SAS® Macro Language 1: Essentials

Course Notes

SAS® Macro Language 1: Essentials Course Notes was developed by Jim Simon. Additional contributions were made by Linda Mitterling and Theresa Stemler. Editing and production support was provided by the Curriculum Development and Support Department.

SAS® Macro Language 1: Essentials Course Notes

Book code E2804, course code LWMC1/MC1, prepared date 3/17/2015. LWMC1_001

ISBN 978-1-62959-731-7

Table of Contents

Course Description

This course focuses on the components of the SAS macro facility and how to design, write, and debug macro systems. Emphasis is placed on understanding how programs with macro code are processed.

To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the Web at support.sas.com/training/ as well as in the Training Course Catalog.

For a list of other SAS books that relate to the topics covered in this Course Notes, USA customers can contact our SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the Publications Catalog on the Web at support.sas.com/pubs for a complete list of books and a convenient order form.

Prerequisites

Before attending this course, you should have completed the SAS® Programming 2: Data Manipulation Techniques course or have equivalent knowledge. Specifically, you should be able to

- use a DATA step to read from or write to a SAS data set or external file
- use DATA step programming statements such as IF-THEN/ELSE, DO WHILE, DO UNTIL, and iterative DO
- use SAS data set options such as DROP=, KEEP=, and OBS=
- use character functions such as SUBSTR, SCAN, INDEX, and UPCASE
- form subsets of data using the WHERE clause
- create and use SAS date values and constants
- use SAS procedures such as SORT, PRINT, CONTENTS, MEANS, FREQ, TABULATE, and CHART.

Chapter 1 Introduction

1.1 Course Logistics

Objectives

- Describe the data that is used in the course.
- Identify the interfaces that are available for workshops.
- Specify the naming convention that is used for course files.
- Define the three levels of exercises.
- Explain the extended learning for this course.
- Execute a SAS program to create course data files.
- Execute a SAS program to define the data location.

3

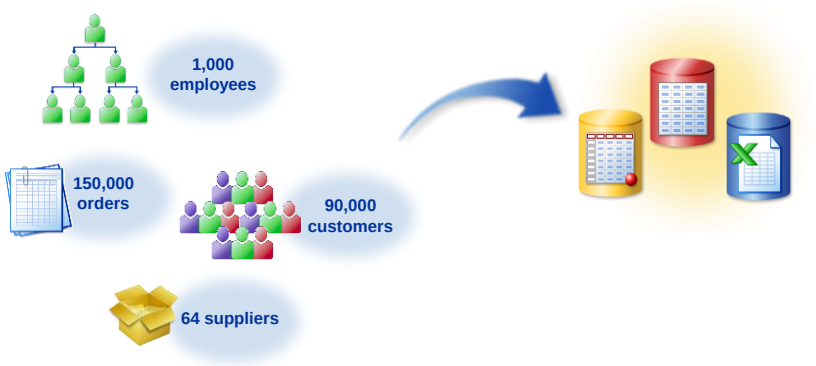
Orion Star Sports & Outdoors

This course focuses on a fictitious global sports and outdoors retailer that has traditional stores, an online store, and a catalog business.

4

Orion Star Data

Large amounts of data are stored in various formats.



5

SAS Programming Interfaces

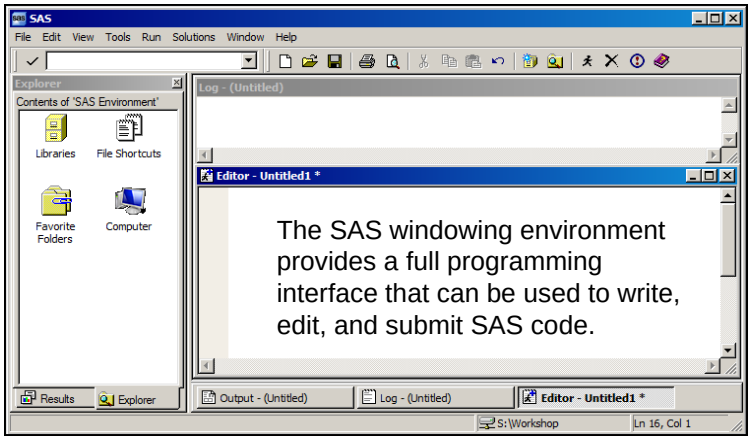
In this course, three interfaces are available for you to use during workshops.

- To enable you to learn more about each of these interfaces, self-directed exercises are available in the course data location.

6

SAS Windowing Environment

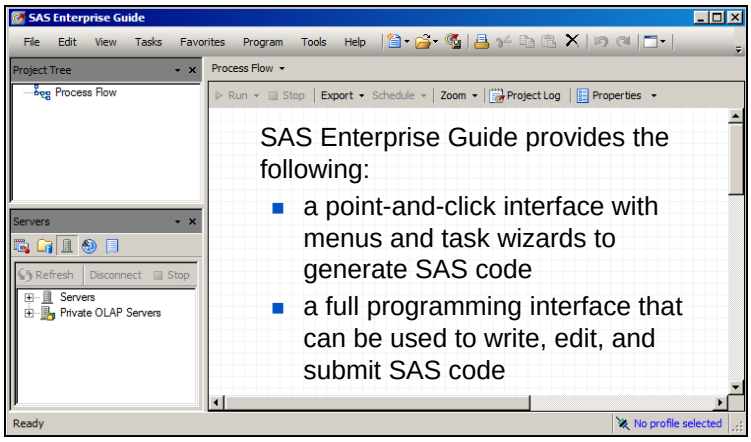
The *SAS windowing environment* is an application that is accessed from different operating environments.



7

SAS Enterprise Guide

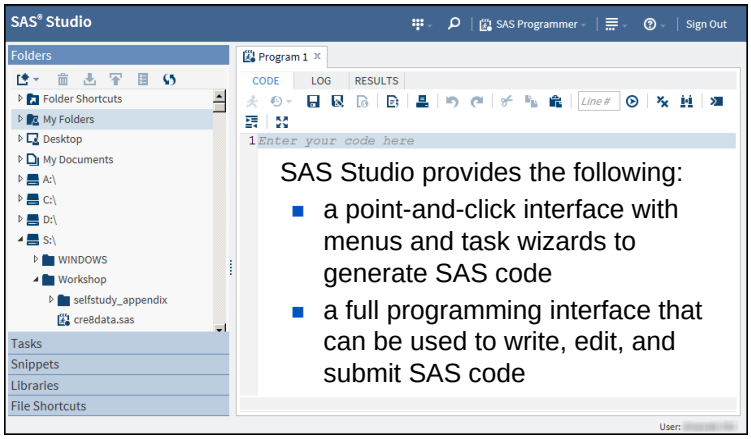
SAS Enterprise Guide is a client application that is accessed from the Windows operating environment.



8

SAS Studio

SAS Studio is a web client that is accessed through an HTML5-compliant web browser.



SAS Studio provides the following:

- a point-and-click interface with menus and task wizards to generate SAS code
- a full programming interface that can be used to write, edit, and submit SAS code

9

1.01 Multiple Choice Poll

Which interface will you use during this course?

a. SAS Studio
b. SAS Enterprise Guide
c. SAS windowing environment

10

Three Levels of Exercises

The course is designed to have you complete only ***one*** set of exercises. Select the level that is most appropriate for your skill set.

Level 1	Provides step-by-step instructions.
Level 2	Provides less information and guidance.
Challenge	Provides minimal information and guidance. You might need to use SAS Help and documentation.

12

Extending Your Learning

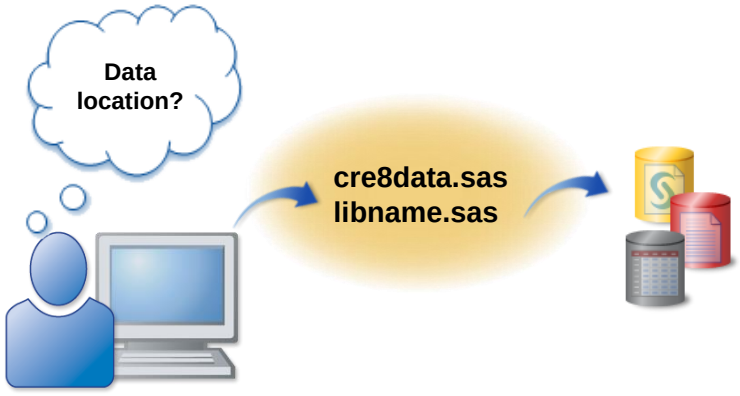
After class, you will have access to an extended learning page that was created for this course. The page includes

- course data and program files
- a PDF file of the course notes
- other course-specific resources.

13

Course Data Creation

You must execute programs to create the files and define the location of the course data.



14

Locating Data Files

In this course, macro variable references are used to give a more flexible approach for locating files.

```
%let path=s:\workshop;

infile "&path\sales.csv";

infile "&path\payroll.dat";
```

15

Creating Course Data Files

This demonstration illustrates how to create the course data files and define the data location.

1. In your SAS session, open the **cre8data** program.
2. Find the following %LET statement:

```
%let path=s:/workshop;
```

3. If your data files are to be created at a location other than **s:\workshop**, change the value that is assigned to the **path** micro variable to reflect the location. If your data files are created in **s:\workshop**, then no change is needed.

   The **cre8data** program uses forward slashes for portability across operating environments. UNIX and Linux require forward slashes. Windows accepts forward slashes and might convert them to back slashes.

4. Submit the program.
5. View the results and verify that the output contains a list of data files.
6. Open the **libname** program, which was created by the **cre8data** program.
7. Submit the program.
8. View the log and verify that there are no errors or warnings.

Exercises

Required Exercise

You ***must*** complete the exercise to create the course data files. If you do not create the data files, all programs in this course will fail.

1. Creating Course Data Files

a. In your SAS session, open the **cre8data** program.

b. Find the following %LET statement:

```
%let path=s:/workshop;
```

c. If your data files are to be created at a location other than **s:\workshop**, change the value that is assigned to the **path** micro variable to reflect the location. If your data files are created in **s:\workshop**, then no change is needed.

The **cre8data** program uses forward slashes for portability across operating environments. UNIX and Linux require forward slashes. Windows accepts forward slashes and might convert them to back slashes.

d. Submit the program.

e. View the results and verify that the output contains a list of data files.

