jun2011
GLOBAL UNITS 4
```

%PUT _USER_;

92

Creating and Referencing Macro Variables

Taking advantage of macro variables requires two steps.

1. Create and assign a value to a macro variable using one of these methods:
   - %LET statement in SAS code
   - INTO clause in a PROC SQL query
   - CALL SYMPUTX routine in SAS code

2. Reference the macro variable in SAS code so that SAS can resolve the macro variable value.
   - Use *&macro-name*

50

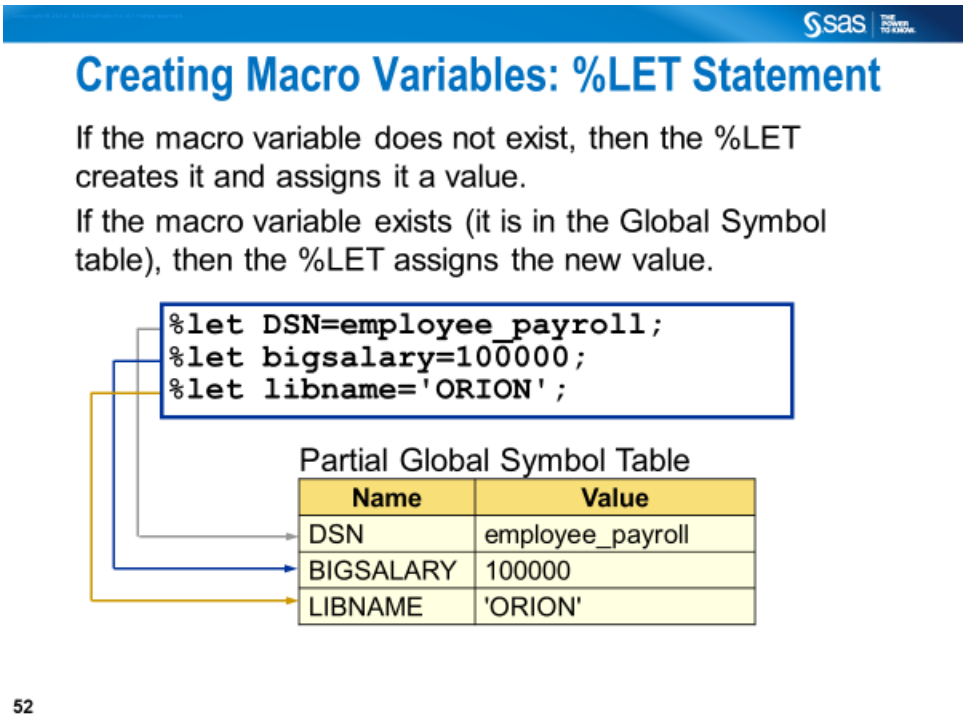
1. The %LET statement is used to create a macro variable manually and assign a value to it.
    - In open code
    - Within a macro program

2. The INTO clause in PROC SQL can automate creating macro variables
3. The CALL SYMPUTX in a data step can also automate creating macro variables.

[SAS Documentation]

Problem Note 51984: A %LET statement might generate unexpected results when used to create a macro variable name in open code

%LET macro statement



Examples of the %LET Statement

Macro variable values have the following characteristics:

- Minimum length is 0 characters (null value).
- Maximum length is 65,534 characters (64K).
- Numbers are stored as text strings.
- Mathematical expressions are not evaluated.
- Case is preserved.
- Leading and trailing blanks are removed.
- Quotation marks, if any, are stored as part of the value.

```
%let year=2007;

%let city=Dallas, TX;
```

Name	Value
year	2007
city	Dallas, TX

18

User-Defined Macro Variables

User-defined macro variables have the following characteristics:

- Macro variable names follow SAS naming conventions.
- If the macro variable already exists, its value is overwritten.
- If the variable or value contain macro triggers, the triggers are evaluated before the %LET statement executes.

```
%let name=Ed Norton;
```

69

Macro Variable References

Macro variable references begin with an ampersand (&) followed by a macro variable name.

Macro variable references

- are also called *symbolic references*
- can appear anywhere in a program
- are not case sensitive
- represent macro triggers
- are passed to the macro processor.

26

&mavar.

A macro variable can also be referenced with an ampersand, the macro variable name, and a period.

Displaying Macro Variable Values

Use the %PUT statement to display the resolved macro variable value along with descriptive text in the SAS log.

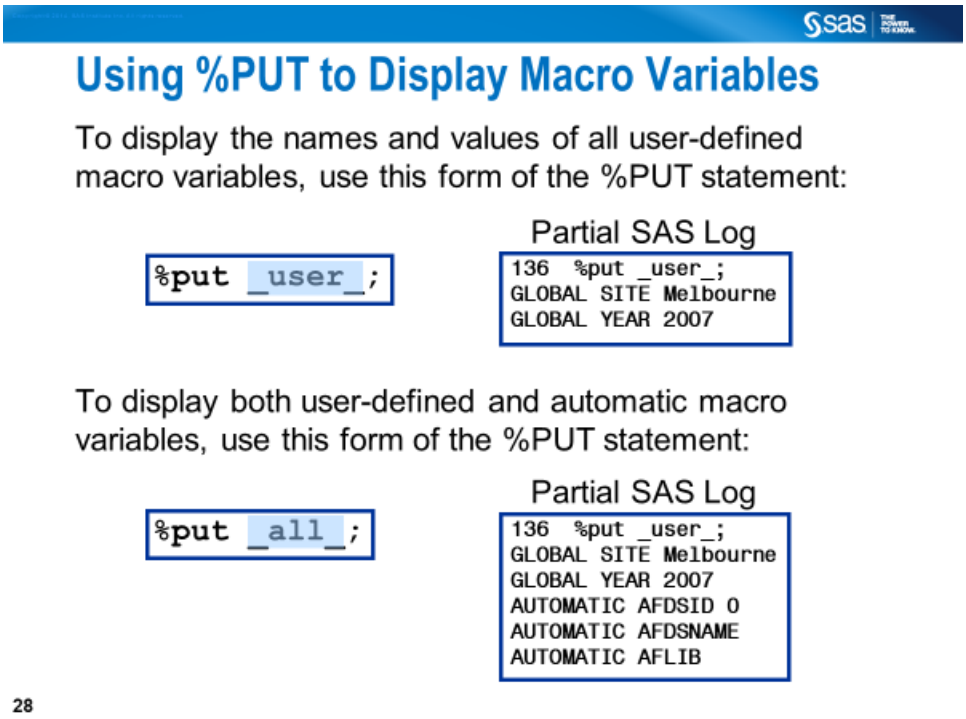
```
%let DSN=employee_payroll;
%let bigsalary=100000;
%put DSN is &DSN;
%put bigsalary is &bigsalary;
```

%PUT *text*;

Partial SAS Log

```
   %put DSN is &DSN;
DSN is employee_payroll
   %put bigsalary is &bigsalary;
bigsalary is 100000
```

64 s108d05



Points to remember: The %PUT statement is valid in open code. It writes to column one of the next line in the SAS log. It writes a blank line if no text is specified.

PUTLOG (or PUT) vs. %PUT

PUTLOG or PUT can write the following to the SAS log

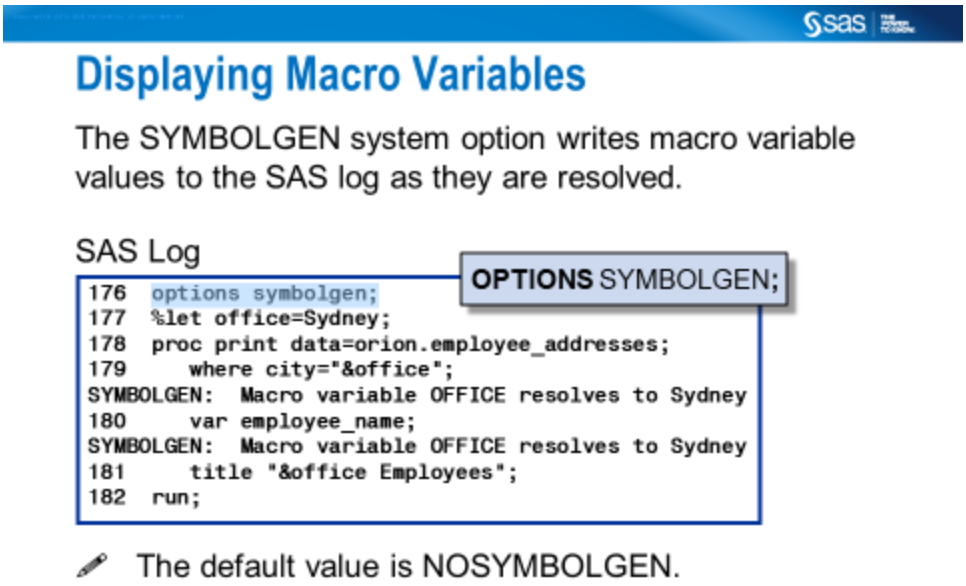
- text strings inside quotation marks
- values of the variables found in the DATA step

- Use the PUTLOG statement to write informational message (includding the debugging message) to the SAS log.

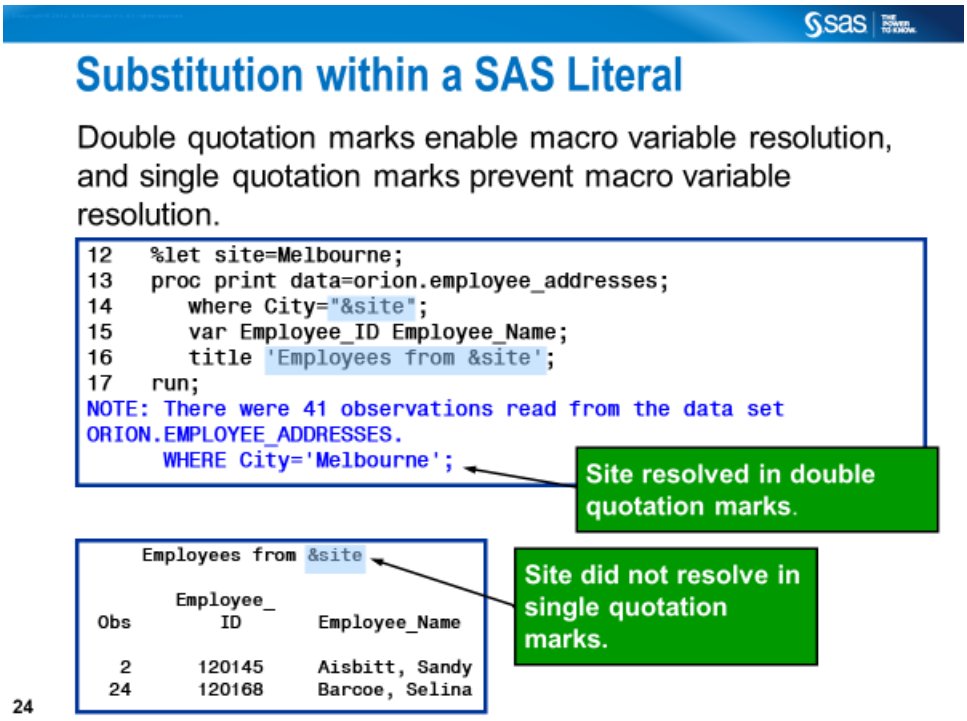
- Use the PUT statement to write to an external file that isspecified in the FILE statement.

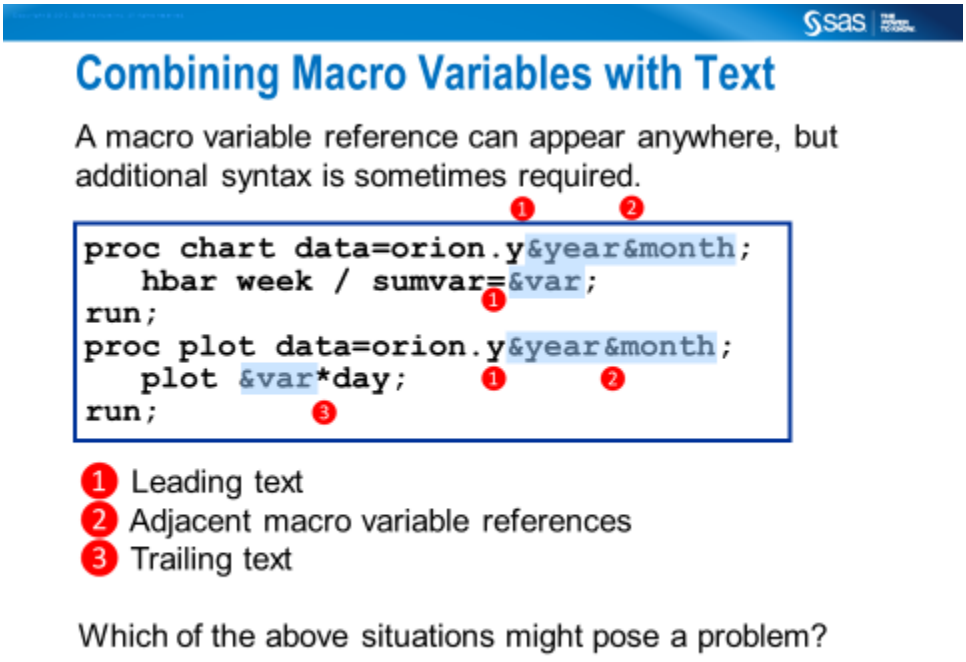
The %PUT statement

- writes text strings and values of the macro variables to the SAS log, starting in column one
- writes a blank line if text is not specified
- does not require quotation marks around text
- is valid in open code
- can appear
    - before the DATA step
    - after the DATA step
    - in the middle of the DATA step



Points to remember: The %PUT statement is valid in open code. It writes to column one of the next line in the SAS log. It writes a blank line if no text is specified.





Leading Text

Leading text is never a problem.

```
%let month=jan;
proc chart data=orion.y2010&month;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y2010&month;
   plot sale*day;
run;
```

```
proc chart data=orion.y2010jan;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
   plot sale*day;
run;
```

101

Adjacent Macro Variable References

Adjacent macro variable references are never a problem.

```
%let year=2010;
%let month=jan;
proc chart data=orion.y&year&month;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y&year&month;
   plot sale*day;
run;
```

```
proc chart data=orion.y2010jan;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
   plot sale*day;
run;
```

102

Trailing Text Solution

A *period* (.) is a special delimiter that ends a macro variable reference.

```
&myvar.
```

special delimiter

The period does not appear as text when the macro variable is resolved.

- The word scanner recognizes the end of a macro variable reference when it encounters a character that cannot be part of the reference.

108

Trailing Text

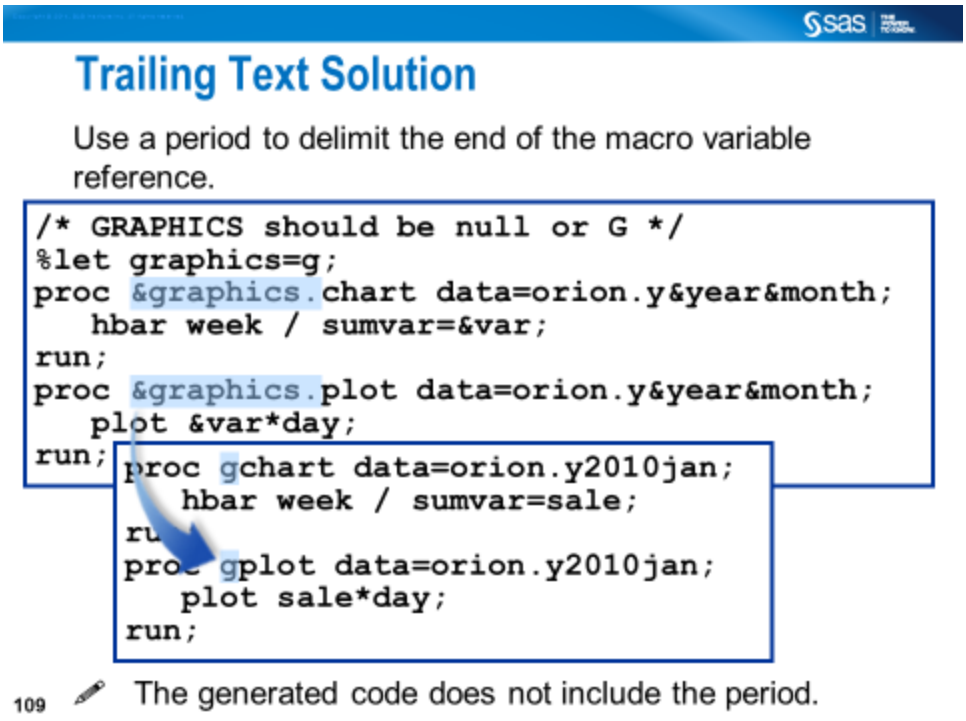
Trailing text can be a problem. Why?

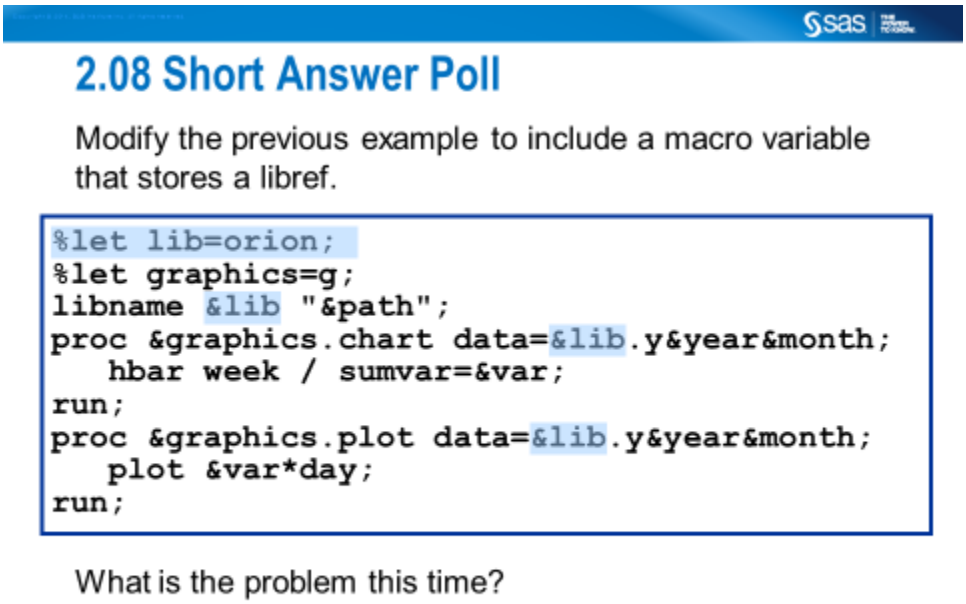
Why is trailing text ***not*** a problem here?

```
%let year=2010;
%let month=jan;
%let var=sale;
proc chart data=orion.y&year&month;
   hbar week / sumvar=&var;
run;
proc plot data=orion.y&year&month;
   plot &var*day;
run;
```

```
proc chart data=orion.y2010jan;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
   plot sale*day;
run;
```

103

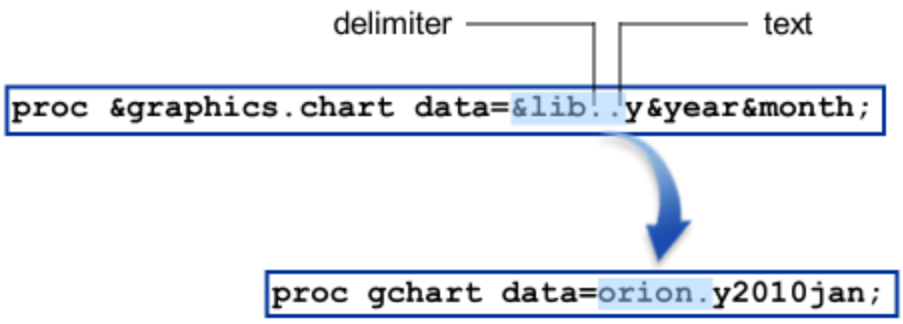




Trailing text can be a problem if a dot is not added after the macro variable.

Corrected Program

Use another period.



112

Trailing Text

Trailing text is a problem if it changes the reference.

It is not a problem here because of the special character.