Directory	
Libref	ORION
Engine	V9
Physical Name	s:\workshop
Filename	s:\workshop

#	Name	Member Type	Obs, Entries or Indexes	Vars	Label	File Size	Last Modified
1	CITY	DATA	31	3		196608	02/02/2015 09:47:03
	CITY	INDEX	1			9216	02/02/2015 09:47:03
2	CONTINENT	DATA	5	2		196608	02/02/2015 09:47:03
	CONTINENT	INDEX	1			9216	02/02/2015 09:47:03
3	COUNTRY	DATA	7	6		196608	02/02/2015 09:47:03
	COUNTRY	INDEX	3			17408	02/02/2015 09:47:03

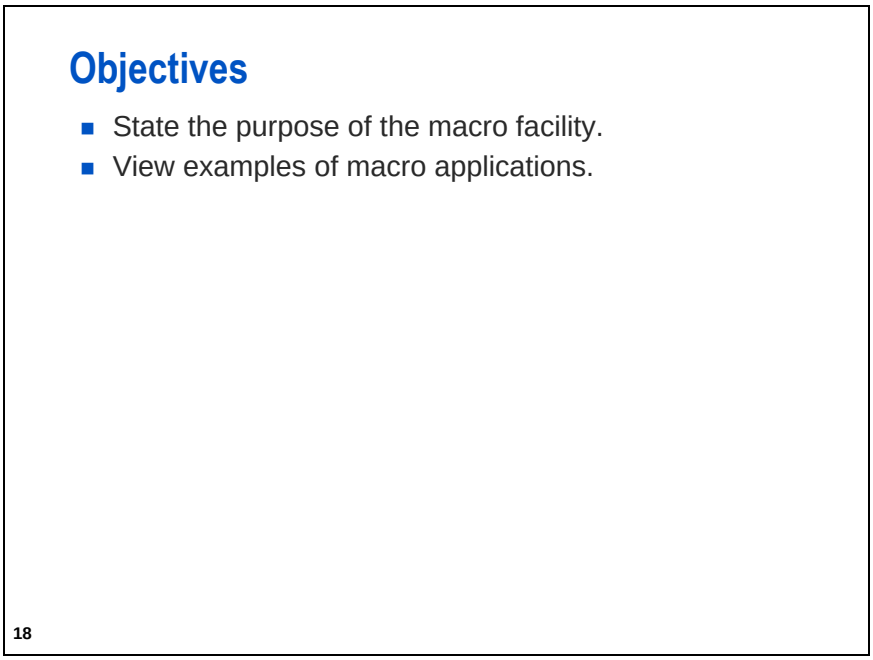
f. Open the **libname** program, which was created by the **cre8data** program.

```
%let path=s:/workshop;
libname orion "s:/workshop";
```

g. Submit the program.

h. View the log and verify that there are no errors or warnings.

1.2 Purpose of the Macro Facility



Purpose of the Macro Facility

The *macro facility* is a ***text processing facility*** for automating and customizing the generation of SAS code, minimizing the amount of code that you must enter.

The macro facility supports the following:

- symbolic substitution within SAS code
- automated production of SAS code
- conditional construction of SAS code
- dynamic generation of SAS code

19

Substituting System Values

Automatic macro variables minimize entering of system values.

```
proc print data=orion.customer;
title "Customer List";
footnote1 "Created 10:24 Monday, 19MAY2014";
footnote2 "on the WIN System Using SAS 9.3";
run;
```

20

Substituting User-Defined Values

User-defined macro variables minimize entering of repetitive values.

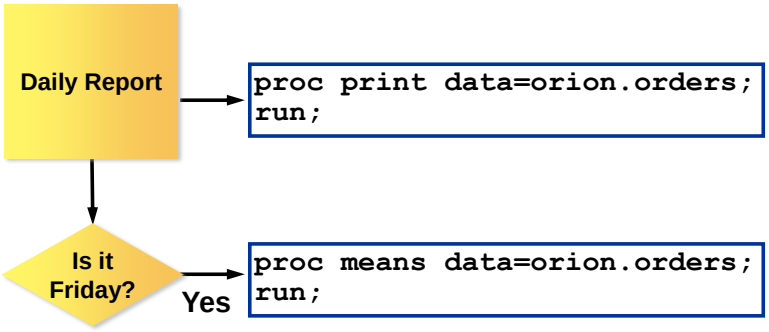
```
proc freq data=orion.order_fact;
   where year(order_date)=2011;
   table order_type;
   title "Order Types for 2011";
run;
proc means data=orion.order_fact;
   where year(order_date)=2011;
   class order_type;
   var Total_Retail_Price;
   title "Price Statistics for 2011";
run;
```

21

Conditional Processing

Macro programs support conditional construction of SAS code.

Example: Generate a detailed report every day and a summary report on Friday.



22

Repetitive Processing

Macro programs support repetitive generation of SAS code.

```
proc print data=orion.year2012;
run;
```

```
proc print data=orion.year2013;
run;
```

```
proc print data=orion.year2014;
run;
```

23

Data-Driven Applications

Macro programs support dynamic data-driven generation of SAS code.

Example: Create separate subsets of a selected data set for each unique value of a selected variable.

```
data AU CA DE IL TR US ZA;
   set orion.customer;
   select(country);
      when("AU") output AU;
      when("CA") output CA;
      when("DE") output DE;
      when("IL") output IL;
      when("TR") output TR;
      when("US") output US;
      when("ZA") output ZA;
      otherwise;
   end;
run;
```

24

Efficiency of Macro-Based Applications

Macro techniques can reduce program development and maintenance time.

- The efficiency of a SAS program depends on the efficiency of the underlying SAS code, regardless of whether the SAS code was entered manually or generated by macro techniques.

25

Developing Macro Applications

If a macro application generates SAS code, use a five-step approach to rapid development and debugging.

Step 1	Write and debug a SAS program without macro coding.
Step 2	Generalize the program by replacing hardcoded values with macro variable references. (Chapter 2)
Step 3	Create a macro definition with macro parameters. (Chapter 3)
Step 4	Add macro-level programming for conditional and iterative processing. (Chapter 5)
Step 5	Add data-driven customization. (Chapter 5)

30

1.02 Short Answer Poll

The macro facility is a ____ processing facility for automating and customizing SAS code.

31

1.3 Program Flow

Objectives

- Identify the tokens in a SAS program.
- Describe how a SAS program is tokenized, compiled, and executed.

35

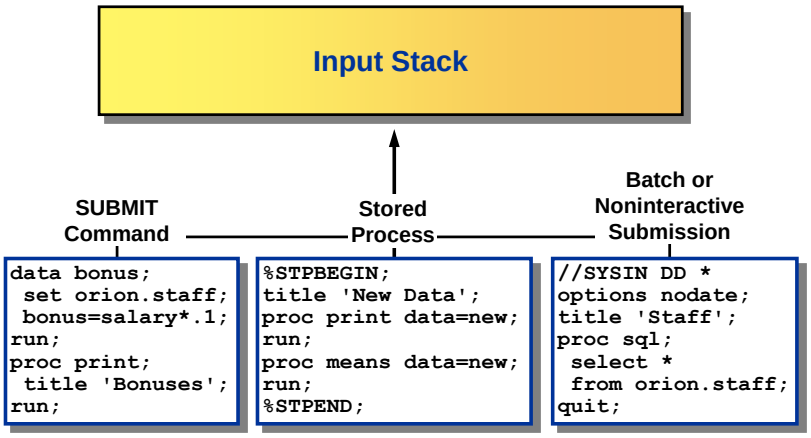
Program Flow

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

Program Flow

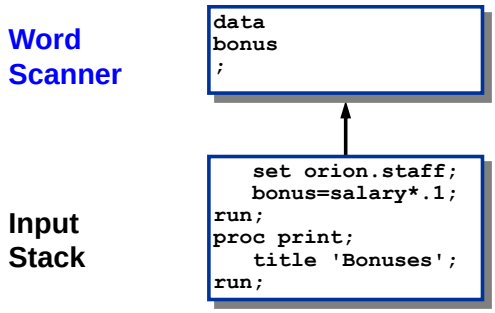


39

Program Flow

When SAS code is in the input stack, a component of SAS called the *word scanner* does the following:

- reads the text in the input stack, character by character, left to right, top to bottom
- breaks the text into fundamental units called *tokens*

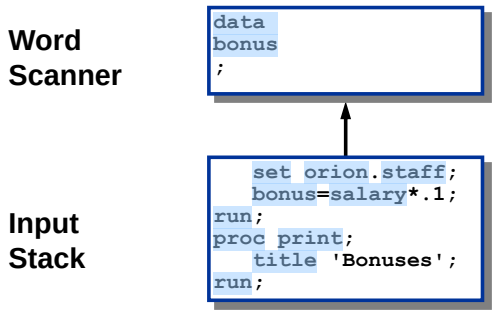


40

Program Flow

The word scanner recognizes four classes of tokens:

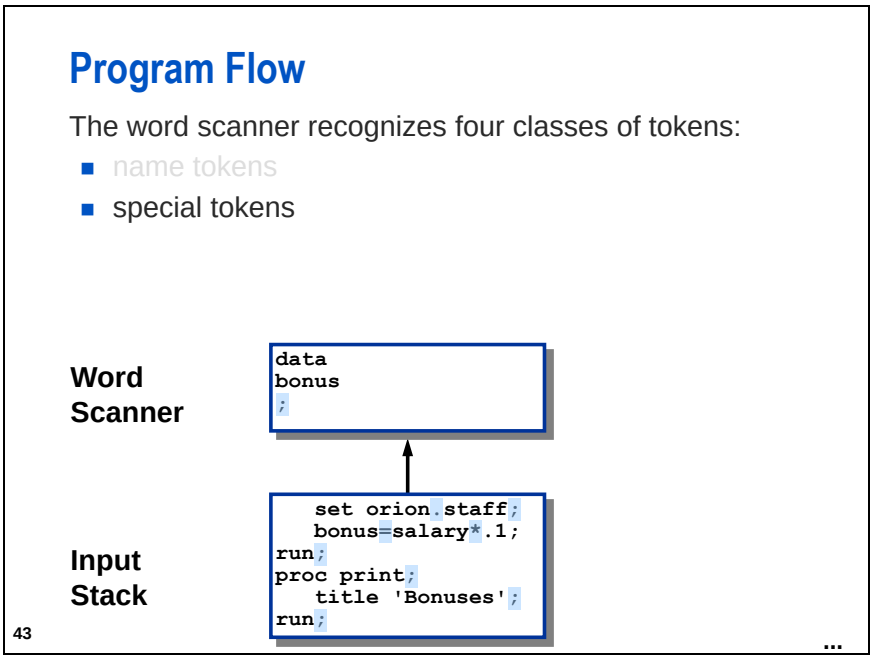
- name tokens



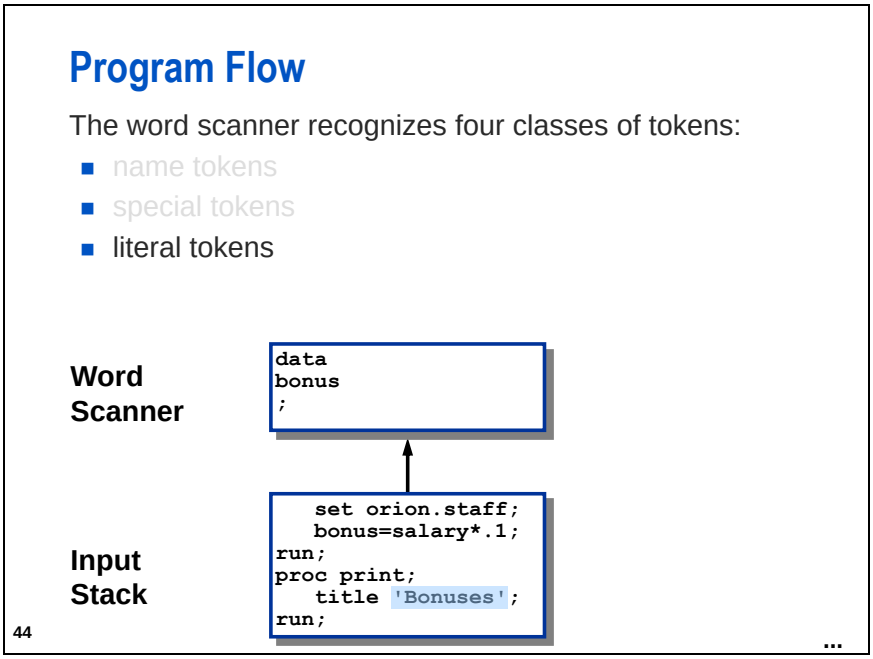
42 ...