```
%let year=2010;
%let month=jan;
%let var=sale;
proc chart data=orion.y&year&month;
   hbar week / sumvar=&var;
run;
proc plot data=orion.y&year&month;
   plot &var*day;
run;
```

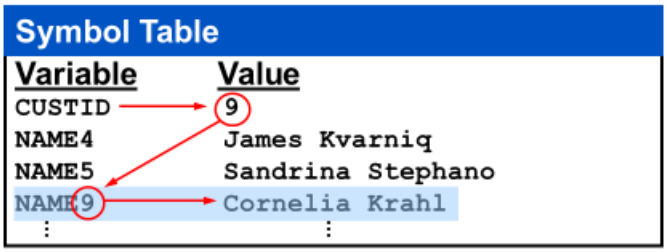
```
proc chart data=orion.y2010jan;
   hbar week / sumvar=sale;
run;
proc plot data=orion.y2010jan;
   plot sale*day;
run;
```

special character

104

Indirect References to Macro Variables

Because the **custID** value matches the numeric suffix of another macro variable, **custID** can indirectly reference the other macro variable.



49

Indirect References to Macro Variables

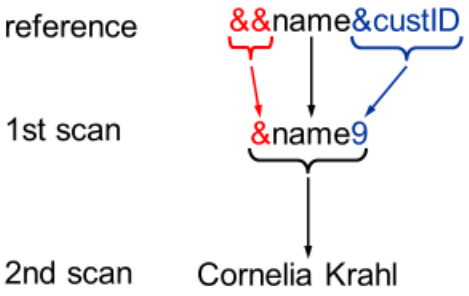
The Forward Rescan Rule

- Multiple ampersands preceding a name token denote an indirect reference.
- Two ampersands (&&) resolve to one ampersand (&).
- The macro processor rescans an indirect reference, left to right, from the point where multiple ampersands begin.
- Scanning continues until no more references can be resolved.

50

Indirect References to Macro Variables

The indirect reference causes a second scan.



52

Indirect References to Macro Variables

The **custID** macro variable indirectly references a **name** macro variable.

Symbol Table

Variable	Value
CUSTID	9
NAME4	James Kvarniq
NAME5	Sandrina Stephano
NAME9	Cornelia Krahl

Scan sequence:

&&name&custID → &name9 → Cornelia Krahl

53

Example: Referencing Macro Variables Indirectly Using a Macro

The double ampersands always resolve into a single ampersand.

```
 1  *Ex7_indirect_reference_1.sas;
 2  %macro ref;
 3  Options symbolgen;
 4  %LET disease1=cvd;
 5  %LET disease2=cancer;
 6  %LET disease3=stroke;
 7  %LET disease4=hbp;
 8  %LET disease5=diabetes;
 9  %LET last_element=5;
10    %DO i = 1 %TO &last_element;
11       %put  &&disease&i;
12     %END;
13  %mend ref;
14  %ref
```

The above code imply macro arrays, which are not explicit arrays like those defined in the DATA step.

Line 4-8: Five macro variables (disease1-disease5) are created.

Line 11: The two ampersands (&&) in front of the macro variable name force the macro processor to scan the macro variable twice.

When the macro %REF is executed, SAS:

- (1st iteration) resolves **&&disease&i** to **&disease1** in the first scan
  &disease1 to **cvd** in the second scan.
- (2nd iteration) resolves **&&diseasei** to **&disease2** in the first scan
- &disease2 to **cancer** in the second scan.
  …
- (5th iteration) resolves **&&disease&i** to **&disease5** in the first scan
- &disease5 **diabetes** in the second scan.

Here is an example of data-driven dynamic programming that uses the iterative %DO loop within a SAS macro to do the following:

- increment an index macro variable whose value ranges from 2005 to 2015
- produce SAS statements for reading raw data from six data files yob2005 through yob2015 that are stored in a zipped file

```
1  *Ex22_Macro_Read_Zipped.sas;
2  Options nocenter nodate nonumber symbolgen;
3  %Let Path = c:\SASCourse\Week9;
4  Libname mylib "&Path";
5  %macro readraw (first=, last=);
6    Filename ZIPFILE SASZIPAM "&Path\names.zip";
7     %do year=&first %to &last;
8       DATA mylib.DSN&Year;
9          INFILE ZIPFILE(yob&year..txt) DLM=',';
10         INPUT name $ gender $ number;
11      RUN;
12      title "Listing from Data Set (Newborns' Names) for &Year";
13         proc print data=mylib.DSN&Year(obs=5) noobs;
14         format number comma7.;
15         run;
16     %end ;
17 %mend readraw;
18 %readraw(first=2000, last=2005)
```

The following program is an extension of the previous program with the addition of lines 18 and 19 below. See page 34 for generated SAS code.

```
 1  *Ex23_Macro_Generates_Code.sas;
 2  Options nocenter nodate nonumber nosymbolgen;
 3  %Let Path = c:\SASCourse\Week9;
 4  Libname mylib "&Path";
 5  %macro readraw (first=, last=);
 6  Filename ZIPFILE SASZIPAM "&Path\names.zip";
 7     %do year=&first %to &last;
 8       DATA mylib.DSN&Year;
 9          INFILE ZIPFILE(yob&year..txt) DLM=',';
10          INPUT name $ gender $ number;
11       RUN;
12       title "Listing from Data Set (Newborns' Names) for &Year";
13          proc print data=mylib.DSN&Year(obs=5) noobs;
14          format number comma7.;
15          run;
16      %end ;
17  %mend readraw;
18  filename mprint "&Path\Ex23_Generated_Code.sas";
19  options mprint mfile;
20  %readraw(first=2000, last=2005)
```

Line 19: The option MPRINT enables you to see the SAS code that the macro processor generates, and the MFILE option enables the code generated in the file specified in the FILENAME statement. By default, SAS does not write to the SAS log SAS statements constructed by the macro processor.

```
Filename ZIPFILE SASZIPAM "c:\SASCourse\Week9\names.zip";
DATA mylib.DSN2000;
INFILE ZIPFILE(yob2000.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
title "Listing from Data Set (Newborns' Names) for 2000";
proc print data=mylib.DSN2000(obs=5) noobs;
format number comma7.;
run;
DATA mylib.DSN2001;
INFILE ZIPFILE(yob2001.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
title "Listing from Data Set (Newborns' Names) for 2001";
proc print data=mylib.DSN2001(obs=5) noobs;
format number comma7.;
run;
DATA mylib.DSN2002;
INFILE ZIPFILE(yob2002.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
title "Listing from Data Set (Newborns' Names) for 2002";
proc print data=mylib.DSN2002(obs=5) noobs;
format number comma7.;
run;
DATA mylib.DSN2003;
INFILE ZIPFILE(yob2003.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
title "Listing from Data Set (Newborns' Names) for 2003";
proc print data=mylib.DSN2003(obs=5) noobs;
format number comma7.;
run;
DATA mylib.DSN2004;
INFILE ZIPFILE(yob2004.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
title "Listing from Data Set (Newborns' Names) for 2004";
proc print data=mylib.DSN2004(obs=5) noobs;
format number comma7.;
run;
DATA mylib.DSN2005;
INFILE ZIPFILE(yob2005.txt) DLM=',';
INPUT name $ gender $ number;
RUN;
```

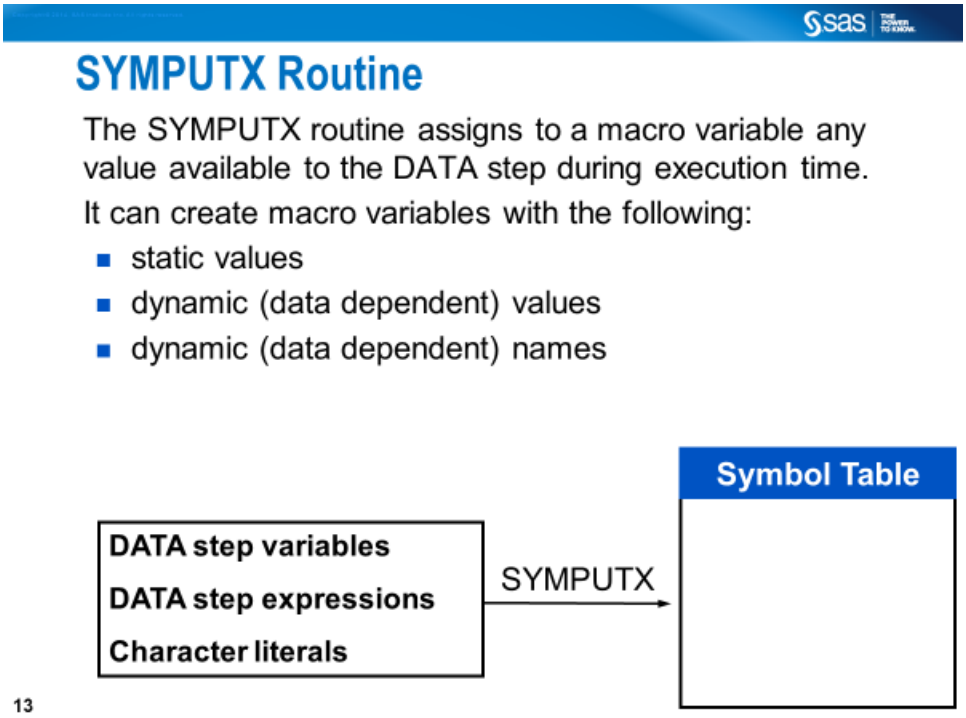
Why do we use macro variables?

One reason for using macro variables is to circumvent the problem that the variable stored in a SAS data set cannot be retrieved in a DATA step without using it as an input data set.

Problem:

```
 1  *Ex1_Motivation_for_macro_variables;
 2  proc means data=sashelp.class mean maxdec=1;
 3   var weight;
 4   output out=stats mean=average_wgt;
 5  run;
 6
 7  data test;
 8   set SASHELP.class;
 9   *This line of code does not work;
10   weight_ratio=weight/average_wgt;
11   run;
```

The variable average_wgt stored in data set STATS cannot be used in the DATA step (lines 7-11) above.



Global Symbol Table

Global macro variables can be created by any of the following:

- %LET statement
- DATA step SYMPUTX routine
- PROC SQL SELECT statement INTO clause
- %GLOBAL statement

73

%GLOBAL Statement

General form of the %GLOBAL statement:

%GLOBAL *macro-variable1 macro-variable2 . . .* **;**

- The %GLOBAL statement adds one or more macro variables to the global symbol table with null values.
- It has no effect on variables already in the global table.
- It can be used anywhere in a SAS program.

74

Local Symbol Table

Local macro variables can be

- created at macro invocation (parameters)
- created during macro execution
- updated during macro execution
- referenced anywhere within the macro.

76

Local Symbol Table

A *local symbol table* is

- created when a macro with a parameter list is called or a local macro variable is created during macro execution
- deleted when the macro finishes execution.

Macros that do not create local variables do not have a local table.

75

%LOCAL Statement

Declare the index variable of a macro loop as a local variable to prevent accidental contamination of a like-named macro variable in the global table or another local table.

```
%macro putloop;
   %local i;
   %do i=1 %to &numrows;
      %put Country&i is &&country&i;
   %end;
%mend putloop;
```

80 m105d13

Local Symbol Table

Local macro variables can be created by the following within a macro definition:

- %LET statement
- DATA step SYMPUTX routine
- PROC SQL SELECT statement INTO clause
- %LOCAL statement

78

Macro variables are global when they are defined outside of a macro; and local to a macro when they are defined inside of a macro when the %GLOBAL statement is not used.

CALL SYMPUTX vs. PROC SQL INTO:

The following code is obtained from Simon (2017).

```
1  * Ex29_symputx_sql_into.sas (Part 1);
2  options center nodate nonumber;
3  * Adapted from Simon (2017);
4  proc means data=sashelp.class noprint;
5   var weight;
6   output out=mystats mean=avweight;
7   run;
8
9  data _null_;
10  set mystats;
11  call symputx('meanweight',avweight);
12  run;
13
14 proc print data=sashelp.class noobs;
15 var name sex weight;
16  where weight > &meanweight;
17 title 'Students weighing more than average weight';
18 title2 "Average weight:(%sysfunc(round(&meanweight, 0.01)) lbs)";
19 title3 '3 DATA/PROC Steps';
20 run;
21 title;
```

```
Students weighing more than average weigh
       Average weight:(100.03 lbs)
            3 DATA/PROC Steps

          Name       Sex    Weight

          Alfred      M      112.5
          Carol       F      102.5
          Henry       M      102.5
          Janet       F      112.5
          Mary        F      112.0
          Philip      M      150.0
          Robert      M      128.0
          Ronald      M      133.0
          William     M      112.0
```

```
23 * Ex29_symputx_sql_into.sas (Part 2);
24 title 'Students weighing more than average weight';
25 title2 'PROC SQL - Just 1 Step';
26 proc sql ;
27  select mean(weight) into: meanweight_x
28         FROM sashelp.class ;
29  select name, sex, weight format=6.1
30         FROM sashelp.class
31  having weight > &meanweight_x;
32 quit;
```

Students weighing more than average weight
PROC SQL - Just 1 Step

Name	Sex	Weight
Alfred	M	112.5
Carol	F	102.5
Henry	M	102.5
Janet	F	112.5
Mary	F	112.0
Philip	M	150.0
Robert	M	128.0
Ronald	M	133.0
William	M	112.0

What Is a Macro Definition?

Like macro variables, macros generate text. However, macros can contain programming logic to dynamically control what text is generated and when it is generated. Macros can also accept parameters.

A macro or macro definition can store the following:

- macro language statements or expressions
- complete or partial SAS statements
- complete or partial SAS steps
- any text
- any combination of the above

37

Defining a Macro

This code defines the **Time** macro, which displays the current time.

```
%macro time;
   %put The current time is %sysfunc
        (time(),timeampm.).;
%mend time;
```

%MACRO *macro-name*;
macro-text
%MEND <*macro-name*>;

Macro names follow SAS naming conventions and cannot be reserved names such as names of macro statements or functions (for example, LET and SCAN).

psm08d06

38

Macro Compilation

To verify macro compilation, specify the MCOMPILENOTE=ALL option.

OPTIONS MCOMPILENOTE=ALL|NONE;

```
options mcompilenote=all;
%macro time;
   %put The current time is %sysfunc
        (time(),timeampm.).;
%mend time;
```

SAS Log

```
1  options mcompilenote=all;
2  %macro time;
3     %put The current time is %sysfunc
4          (time(),timeampm.).;
5  %mend time;
NOTE: The macro TIME completed compilation without errors.
      3 instructions 76 bytes.
```

- The default value is MCOMPILENOTE=NONE.

8 m103d01

Calling a Macro

To call a macro, precede the macro name with a percent sign (%).

```
%time
```

%macro-name

SAS Log

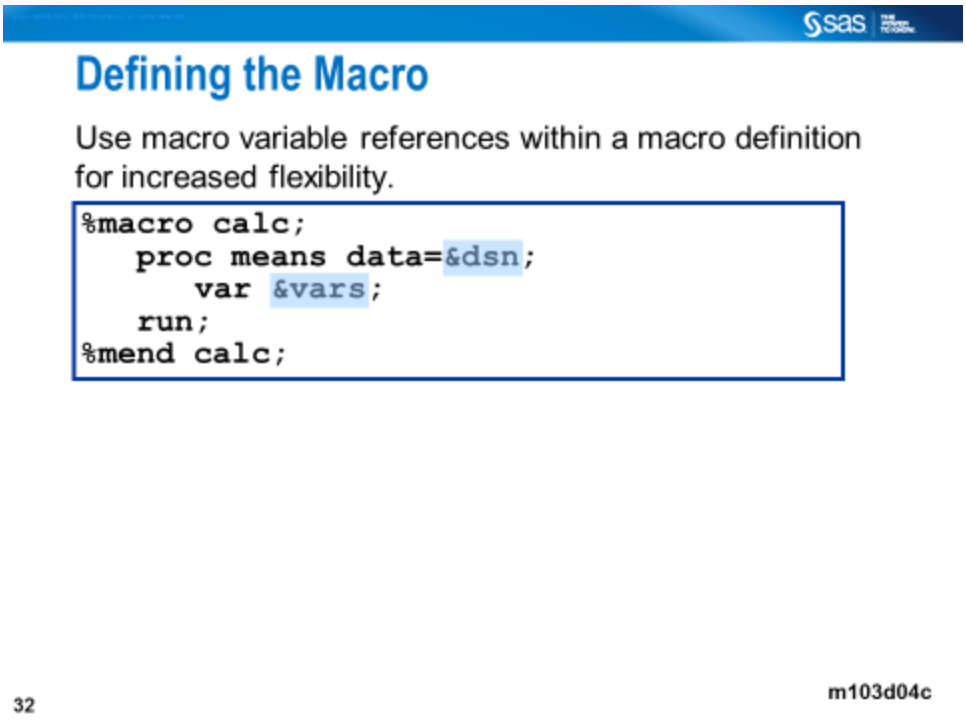
```
178 %time
The current time is 2:49:39 PM.
```

A *macro call*

- can appear anywhere (similar to a macro variable reference)
- represents a macro trigger
- is passed to the macro processor
- is ***not*** a statement (no semicolon required)
- causes the macro to execute.

9 m103d01

Macro Compilation Process	Macro Execution Process
• The user submits the macro definition (%MACRO to %END sandwich). • The macro facility - intercepts the macro definition - performs a macro compilation • The compiled macro definition gets stored in WORK.SAMACR	• SAS checks for the existence of the macro in the catalog WORK.SASMACR. • If the macro exists, it gets executed. • The macro call is replaced by any text generated by the macro • Generated code is compiled and executed by Base SAS.



Calling the Macro

Calling this macro involves two steps.

Step 1 Precede the call with %LET statements to populate macro variables referenced within the macro.

```
%let dsn=orion.order_fact;
%let vars=quantity;
```

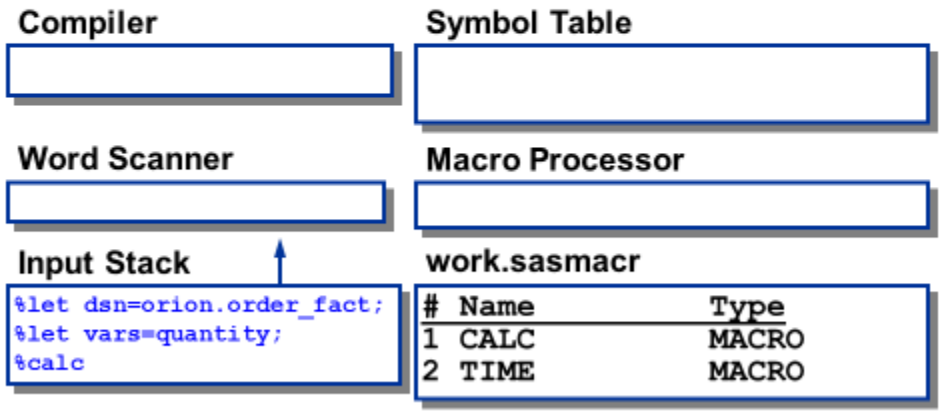
Step 2 Precede the macro name with a percent sign (%) to call the macro.