Name tokens contain one or more characters beginning with a letter or underscore and continuing with letters, underscores, or numerals.

Format and informat names (COMMA11.2, $5.) contain a period.



Special tokens can be any character, or combination of characters, other than a letter, underscore, or numeral.



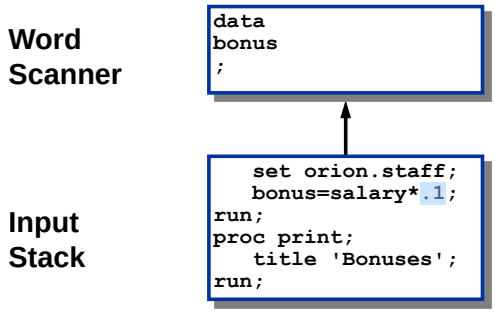
A *literal token* is a string of characters enclosed in single or double quotation marks.

The string is treated as a unit by the compiler.

Program Flow

The word scanner recognizes four classes of tokens:

- name tokens
- special tokens
- literal tokens
- number tokens



45

Number tokens can be integer numbers or floating-point numbers containing a decimal point, an exponent, or both.

A SAS date constant ('01jan2012'd) is a number token.

1.03 Poll

Blanks are tokens.

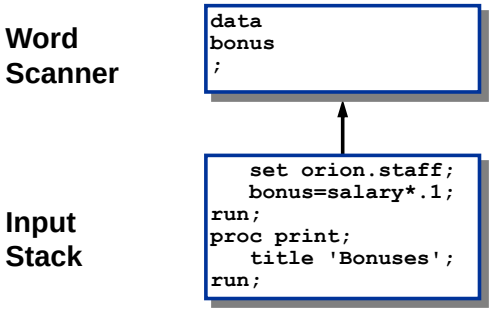
- ❍ True
- ❍ False

46

Tokenization

A token ends when the word scanner detects

- a blank
- the beginning of another token.

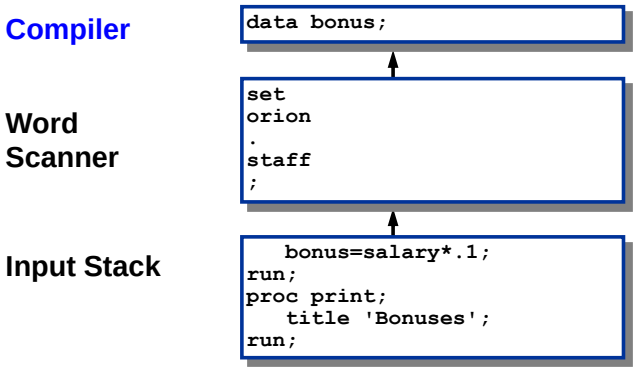


The maximum length of a token is 32,767 characters.

50

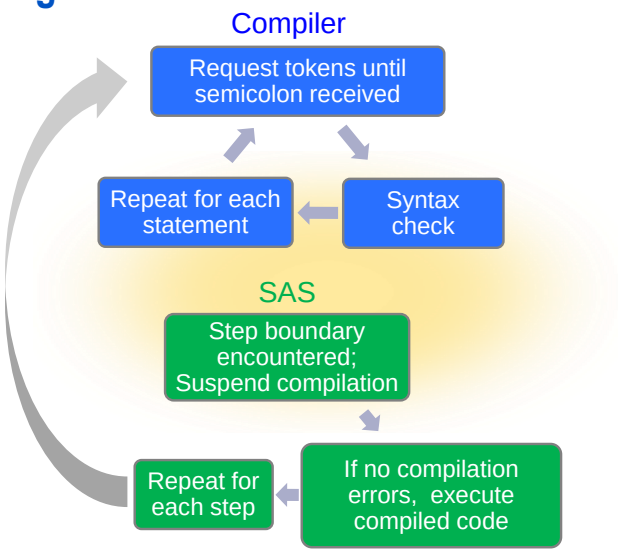
Program Flow

The word scanner passes tokens, one at a time, to the appropriate compiler, as the compiler demands.



52

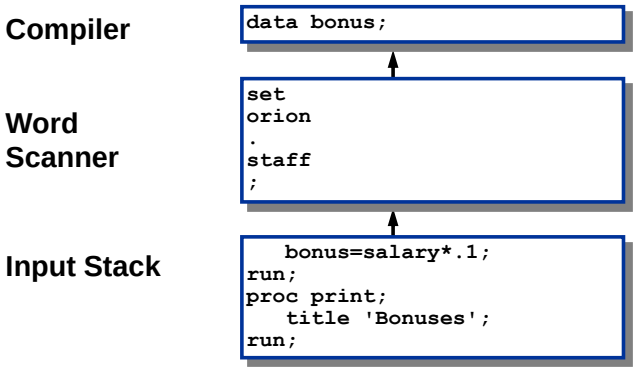
Program Flow



53

Program Flow

How does the macro language fit in?



54

Macro Triggers

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

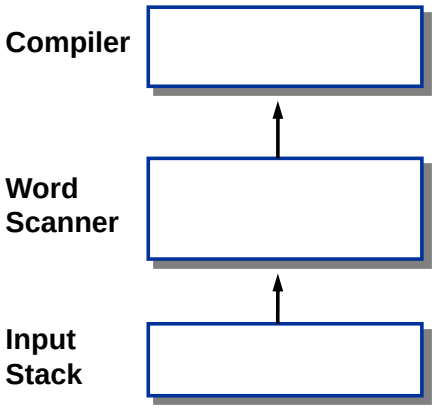
&name-token	a macro variable reference
%name-token	a macro statement, function, or call

- The word scanner passes macro triggers to the *macro processor*.

55

Program Flow

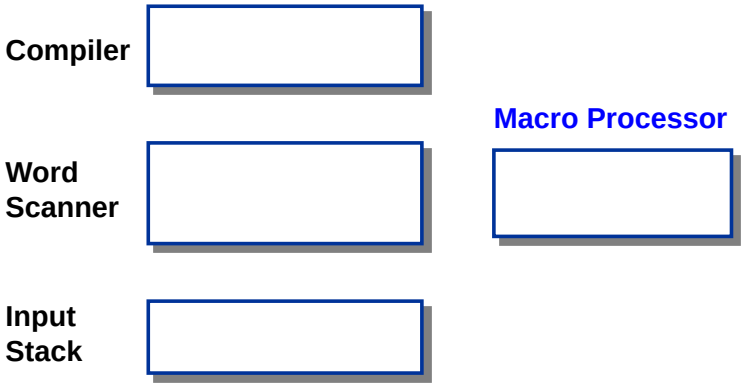
Where does the macro processor fit in?



56

Macro Processor

The macro processor executes macro triggers, including macro variable resolution, macro language statements, macro functions, and macro calls, requesting tokens as necessary.



57

Macro Statements

The following are characteristics of macro statements:

- begin with a percent sign (%) followed by a name token
- end with a semicolon
- represent macro triggers
- are executed by the macro processor

58

%PUT Statement

Example: Write text to the SAS log.

SAS Log

```
12 %put Invalid code!;
Invalid code!
```

%PUT text;

The %PUT statement does the following:

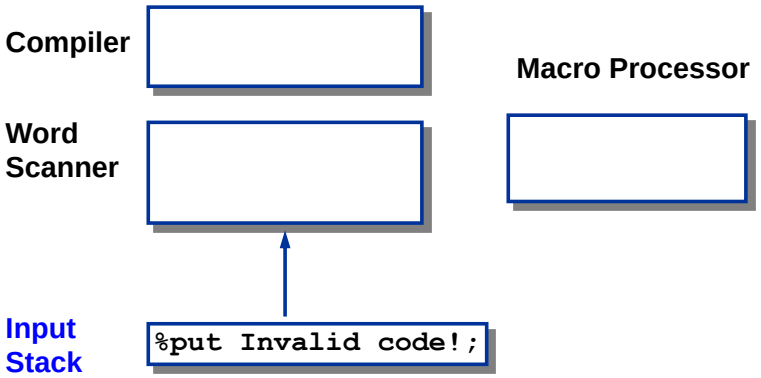
- writes text to the SAS log
- writes to column one of the next line
- writes a blank line if no text is specified

- Quotation marks are not required around text.
- The %PUT statement is valid in open code (anywhere in a SAS program).

59

Program Flow

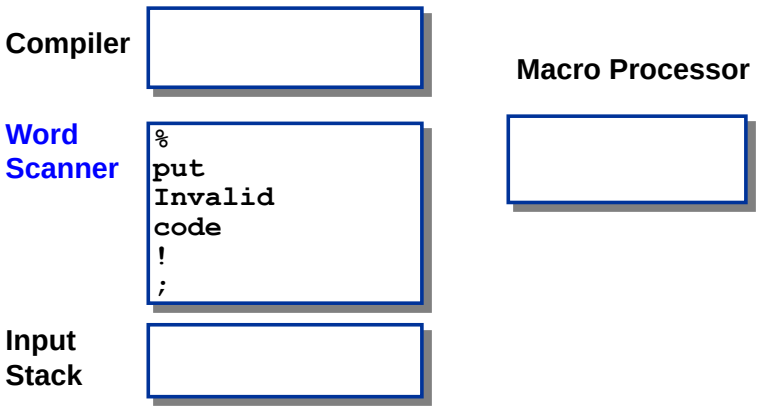
The %PUT statement is submitted.



60 ...

Program Flow

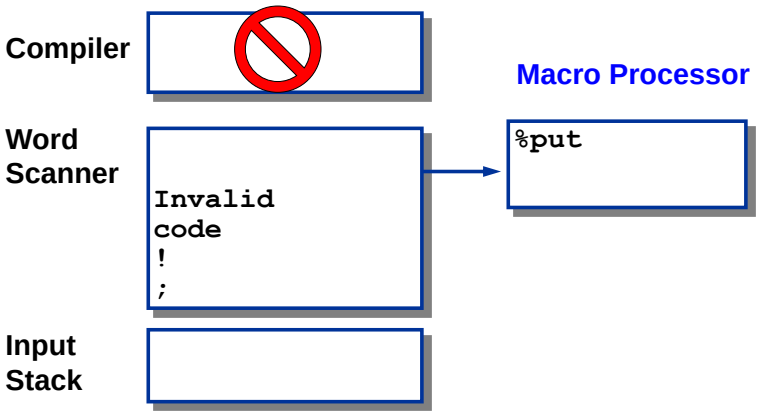
The statement is tokenized.