```
%let dsn=orion.order_fact;
%let vars=quantity;
%calc
```

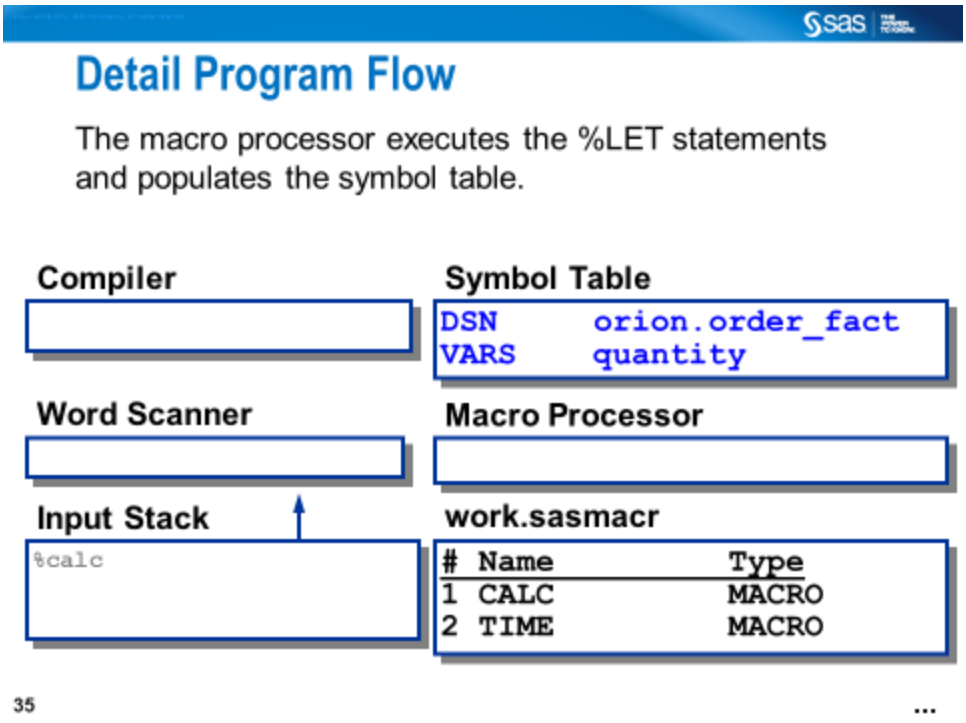
33 m103d04c

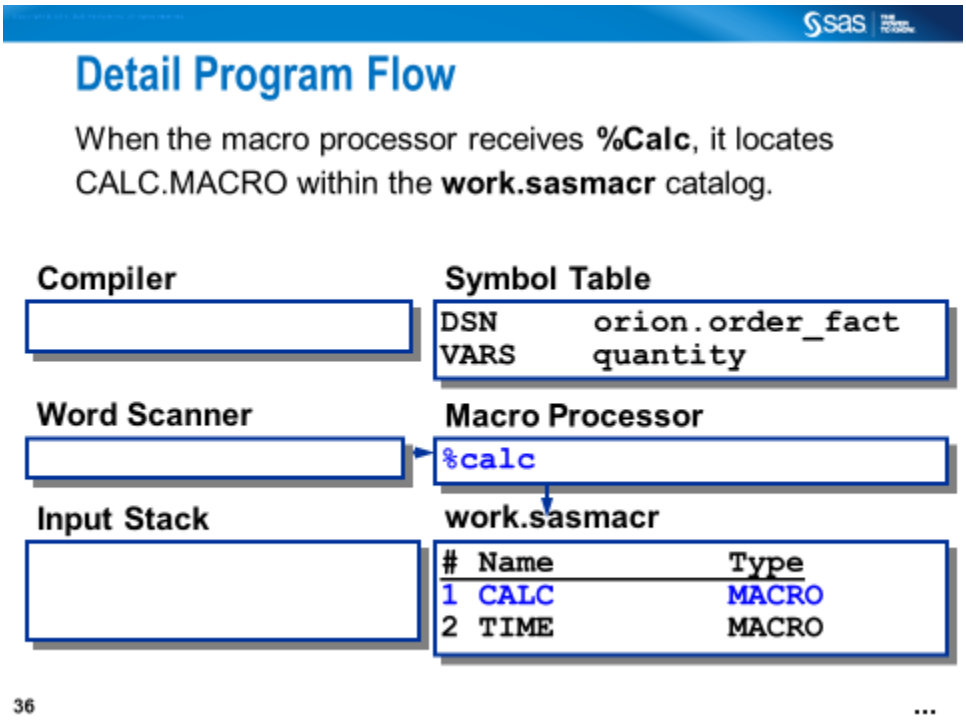
Detail Program Flow

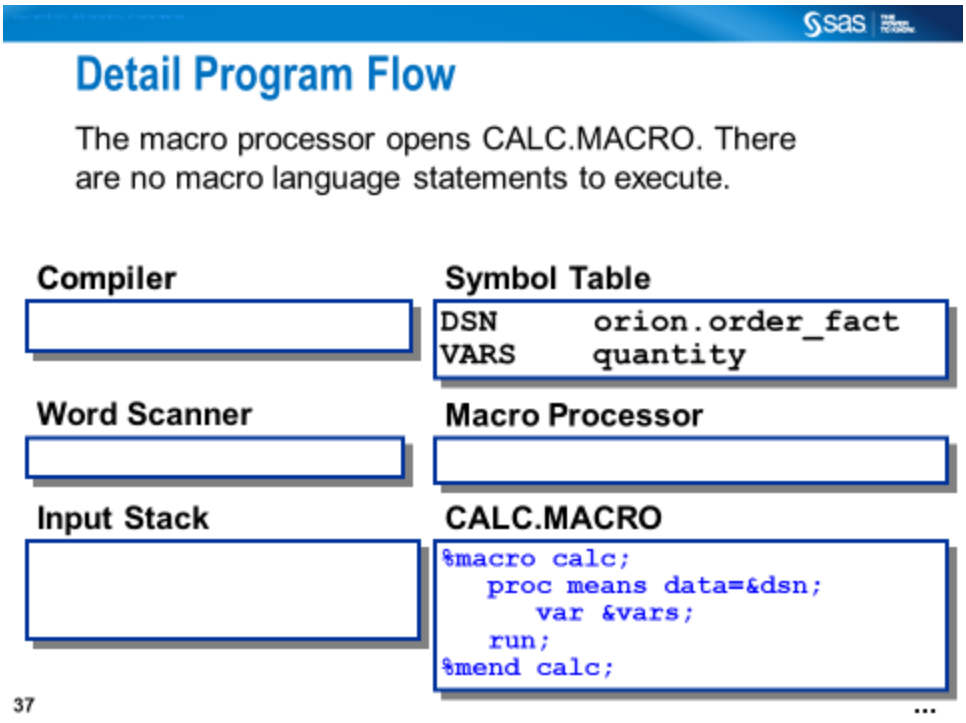
Example: Create macro variables and call the **Calc** macro.

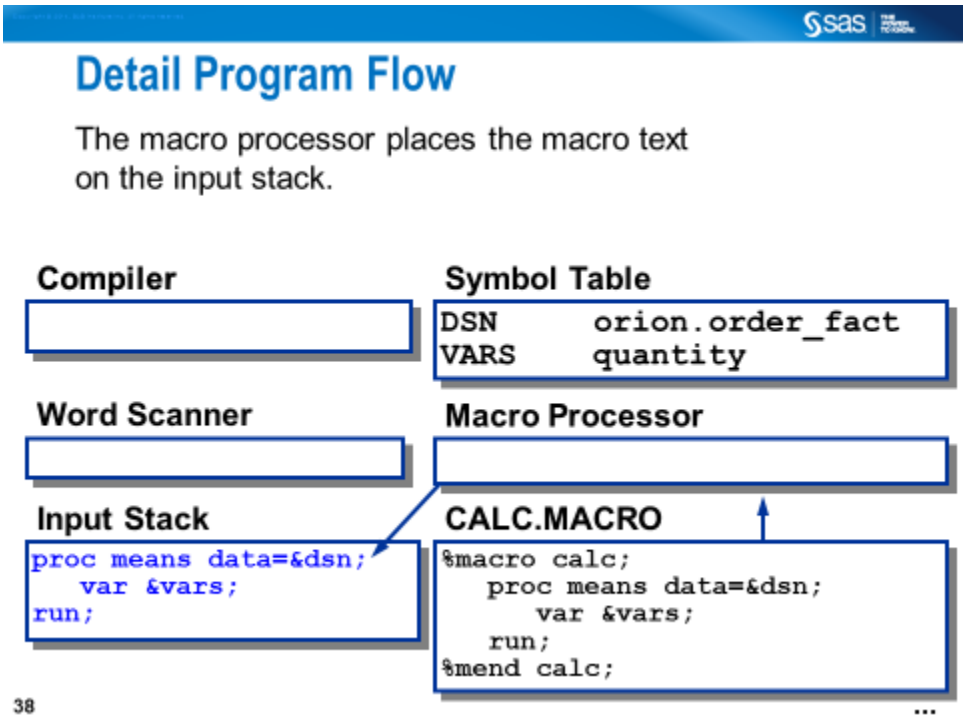


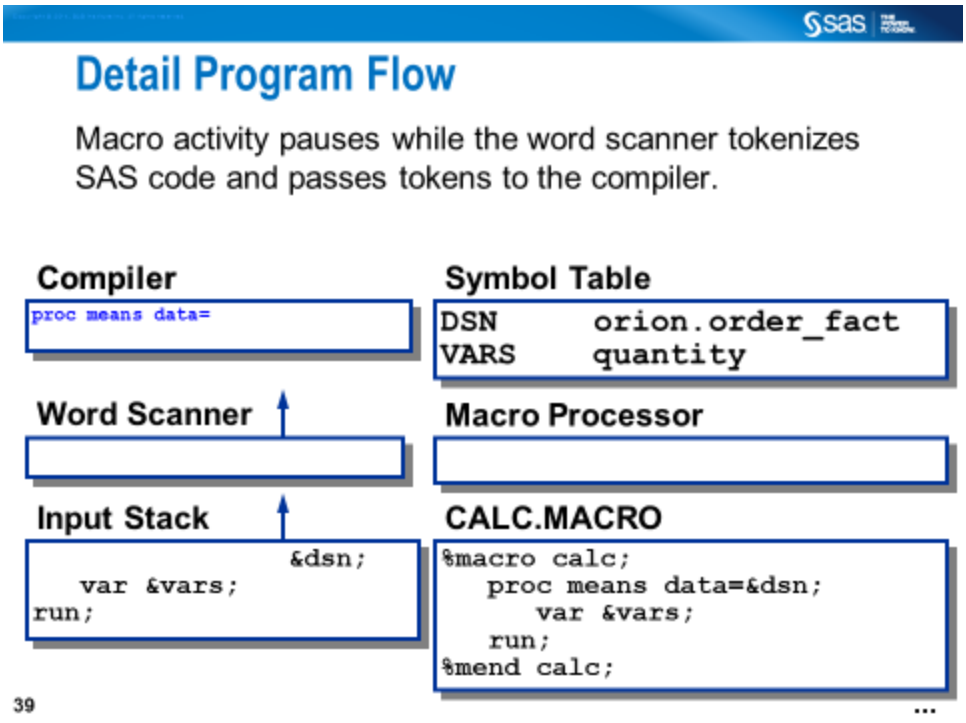
34 ...

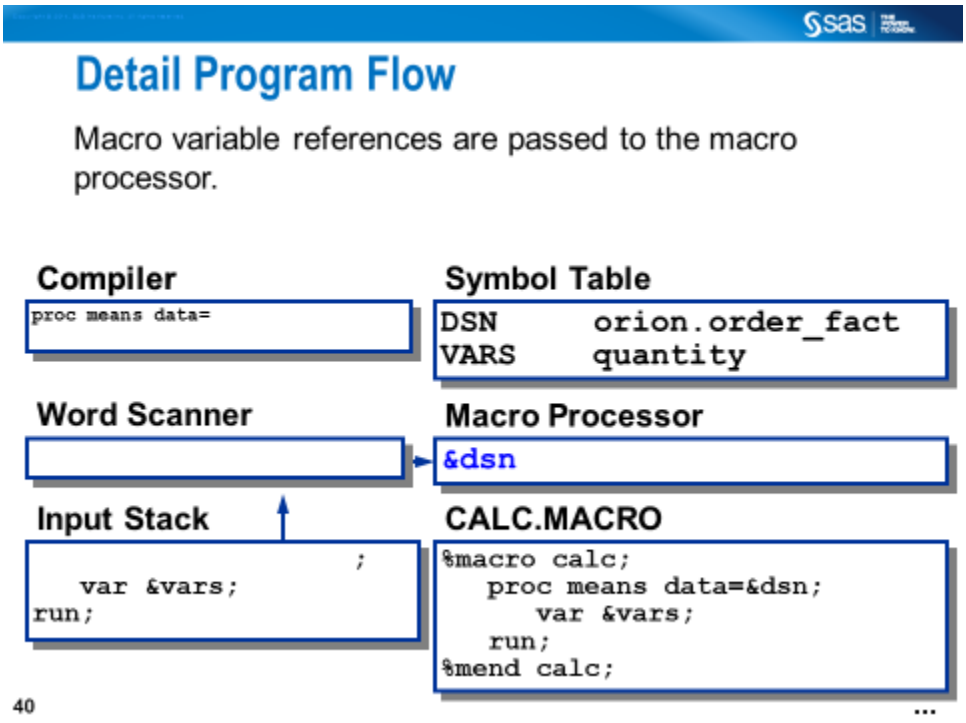


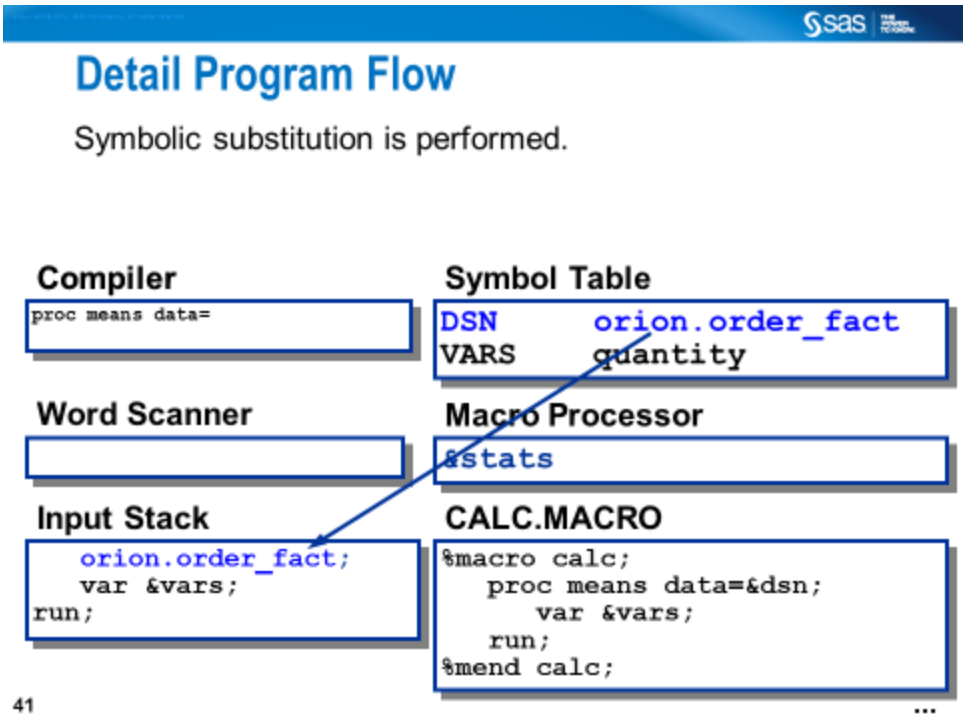


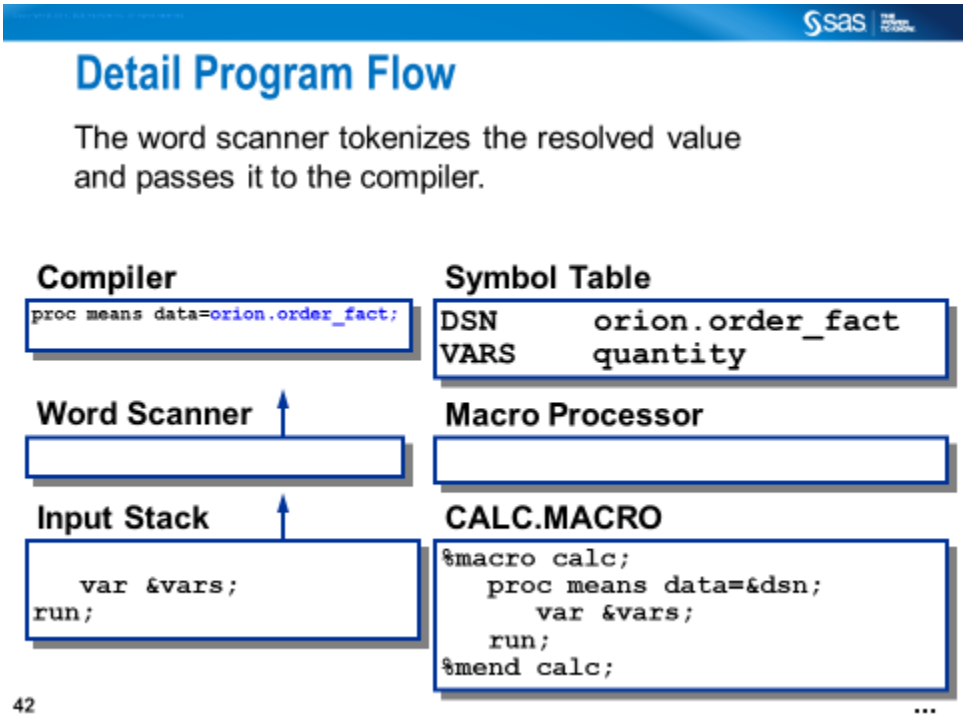






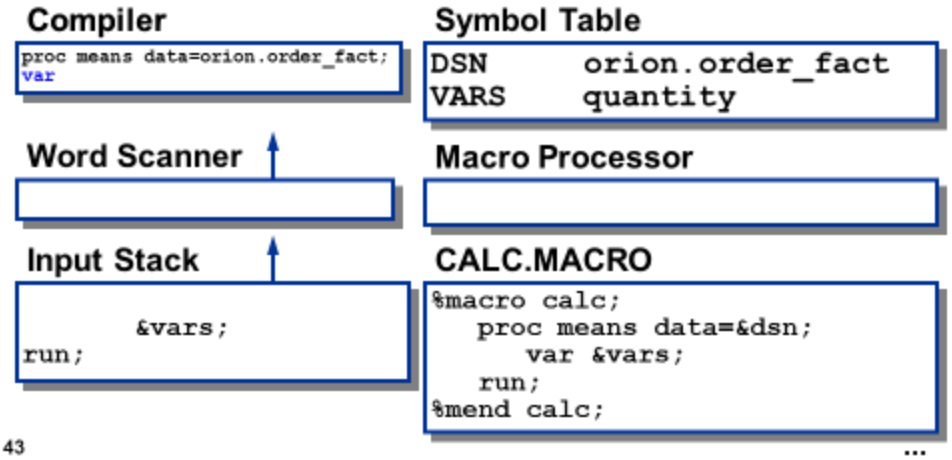






Detail Program Flow

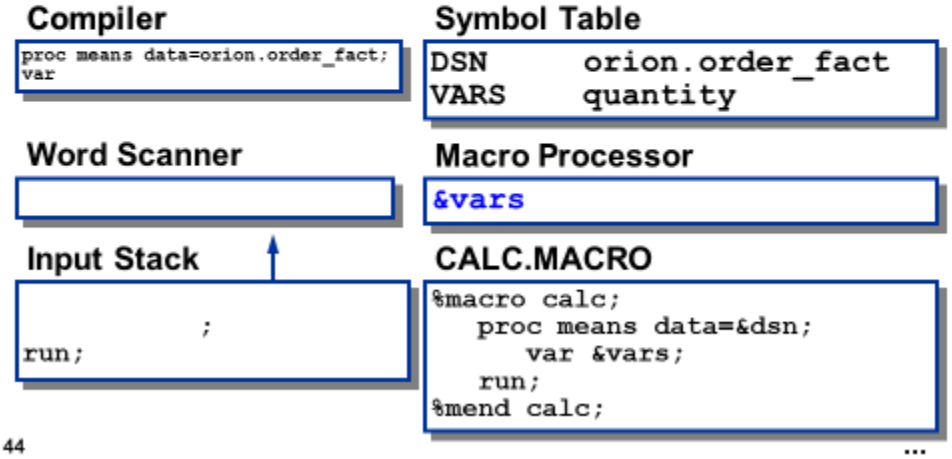
Word scanning continues.



43 ...

Detail Program Flow

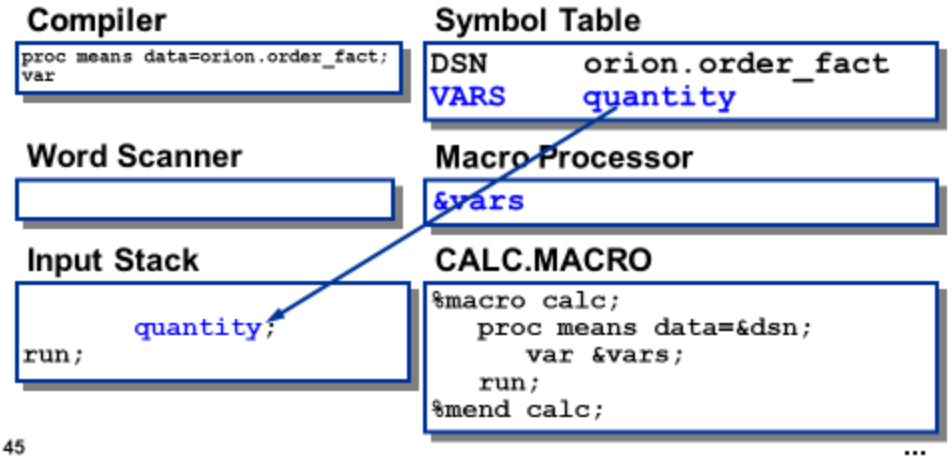