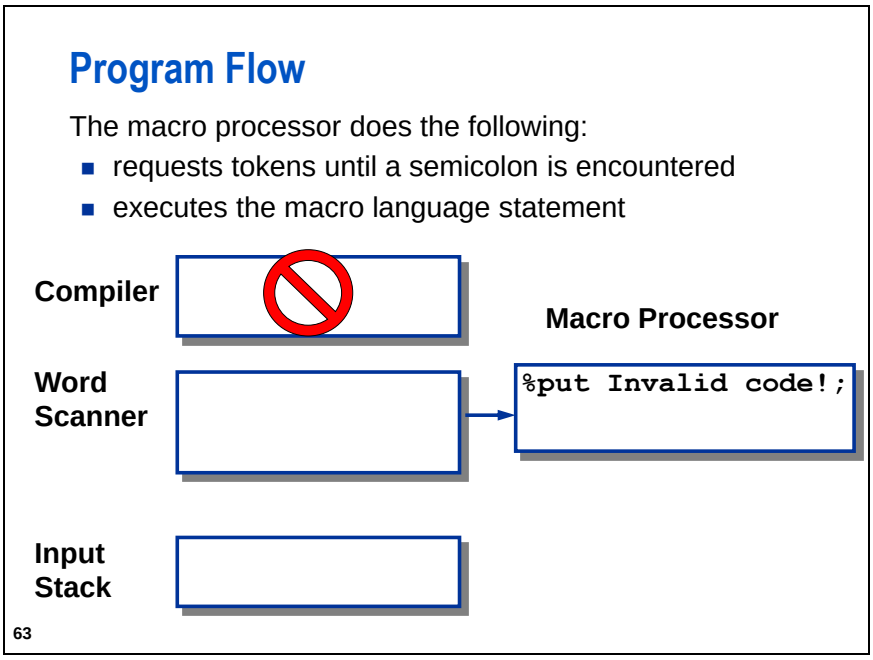
61 ...

Program Flow

The macro trigger is passed to the macro processor.



62 ...



Exercises

Level 1

2. Writing Text to the SAS Log with the %PUT Statement

Submit a %PUT statement that writes your name to the SAS log.

3. Writing NOTE, WARNING, and ERROR Messages to the SAS Log with the %PUT Statement

a. Open the program **m101e02** shown below into the Editor window.

```
%put NOTE: Is this a SAS note?;
%put WARNING: Is this a SAS warning?;
%put ERROR: Is this a SAS error?;
```

b. Submit the program and review the results in the SAS log. What is unusual about the results?

_____

_____

_____

c. Replace the colon so that a hyphen follows the keywords NOTE, WARNING, and ERROR. Submit the program and review the results in the SAS log. How do the results change?

_____

_____

_____

d. Modify the program so that the keywords NOTE, WARNING, and ERROR are in lowercase. Submit the program and review the results in the SAS log. Did the change affect the results?

_____

_____

_____

Level 2

4. Writing Special Characters to the SAS Log with the %PUT Statement

a. Submit the following %PUT statement:

```
%put Can you display a semicolon ; in your %PUT statement?;
```

b. Does the %PUT statement generate any text? If so, what is the text displayed in the SAS log?

_____

_____

c. Does the %PUT statement generate any error messages? If so, what is the cause of the error?

_____

_____

d. Is the second %PUT interpreted as text or a macro keyword?

_____

_____

1.4 Solutions

Solutions to Exercises

1. Creating Course Data Files

This is exercise is self-guided, so no solution is needed.

2. Writing Text to the SAS Log with the %PUT Statement

```
%put Jane Doe;
```

3. Writing NOTE, WARNING, and ERROR Messages to the SAS Log with the %PUT Statement

a. Open the program into the Editor window.

```
%put NOTE: Is this a SAS note?;
%put WARNING: Is this a SAS warning?;
%put ERROR: Is this a SAS error?;
```

b. The keywords NOTE, WARNING, and ERROR make the results of the %PUT statements resemble standard SAS NOTE, WARNING, and ERROR messages. Depending on the operating environment, the messages might be color-coded or just displayed in bold.

c. The hyphen forces the keywords NOTE, WARNING, and ERROR to be suppressed.

d. The keywords NOTE, WARNING, and ERROR are case-sensitive and do not have the desired effect when entered in lowercase.

4. Writing Special Characters to the SAS Log with the %PUT Statement

a. Submit the following %PUT statement:

```
%put Can you display a semicolon ; in your %PUT statement?;
```

b. The first semicolon in the %PUT statement is treated as a special token, not as plain text, and ends the statement. The %PUT statement generates the following text:

Partial SAS Log

```
1      %put Can you display a semicolon ; in your %PUT statement?;
Can you display a semicolon
```

c. An error message is generated after the first semicolon because the word IN is interpreted as an invalid keyword.

Partial SAS Log

```
1    %put Can you display a semicolon ; in your %PUT statement?;
                                       --
                                       180
ERROR 180-322: Statement is not valid or it is used out of proper order.
```

d. Because of the first semicolon, the second %PUT is interpreted as a macro keyword that generates the following text:

Partial SAS Log

```
statement?
```

The method for interpreting special tokens as plain text is addressed later in the course.

Solutions to Student Activities (Polls/Quizzes)

1.02 Short Answer Poll – Correct Answer

The macro facility is a text processing facility for automating and customizing SAS code.

32

1.03 Poll – Correct Answer

Blanks are tokens.

- ❍ True
- ❍ False

Blanks are *not* tokens. Blanks delimit tokens.

47

Chapter 2 Macro Variables

2.1 Introduction to Macro Variables

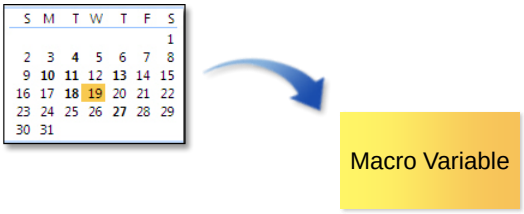
Objectives

- Describe the purpose of macro variables.
- Describe where macro variables are stored.
- Identify the two types of macro variables.

3

Business Scenario

Production programs might need to be updated with information, such as the current date, each time they are run. Use macro variables to minimize typing and make the code more flexible and reusable.



4

What Are Macro Variables?

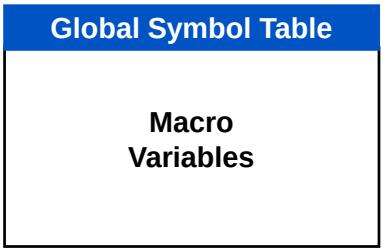
Macro variables store text, including the following:

- 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

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	03FEB15
SYSDATE9	03FEB2015
SYSDAY	Tuesday
SYSTIME	10:47
SYSUSERID	joeuser
.	.
.	.

7

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	19may2014
	UNITS	4

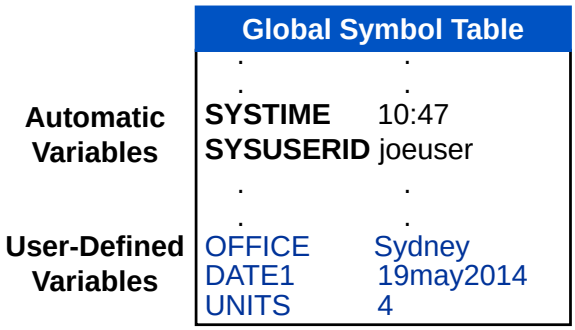
The global symbol table is deleted at the end of your SAS job or session.

8

Global Macro Variables

Macro variables in the global symbol table

- are global in scope (always available)
- have a minimum length of 0 characters (null value)
- have a maximum length of 65,534 (64K) characters
- store numeric tokens as text.



9

2.01 Short Answer Poll

What are the two types of macro variables?

10

2.2 Automatic Macro Variables

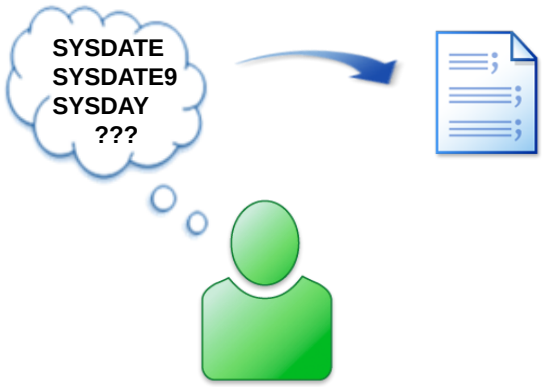
Objectives

- Identify selected automatic macro variables.
- Display automatic macro variables in the SAS log.

14

Business Scenario

Use automatic macro variables to minimize typing.



15

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 (03FEB15)
SYSDATE9	Date of SAS invocation (03FEB2015)
SYSDAY	Day of the week of SAS invocation (Tuesday)
SYSTIME	Time of SAS invocation (10:47).
SYSSCP	Operating system abbreviation (WIN, OS, HP 64)
SYSVER	Release of SAS software (9.4)
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

```
639582  %put _automatic_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSADDRBITS 64
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 3000
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATASTEPPHASE
AUTOMATIC SYSDATE 03FEB15
AUTOMATIC SYSDATE9 03FEB2015
AUTOMATIC SYSDAY Tuesday
```

%PUT _AUTOMATIC_**;**

19

2.02 Poll

Submit the statement below.

```
%put _automatic_;
```

Does **SYSTIME** match the current time?

- ❍ Yes
- ❍ No

20

2.3 Macro Variable References

Objectives

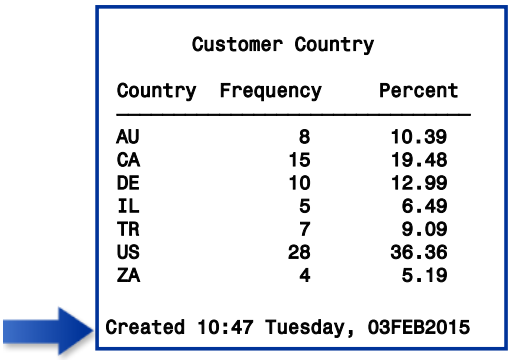
- Explain how macro variable references are handled by the word scanner and macro processor.

24

Business Scenario

Use automatic macro variables to generate footnotes with the following information:

- report time
- report day
- report date



Customer Country

Country	Frequency	Percent
AU	8	10.39
CA	15	19.48
DE	10	12.99
IL	5	6.49
TR	7	9.09
US	28	36.36
ZA	4	5.19

Created 10:47 Tuesday, 03FEB2015

25

Macro Variable References

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

Macro variable references

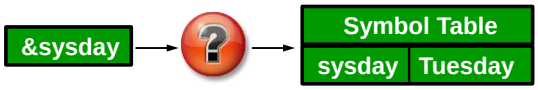
- are also called *symbolic references*
- can appear anywhere in a program
- are not case sensitive
- represent macro triggers
- are passed to the macro processor.

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

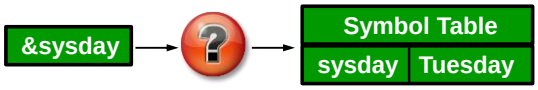


27

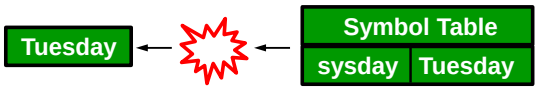
Macro Variable References

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

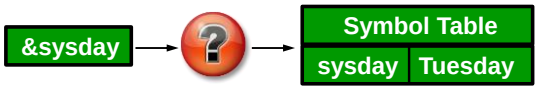


28

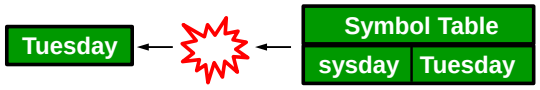
Macro Variable References

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



- issues a warning to the SAS log if the macro variable is not found in the symbol table

```
WARNING: Apparent symbolic reference SYDAY not resolved.
```

29

Macro Variable References

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

SAS Log

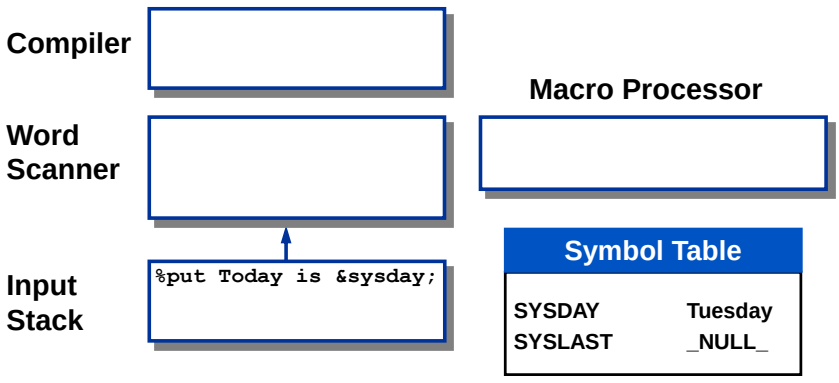
```
14    %put Today is &sysday;
Today is Tuesday
15    %put &=sysday;
SYSDAY=Tuesday
```

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

30

An equal sign between the ampersand and the macro variable name displays the macro variable's name followed by the macro variable's value.

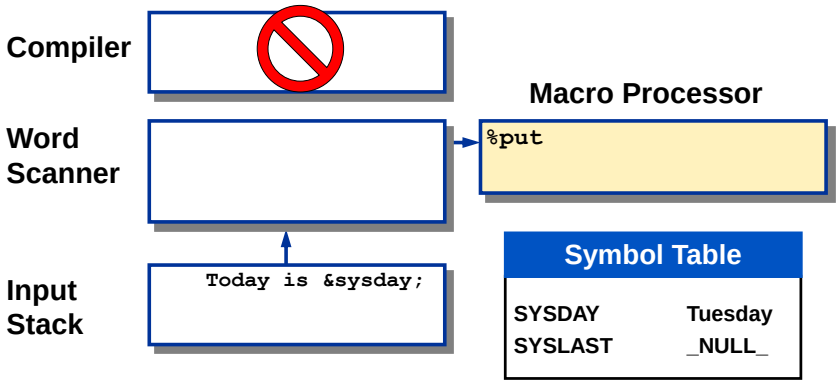
Substitution within a Macro Statement



31 ...

Substitution within a Macro Statement

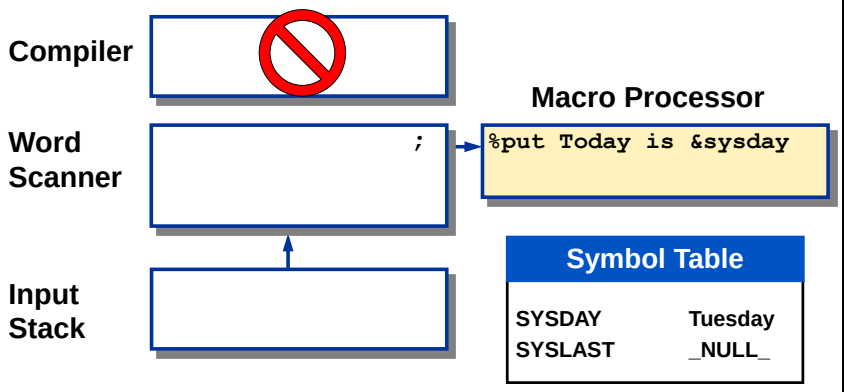
When a macro trigger is encountered, it is passed to the macro processor for evaluation.



32 ...

Substitution within a Macro Statement

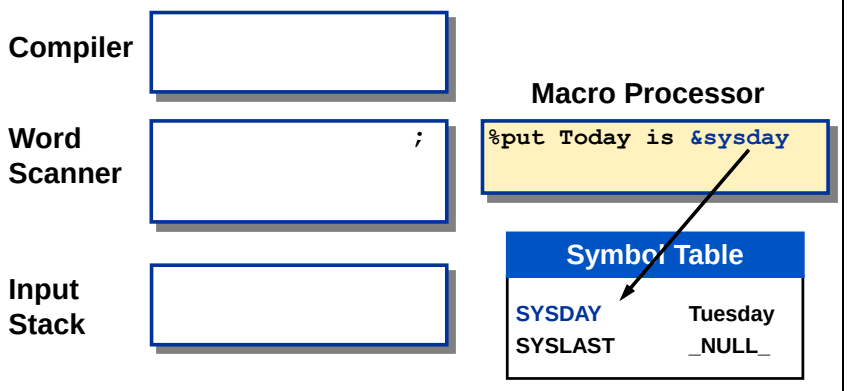
The macro processor requests additional tokens.



33 ...

Substitution within a Macro Statement

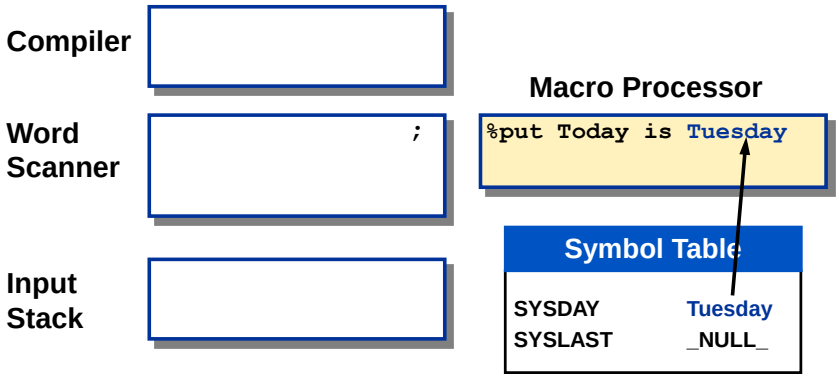
The macro variable reference triggers the macro processor to search the symbol table for the reference.



34 ...

Substitution within a Macro Statement

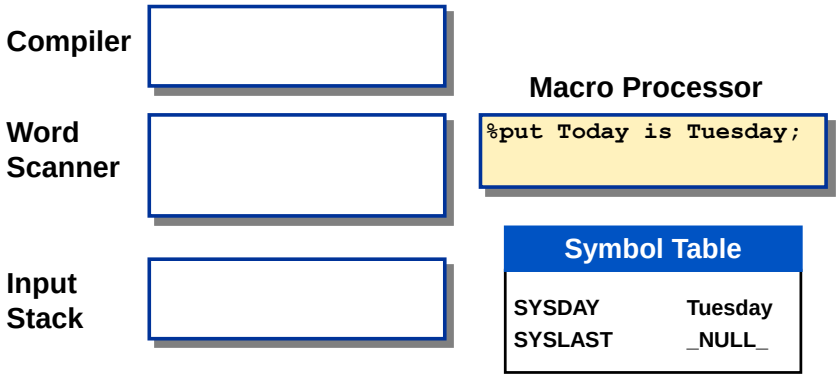
The macro processor resolves the macro variable reference, substituting its value.



35 ...

Substitution within a Macro Statement

The macro processor requests additional tokens until a semicolon is encountered. Then it executes the %PUT statement, writing the resolved text to the SAS log.



36

2.03 Short Answer Poll

Open and submit **m102d01**. What are the footnotes in the PROC FREQ output?

```
proc freq data=orion.Customer;
   table Country / nocum;
   footnote 'Created &systime &sysday, &sysdate9';
run;
```

37 m102d01

Substitution within a SAS Literal

To reference macro variables within a literal, enclose the literal in double quotation marks.

```
proc freq data=orion.Customer;
   table Country / nocum;
   footnote "Created &systime &sysday, &sysdate9";
run;
```

39 m102d01

Substitution within a SAS Literal

PROC FREQ Output

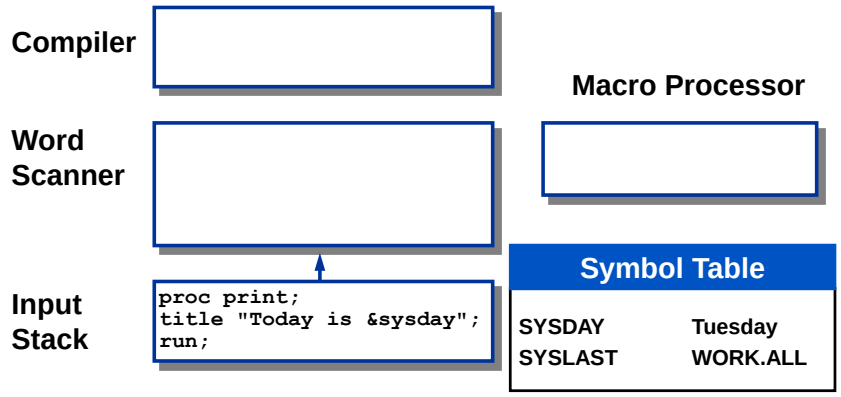
Customer Country

Country	Frequency	Percent
AU	8	10.39
CA	15	19.48
DE	10	12.99
IL	5	6.49
TR	7	9.09
US	28	36.36
ZA	4	5.19

Created 10:47 Friday, 06JAN2014

40 m102d01

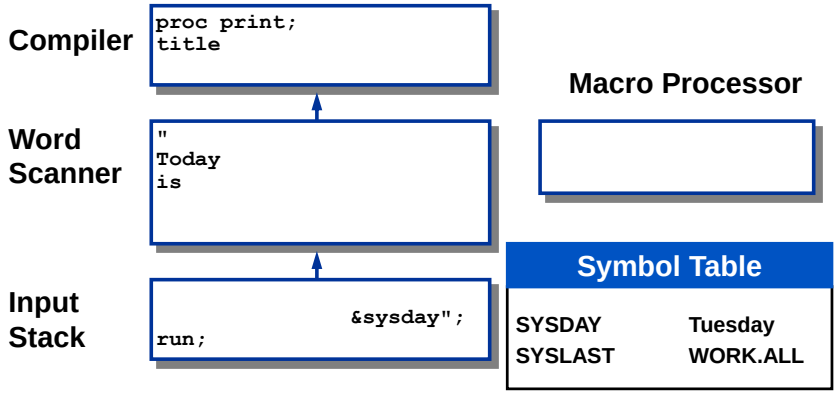
Substitution within a SAS Literal



41 ...

Substitution within a SAS Literal

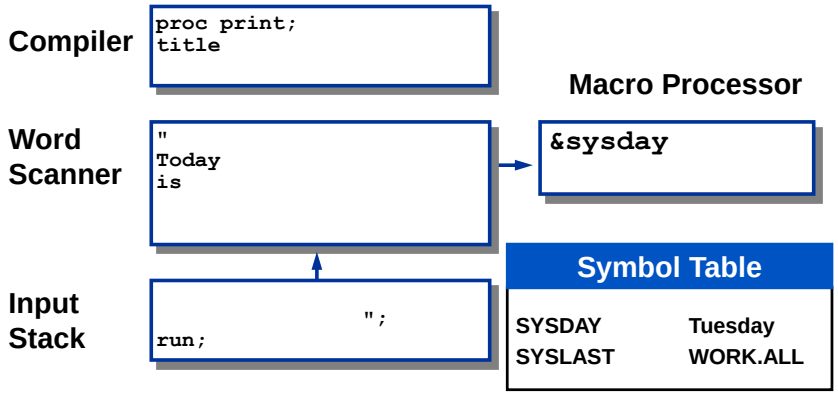
SAS statements are passed to the compiler.