The macro variable reference is passed to the macro processor.



44 ...

Detail Program Flow

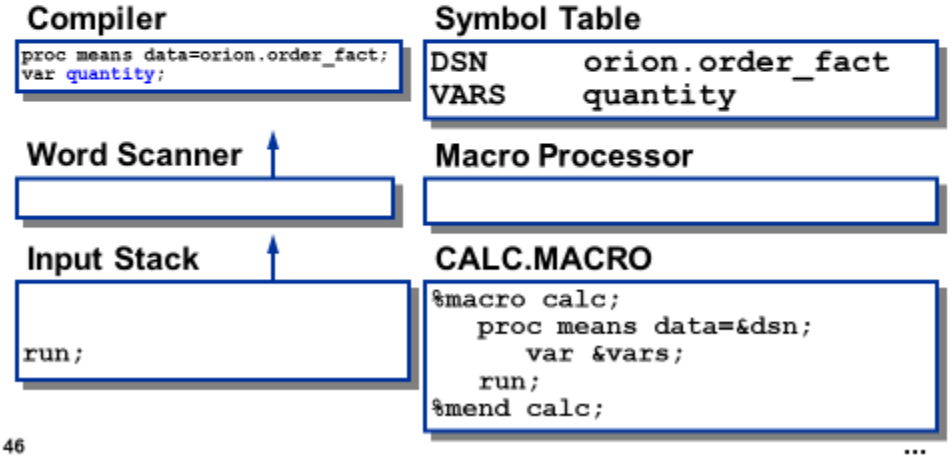
Symbolic substitution is performed.



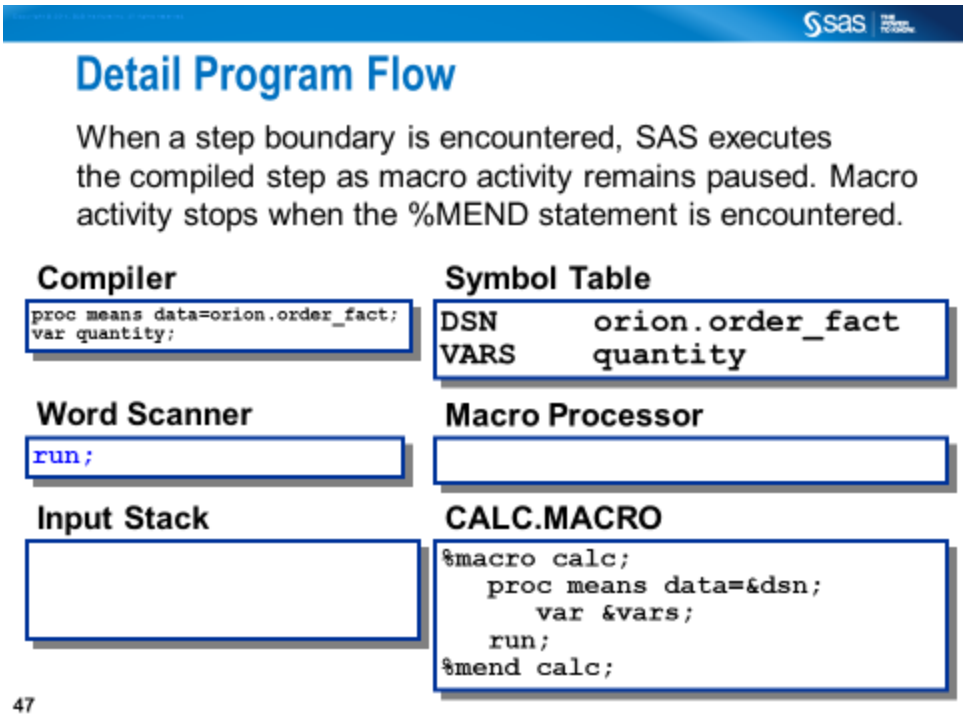
45

Detail Program Flow

The word scanner tokenizes the resolved value and passes it to the compiler.



46



Ways to Store/Access Macros

- %INCLUDE of source entries non-compiled (macro definitions)
- Stored Compiled macros (Compiled macro definitions)
- AUTOCALL libraries that include macros

%INCLUDE Statement

- retrieves macro definitions from an external file and places it on the input stack
- is a global SAS statement, not a macro language statement

Consider the following macro definition saved in a file.