42 ...

Substitution within a SAS Literal

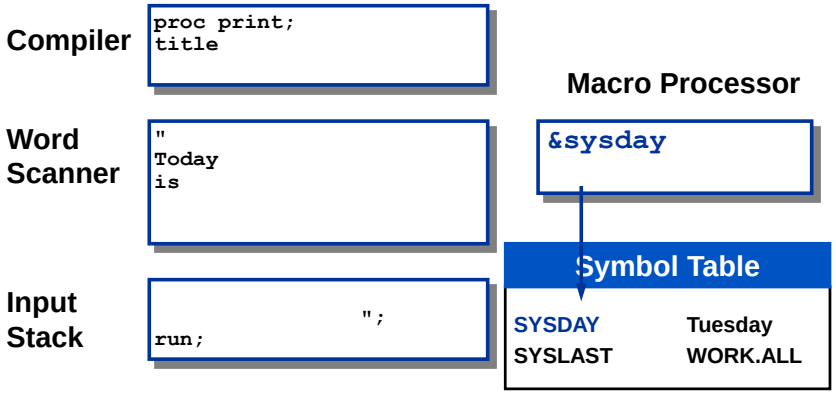
The macro trigger is passed to the macro processor.



43 ...

Substitution within a SAS Literal

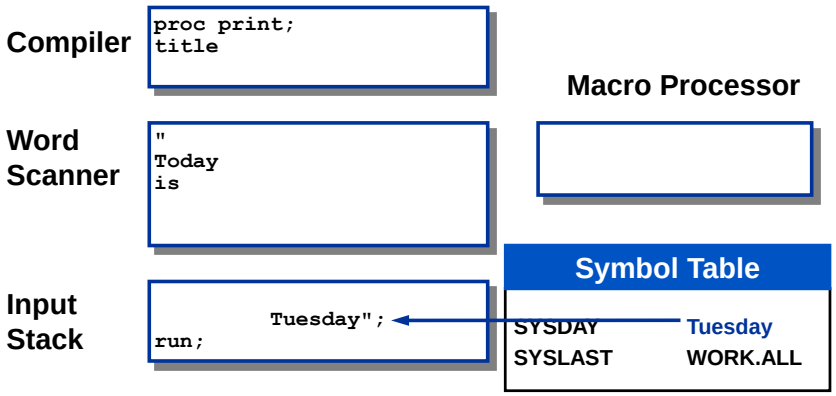
The macro processor searches the symbol table.



44 ...

Substitution within a SAS Literal

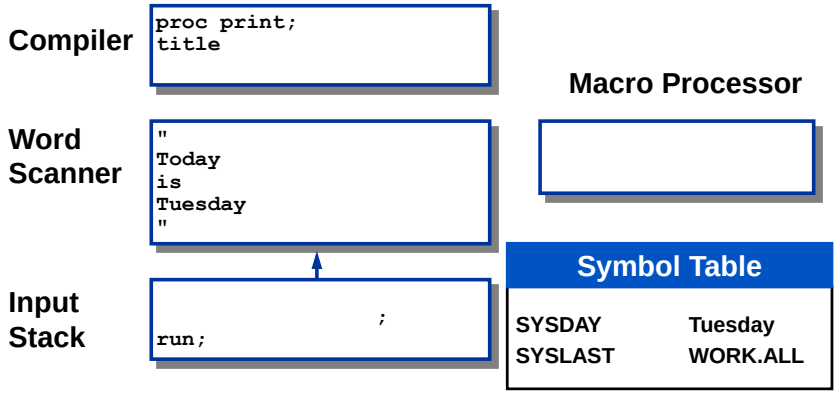
The resolved reference is passed back to the input stack.



45 ...

Substitution within a SAS Literal

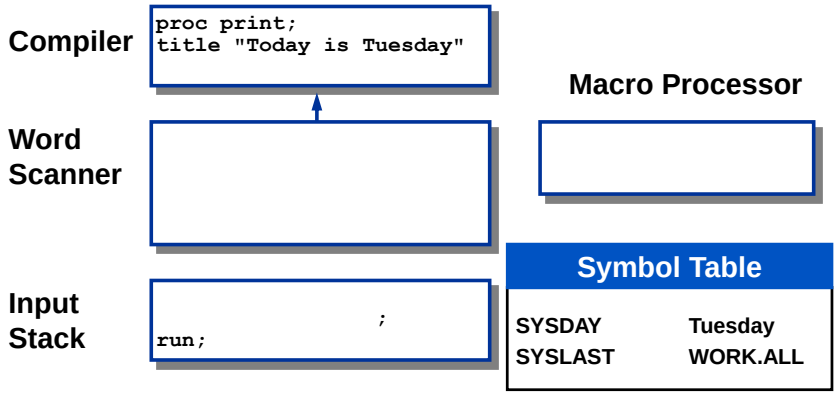
Word scanning continues.



46 ...

Substitution within a SAS Literal

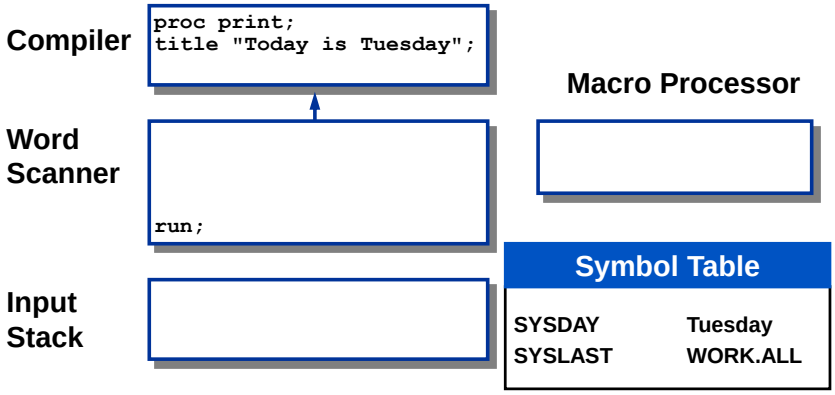
The literal is passed to the compiler as a unit.



47 ...

Substitution within a SAS Literal

When a step boundary is encountered, compilation ends and execution begins.



48

2.04 Multiple Choice Poll

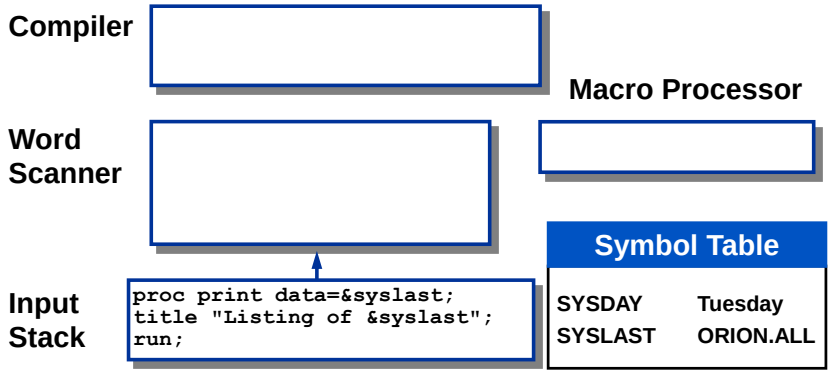
Macro variable references are resolved by which of the following?

a. SAS compiler
b. macro processor
c. word scanner

49

Substitution within SAS Code

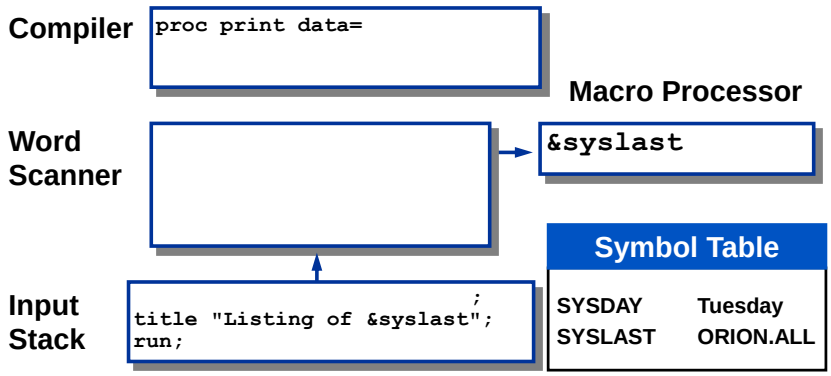
Example: Generalize PROC PRINT to print the last created data set, using the automatic macro variable **SYSLAST**.



51 ...

Substitution within SAS Code

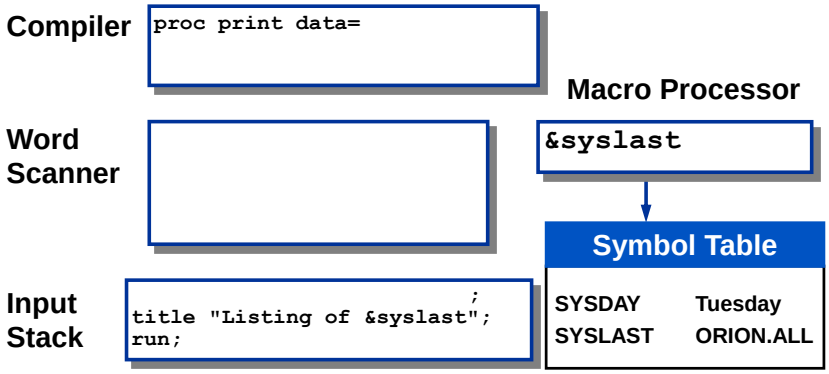
SAS statements are passed to the compiler. When a macro trigger is encountered, it is passed to the macro processor for evaluation.



52 ...

Substitution within SAS Code

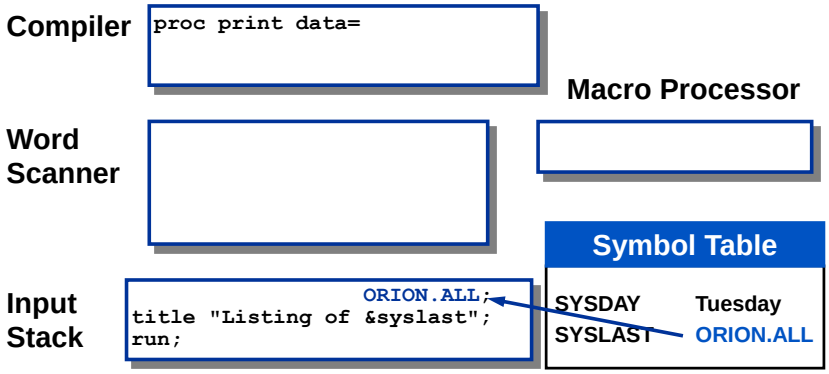
The *macro variable reference* triggers the macro processor to search the symbol table for the reference.



53 ...