```
1  *Macro_definitions.sas;
2  Options nocenter nodate nonumber symbolgen;
3  %Let Path = c:\SASCourse\Week9;
4  Libname mylib "&Path";
5  %macro readraw (first=, last=);
6      Filename ZIPFILE SASZIPAM "&Path\names.zip";
7      %do year=&first %to &last;
8          DATA mylib.DSN&Year;
9            INFILE ZIPFILE(yob&year..txt) DLM=',';
10           INPUT name $ gender $ number;
11         RUN;
12         title " Listing from Data Set (Baby Names)";
13         proc print data=mylib.DSN&Year(obs=5) noobs split='*';
14         label name = "Newborn's*Name" number ='Name Count';
15         format number comma7.;
16         run;
17         title;
18     %end ;
19 %mend readraw;
```

```
1  *Ex17_percent_include_Macro.sas;
2  %INCLUDE 'C:/SASCourse/Week9/Macro_definitions.sas'/Source2;
3  %readraw(first=1998, last=2003)
```

Line 2: The quoted (Windows) filename includes the macro definitions. The SOURCE2 option displays inserted SAS statements in the SAS log. Note that the % include is a global statement, not a macro statement.

Below is an example of use of the %INCLUDE statement, each containing a DATA or PROC step.

```
1  *Ex17_percent_include_x.sas;
2  %include "c:\sascourse\Week9\Part1.sas"/source2;
3  %include "c:\sascourse\Week9\Part2.sas"/source2;
4  %include "c:\sascourse\Week9\Part3.sas"/source2;
```

Alternatively, you can use the SAS filename statement to associate fileref with an external file as shown below.

```
1  *Ex17_percent_include.sas;
2  filename jobs "c:\sascourse\Week9";
3  %include jobs(Part1)/source2;
4  %include jobs(Part2)/source2;
5  %include jobs(Part3)/source2;
```

Lines 3-5: %INCLUDE brings in SAS statements stored in external files (part1, part2, and part3). The SOURCE2 option displays inserted SAS statements in the SAS log. Note that the % include is not a macro statement.

Stored Compiled Macros

The stored compiled macro facility

- stores macros permanently in a designated library
- saves macro compilation time
- is enabled with the MSTORED and SASMSTORE= system options.