Substitution within SAS Code

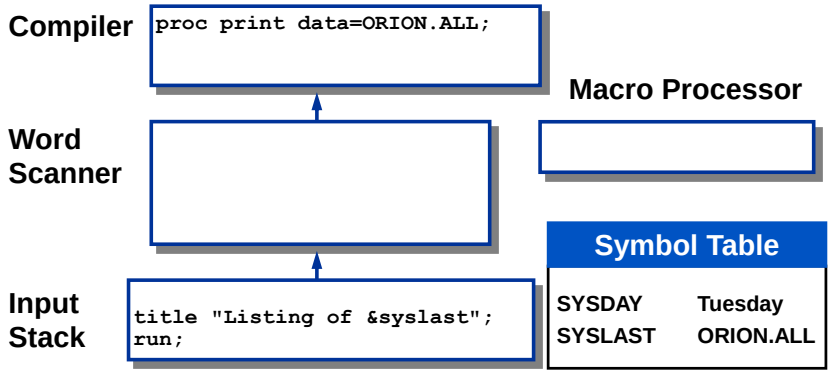
The macro processor resolves the macro variable reference, passing its resolved value back to the input stack.



54 ...

Substitution within SAS Code

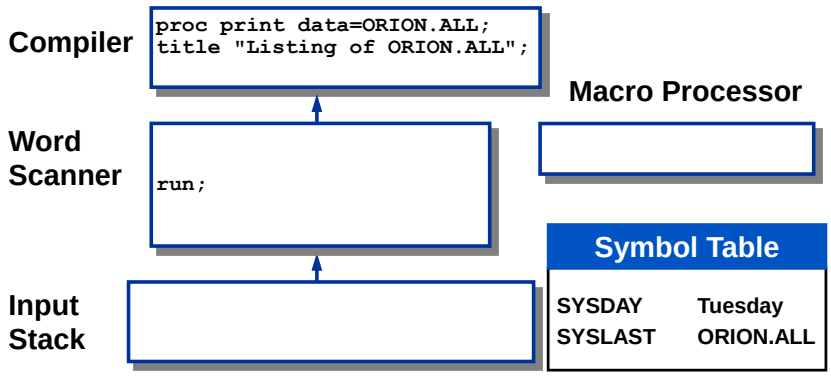
Word scanning continues.



55 ...

Substitution within SAS Code

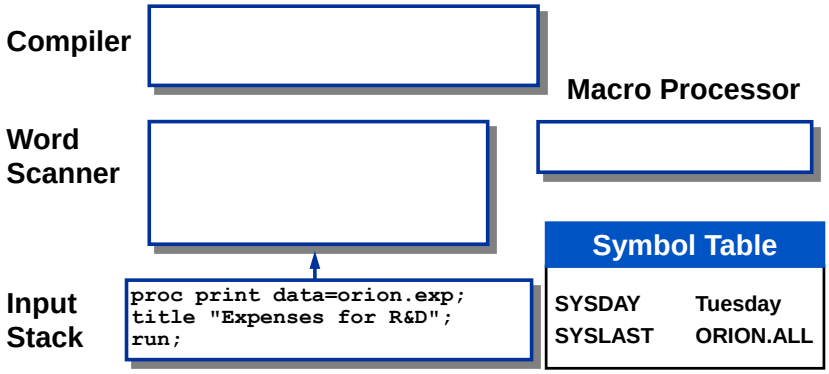
A step boundary is encountered. Compilation ends. Execution begins.



56

Unresolved Reference

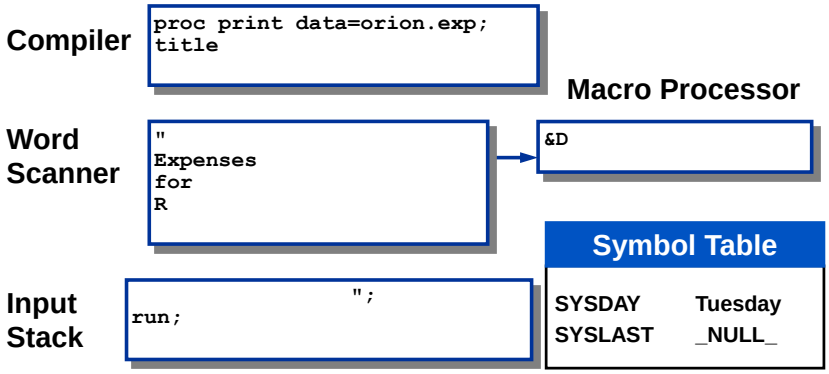
Example: Reference a nonexistent macro variable within a SAS literal.



57 ...

Unresolved Reference

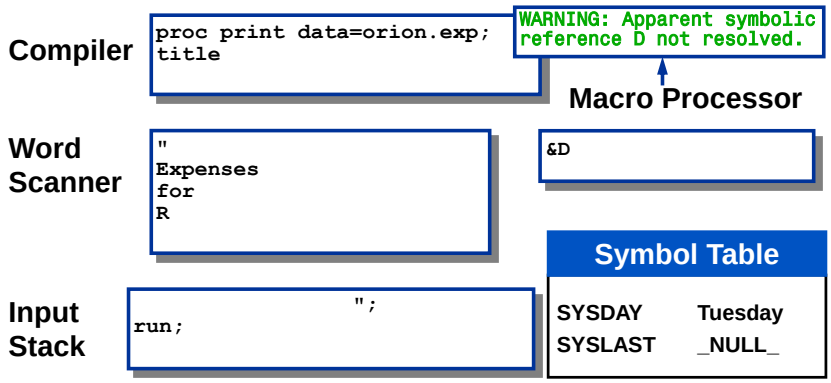
The macro trigger is passed to the macro processor for evaluation.



58 ...

Unresolved Reference

The macro processor issues a warning to the SAS log when it cannot resolve a reference.

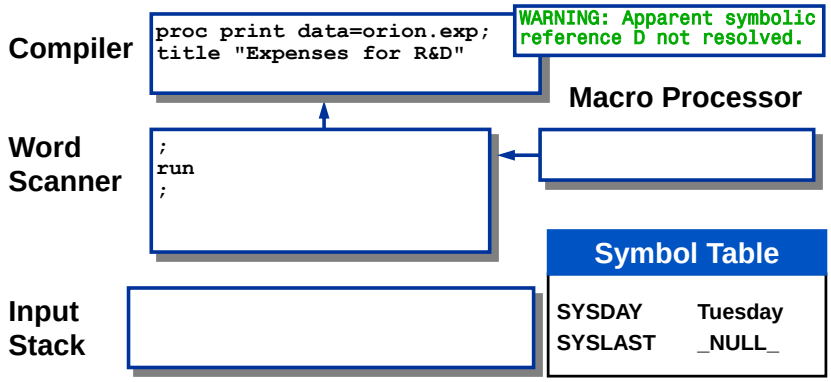


59

...

Unresolved Reference

If the macro processor cannot resolve a reference, the tokens are passed back to the word scanner. The word scanner passes them to the compiler.



60

Unresolved Reference

Example: Reference a nonexistent macro variable within SAS code.

```
proc print data=&sydlast;
title "Listing of &syslast";
run;
```

61

Unresolved Reference

SAS Log

```
1    proc print data=&sydlast;
                     -
                     22
                     200
WARNING: Apparent symbolic reference SYDLAST not resolved.
ERROR: File WORK.SYDLAST.DATA does not exist.
ERROR 22-322: Expecting a name.
ERROR 200-322: The symbol is not recognized and will be ignored.
2    title "Listing of &syslast";
3    run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.65 seconds
      cpu time            0.09 seconds
```

62

Exercises

Submit a LIBNAME statement to assign the **orion** libref to the course SAS library according to instructions provided by the instructor.

libname orion '_____';

Level 1

1. Displaying Automatic Macro Variables

a. Use the %PUT statement to list all automatic macro variables in the SAS log.

b. What are the values of the following automatic macro variables?

- **SYSLAST** _____
- **SYSUSERID** _____
- **SYSTIME** _____
- **SYSDATE9** _____

c. Are the values for the automatic macro variables **SYSTIME** and **SYSDATE9** accurate?

_____

_____

Level 2

2. Using Automatic Macro Variables

a. Using the SORT procedure, sort the data set **orion.continent** by **Continent_Name**.

Use the OUT= option in the PROC SORT statement so that you do not overwrite the original data set.

b. Using the PRINT procedure and an automatic macro variable, print the most recently created data set and display the data set name in the title.

c. Submit the program and examine the results.

3. Using Automatic Macro Variables

a. What is the value of the automatic macro variable **SYSLAST** after the following DATA step is submitted?

_____

```
data new;
   set orion.continent;
run;
```

b. What is the value of the automatic macro variable **SYSLAST** after the following PROC PRINT step is submitted?

_____

```
proc print data=orion.continent;
run;
```

Challenge

4. Using SAS Date Constants

a. Open the **m102e04** program shown below into the Editor window.

```
proc print data=orion.employee_payroll;
   format Birth_Date Employee_Hire_Date date9.;
run;
```

b. Modify the program so that it subsets the data to return only the employees hired between January 1, 2007, and today. Use the automatic macro variable **SYSDATE9** to return today's date.

PROC PRINT Output

Obs	Employee_ID	Employee_ Gender	Salary	Birth_ Date	Employee_ Hire_Date	Employee_ Term_Date	Marital_ Status	Dependents
310	121034	M	27110	23AUG1988	01JAN2007	.	S	0
361	121085	M	32235	12NOV1986	01JAN2007	.	S	0
364	121088	M	27240	10JUN1988	01JAN2007	.	S	0

2.4 User-Defined Macro Variables

Objectives

- Create user-defined macro variables.
- Display values of user-defined macro variables in the SAS log.

66

Business Scenario

Further automate your programs with user-defined macro variables.

67

%LET Statement

The %LET statement creates a macro variable and assigns it a value.

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

%LET *variable=value*;

Symbol Table	
name	Ed Norton

- %LET statements are valid in open code (anywhere in a SAS program).

68

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

Determine the name and value of each macro variable created by these %LET statements.

```
%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

71 ...

%LET Statement Examples

Leading and trailing blanks are removed.

```
%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

72 ...

%LET Statement Examples

Quotation marks are stored as part of the value.

```
%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 '

73 ...

Storing quotation marks as part of a macro variable's value is ***not*** recommended.

%LET Statement Examples

Quotation marks are stored as part of the value.

```
%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"

74 ...

Storing quotation marks as part of a macro variable's value is ***not*** recommended.

%LET Statement Examples

A null value is stored.

```
%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	

75 ...

%LET Statement Examples

Mathematical expressions are not evaluated.

```
%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	3+4

76 ...

%LET Statement Examples

Numeric tokens are stored as character strings.

```
%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	3+4
total	0

77 ...

%LET Statement Examples

The macro trigger is evaluated before the %LET statement executes. The previous value of **total** is replaced.

```
%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	3+4
total	0+3+4

78 ...

%LET Statement Examples

The text value is stored.

```
%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	3+4
total	0+3+4
x	varlist

79 ...

%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	3+4
total	0+3+4
x	varlist
varlist	name age height

80

2.05 Short Answer Poll

What is the value of the macro variable **z**?

```
%let x=15;
%let y=10;
%let z=&x-&y;
```

81

%LET Statement Examples

Use a macro variable to select a ***numeric*** value.

Step 1 Hardcode the numeric value.

```
proc print data=orion.Order_Fact;
   where Quantity > 4;
   var Order_Date Product_ID Quantity;
   title "Orders exceeding 4 units";
run;
```

83 m102d02

%LET Statement Examples

Use a macro variable to select a ***numeric*** value.

Step 2 Create a macro variable.

```
%let units=4;

proc print data=orion.Order_Fact;
   where Quantity > 4;
   var Order_Date Product_ID Quantity;
   title "Orders exceeding 4 units";
run;
```

84 m102d02

Macro variables store numbers as text, not as numeric values.

%LET Statement Examples

Use a macro variable to select a ***numeric*** value.

Step 3 Reference the macro variable.