```
libname orion "S:\workshop";
options mstored sasmstore=orion;
```

OPTIONS MSTORED SASMSTORE=*libref*;

11 m202d02

Creating a Stored Compiled Macro

Store a permanent compiled macro using options in the %MACRO statement.

```
%macro calc(stats,vars) / store source;
   proc means data=orion.order_fact &stats;
      var &vars;
   ru
%mend
```

%MACRO *macro-name (parameters) /*
<STORE> <SOURCE> <SECURE> <DES='*text*'**>;**

STORE	Stores the compiled macro as a catalog entry in the library indicated by the SASMSTORE= system option.
SOURCE	Stores the macro source code with the compiled code.
SECURE	Encrypts the stored compiled macro and disables the %COPY statement and the SYMBOLGEN, MLOGIC, and MPRINT options.
DES='*text*'	Specifies a description of the macro catalog entry.

12 m202d02

Clearing the SASMSTORE Libref

The stored compiled macro facility places a lock on the SASMSTORE= libref during a session.

To close the stored compiled macro catalog and unlock the libref, issue the following statement:

```
%sysmstoreclear;
```

%SYSMSTORECLEAR was introduced in SAS 9.3.

13

Accessing a Stored Compiled Macro

To access a stored compiled macro in a new SAS session, do the following:

1. Assign a libref to the location of the compiled macro.
2. Activate the stored compiled macro facility.
3. Call the stored macro.

```
libname orion "S:\workshop";
options mstored sasmstore=orion;
%calc(min max,quantity)
```

m202d02

14

Storing Macros Using the Stored Compiled Macro Facility

Macros can be compiled and then stored temporarily or permanently for later use. When you compile a macro, it is usually stored in the temporary WORK.SASMACR catalog. However, you can store a macro in compiled form in a SAS macro catalog also named SASMACR, but in a permanent library. The following code is adapted from Simon (2017).

```
1  *Ex24_Compiled_Mstored.sas;
2  options nodate nonumber symbolgen;
3  %LET Path= C:\SASCourse\Week9;
4  Libname New "&Path";
5  options mstored sasmstore=New;
6  %macro format(value,format)/store source
7           des='Macro that formats macro variables';
8      %if %datatyp(&value)=CHAR
9           %then %sysfunc(putc(&value,&format));
10     %else %left(%qsysfunc(putn(&value,&format)));
11 %mend format;
12 %sysmstoreclear;
```

Line 5: In the OPTIONS statement, the system option MSTORED turns on the use of stored compiled macros. The system option SASMSTORE= specifies the libref that contains a SAS catalog named SASMACR.

Lines 6-11: This is the macro definition. Note the STORE option (required), the SOURCE option, and the DES option on the %MACRO statement. The NEW.SASMACR catalog is opened when the entire macro written to it.

Line 12: The %SYSMSTORECLEAR statement closes the SASMACR catalog, preventing the libref from being reassigned.

Displaying the Information from the Catalog

The CATALOG procedure can be used to display the relevant information, including the description specified when the macro was compiled and stored. Below is an example.

```
1  *Ex25_Macro_catalog.sas;
2  %LET Path = C:\SASCourse\Week9;
3  LIBNAME NEW "&Path";
4  proc catalog catalog=NEW.sasmacr;
5  contents;
6  run;
```

Contents of Catalog NEW.SASMACR

#	Name	Type	Create Date	Modified Date	Description
1	FORMAT	MACRO	10/23/2019 06:05:13	10/23/2019 06:05:13	Macro that formats macro variables

Calling a Compiled-Stored Macro

```
 1  *Ex26_Access_Compiled_Macro
 2  libname Mylib  'C:\SASCourse\Week9';
 3  options mstored sasmstore=Mylib;
 4  proc format;
 5  value $s_group
 6        'E' = 'Experimental Group'
 7        'C' = 'Comparison Group';
 8  value score
 9         0-9 = 'Low'
10         10-99 = 'Medium'
11         100-999='High';
12  run;
13  %put %format(E, $s_Group.);
14  %put %format(50, score.);
```

Line 3: In the OPTIONS statement, the system option MSTORED turns on the use of stored compiled macros. The system option SASMSTORE= specifies the libref that contains a SAS catalog named SASMACR.

Lines 5-6: %format is macro call. The **format** macro was earlier compiled and stored permanently in the folder specified in line 2.

```
41   *Ex26_Access_Compiled_Macro
42   libname Mylib  'C:\SASCourse\Week9';
43   options mstored sasmstore=Mylib;
44   proc format;
45   value $s_group
46         'E' = 'Experimental Group'
47         'C' = 'Comparison Group';
NOTE: Format $S_GROUP is already on the library WORK.FORMATS.
NOTE: Format $S_GROUP has been output.
48   value score
49         0-9 = 'Low'
50         10-99 = 'Medium'
51         100-999='High';
NOTE: Format SCORE is already on the library WORK.FORMATS.
NOTE: Format SCORE has been output.
52   run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


53   %put %format(E, $s_Group.);
Experimental Group
54   %put %format(50, score.);
Medium
```

Making the Autocall Macro Available

The following macro definition uses %SYSFUNC to call the DATE function. When invoked, the macro call retrieves the current date (a numeric value) and then displays it as a formatted character string (Carpenter, 2016).

For an autocall macro, it is required that the macro name and the file name that includes macro definition are the same. The example code below meets the requirement.

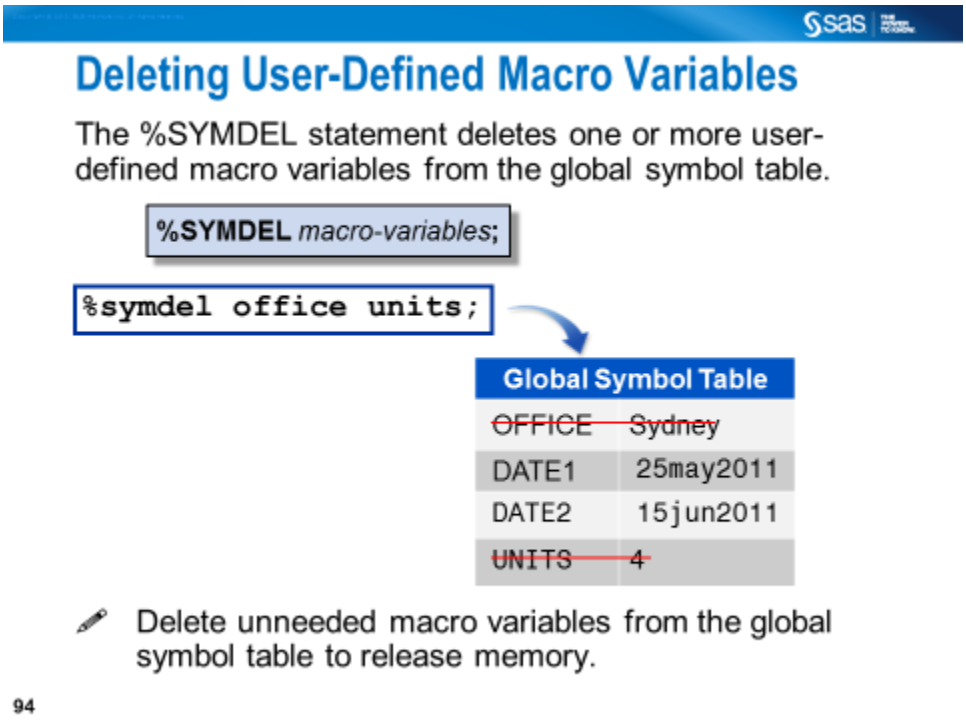
```
1  *Date_macro.sas;
2  *Saved in C:\SASCourse\Week9;
3  %macro date_macro;
4    %sysfunc(left(%qsysfunc(date(),worddate18.)));
5  %mend date_macro;
```

With the MATOSOURCE and SASAUTOS= options (Line 3 below), you have automatically access to the macro. Notice that the macro call (%date_macro) is in double quotes in the TITLE statement.

```
1  *Ex28_Autocall.sas;
2  filename mymacros 'C:\sascourse\Week9';
3  options nodate nonumber mautosource sasautos=(mymacros, sasautos);
4  title "Means for height and weight - Job run on %date_macro";
5  proc print data=sashelp.class (obs=5) noobs;
6  run;
```

Means for height and weight - Job run on October 24, 2019

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5



The macro statements can delete user-defined macro variables.

The SYMDEL routine can also be used to remove user-defined macro variables from the global symbol table.

Review Questions

1. Describe the ways one could create macro variables.
2. What is the purpose of creating macro variables in SAS?
3. Which points need to be considered when creating macro variables using a %LET statement?
4. What is the maximum length of a macro variable?
5. What is an automatic macro variable?
6. Name some common automatic macro variables and state their usage.
7. Where are the macro variables stored?
8. Where can you reference a macro variables?
9. What rule has to be followed when referencing a macro variable in the TITLE or FOOTNOTE statement?
10. Describe the purpose of using the %PUT statement?
11. What is the difference between the %PUT statement and the SYMBOLGEN option?
12. What macro functions would enable you to do arithmetic on the macro variables whose values are numeric?
13. Describe when you use the CALL SYMPUTX routine to create macro variables?
14. How do you create a single macro variable which can hold all the distinct values of DATA step variable?
15. How do you create a series of macro variables, each containing a distinct value of a DATA step variable?
16. How would you define a macro and then call it for execution?
17. What is the purpose of using a macro?
18. What is the difference between %LOCAL and %GLOBAL statements?
19. When would you indirectly reference macro variables?
20. How do you convert macro variables to DATA step variables?
21. Show code examples how to:
    - Add text before a macro variable reference
    - Add text after a macro variable reference
    - Add two macro variable references one after another
    - Reference a permanent SAS data set (a two-level SAS name) where the *libref* part is in the form of a macro variable reference
22. Explain the global symbol table.
23. How would you delete macro variables from a global symbol table?
24. How is a local symbol table created?