```
%let units=4;

proc print data=orion.Order_Fact;
   where Quantity > &units;
   var Order_Date Product_ID Quantity;
   title "Orders exceeding &units units";
run;
```

Partial SAS Log

```
NOTE: There were 6 observations read from the data set ORION.ORDER_FACT.
      WHERE Quantity>4;
```

85 m102d02

Macro variables store numbers as text, not as numeric values.

%LET Statement Examples

Use a macro variable to select a ***character*** value.

Step 1 Hardcode the character value.

```
proc print data=orion.Employee_Addresses;
   where City="Sydney";
   var Employee_Name;
   title "Sydney Employees";
run;
```

86 m102d03

%LET Statement Examples

Use a macro variable to select a ***character*** value.

Step 2 Create a macro variable.

```
%let office=Sydney;

proc print data=orion.Employee_Addresses;
   where City="Sydney";
   var Employee_Name;
   title "Sydney Employees";
run;
```

87 m102d03

%LET Statement Examples

Use a macro variable to select a ***character*** value.

Step 3 Reference the macro variable.

```
%let office=Sydney;

proc print data=orion.Employee_Addresses;
   where City="&office";
   var Employee_Name;
   title "&office Employees";
run;
```

Partial SAS Log

```
NOTE: There were 67 observations read from the data set
      ORION.EMPLOYEE_ADDRESSES.
      WHERE City='Sydney';
```

88 m102d03

%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

          Order_
Obs         Date      Product_ID    Quantity

526    05JUN2011    240100100679         2
527    06JUN2011    230100400007         1
```

89 m102d04

2.06 Multiple Choice Poll

Which WHERE statement is correct?

a. where City=&office;
b. where City='&office';
c. where City="&office";

90

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 DATE1 25may2011
GLOBAL DATE2 15jun2011
GLOBAL OFFICE Sydney
GLOBAL UNITS 4
```

%PUT _USER_;

92

Use _ALL_ to display all user-defined and automatic macro variables in the SAS log, as follows:

```
%put _all_;
```

Displaying Macro Variables

The SYMBOLGEN system option writes macro variable values to the SAS log as they are resolved.

SAS Log

OPTIONS SYMBOLGEN;

```
176  options symbolgen;
177  %let office=Sydney;
178  proc print data=orion.employee_addresses;
179     where city="&office";
SYMBOLGEN:  Macro variable OFFICE resolves to Sydney
180     var employee_name;
SYMBOLGEN:  Macro variable OFFICE resolves to Sydney
181     title "&office Employees";
182  run;
```

- The default value is NOSYMBOLGEN.

93

After debugging, issue the following statement to turn off the SYMBOLGEN option:

```
options nosymbolgen;
```

Deleting User-Defined Macro Variables

The %SYMDEL statement deletes one or more user-defined macro variables from the global symbol table.

%SYMDEL *macro-variables*;

```
%symdel office units;
```

Global Symbol Table	
DATE1	25may2011
DATE2	15jun2011
~~OFFICE~~	~~Sydney~~
~~UNITS~~	~~4~~

- Delete unneeded macro variables from the global symbol table to release memory.

94

Exercises

Level 1

5. Defining and Using Macro Variables for Character Substitution

a. Open the **m102e05** program shown below into the Editor window. Submit the program and examine the output that it creates.

```
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains 'Gold';
   title 'Gold Customers';
run;
```

b. Precede the program with a %LET statement to assign the value *Gold* to **type**. Modify the program so that the two occurrences of *Gold* are replaced by references to the macro variable **type**. Submit the program. It produces the same output as before.

c. Include the appropriate system option to display resolved values of macro variables in the SAS log. Resubmit the program and examine the log.

d. Modify the value of **type** to *Internet*. Resubmit the program and examine the log.

e. Turn off the system option from part **c** above.

Level 2

6. Defining and Using Macro Variables for Numeric Substitution

a. Open the **m102e06** program shown below into the Editor window. Edit the program to display only the Gold-level customers between the ages of 30 to 45.

Customer ages range from 19 to 73.

```
%let type=Gold;
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type";
   title "&type Customers";
run;
```

SAS Output

Gold Customers between 30 and 45

Obs	Customer_Name	Customer_ Gender	Customer_ Age
3	Cornelia Krahl	F	33
11	Oliver S. Füßling	M	43
32	James Klisurich	M	38
35	Viola Folsom	F	38
40	Kyndal Hooks	F	43
57	Rita Lotz	F	43
75	Angel Borwick	F	38

b. Modify the program so that the values *30* and *45* are replaced by references to the macro variables **age1** and **age2**, respectively.

c. Include the appropriate system option to display resolved values of macro variables in the SAS log. Resubmit the program and examine the log.

d. Modify the values of **age1** and **age2**. Resubmit the program and examine the log.

e. Turn off the system option from part **c** above.

7. Deleting Macro Variables

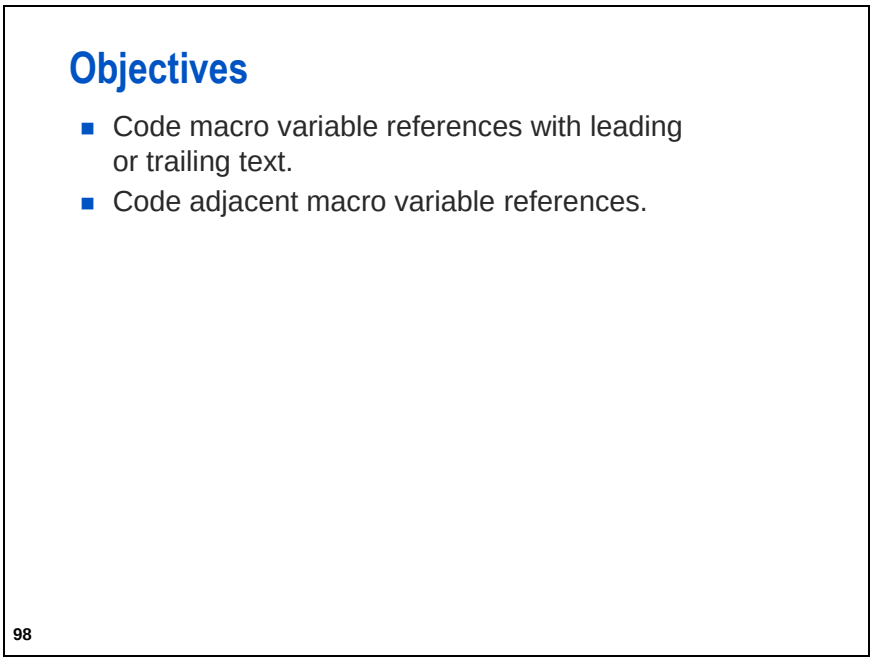
a. Open the program **m102e07** program shown below into the Editor window. Submit the program to create the macro variables.

```
%let pet1=Paisley;
%let pet2=Sitka;
```

b. Delete all user-defined macro variables.

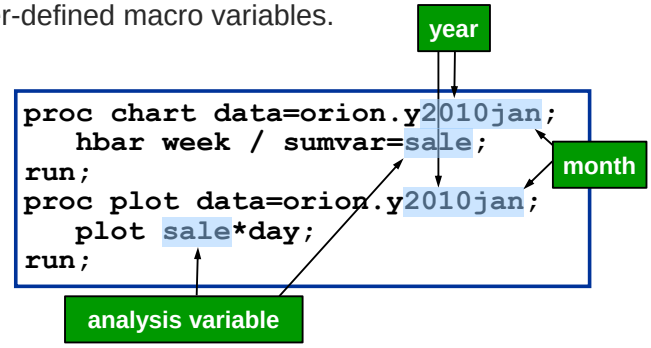
c. Use the %PUT statement to verify that the macro variable deletion is successful.

2.5 Delimiting Macro Variable References



Business Scenario

The production program below requires the user to repetitively enter the desired year, month, and analysis variable. Automate the program with user-defined macro variables.



What is different about this scenario?

99

Combining Macro Variables with Text

A macro variable reference can appear anywhere, but additional syntax is sometimes required.

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

1. Leading text
2. Adjacent macro variable references
3. Trailing text

Which of the above situations might pose a problem?

100

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

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

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

Setup for the Poll

SAS software includes the following procedures:

Base SAS	PROC CHART	PROC PLOT
SAS/GRAPH	PROC GCHART	PROC GPLOT

Create a macro variable that toggles between a Base SAS or SAS/GRAPH procedure.

105

2.07 Short Answer Poll

Create a macro variable that toggles between a Base SAS or SAS/GRAPH procedure.

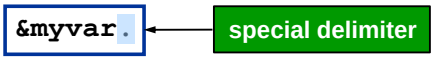
```
/* GRAPHICS should be null or G */
%let graphics=g;
proc &graphicschart data=orion.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphicsplot data=orion.y&year&month;
   plot &var*day;
run;
```

What is the problem with this reference?

106

Trailing Text Solution

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



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 Solution

Use a period to delimit the end of the macro variable reference.

```
/* GRAPHICS should be null or G */
%let graphics=g;
proc &graphics.chart data=orion.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphics.plot data=orion.y&year&month;
   plot &var*day;
run;
```

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

109

- The generated code does not include the period.

2.08 Short Answer Poll

Modify the previous example to include a macro variable that stores a libref.

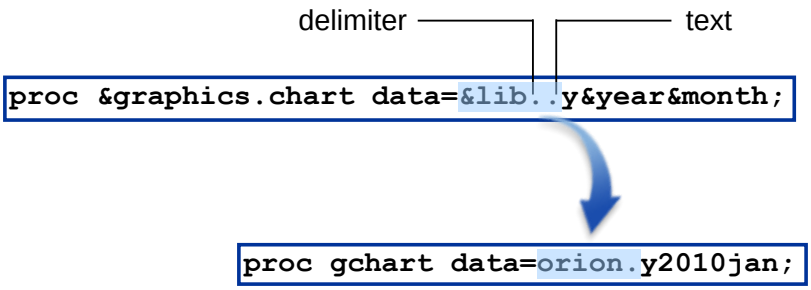
```
%let lib=orion;
%let graphics=g;
libname &lib "&path";
proc &graphics.chart data=&lib.y&year&month;
   hbar week / sumvar=&var;
run;
proc &graphics.plot data=&lib.y&year&month;
   plot &var*day;
run;
```

What is the problem this time?

110

Corrected Program

Use another period.



112

Exercises

Level 1

8. Consecutive Macro Variable References

a. Open the **m102e08** program shown below into the Editor window. Submit the program and examine the output that it creates.

```
proc print data=orion.organization_dim;
   where Employee_Hire_Date='01AUG2006'd;
   id Employee_ID;
   var Employee_Name Employee_Country Employee_Hire_Date;
   title 'Personal Information for Employees Hired in AUG 2006';
run;
```

b. Precede the program with %LET statements to assign the value *AUG* to **month** and the value *2006* to **year**. Modify the program so that the two occurrences of *AUG* and *2006* are replaced by references to the macro variables **month** and **year**, respectively. Submit the program. It produces the same output as before.

c. Modify the value of **month** to *JUL* and **year** to *2003*. Resubmit the program.

Level 2

9. Macro Variable References with Delimiters

a. Open the **m102e09** program shown below into the Editor window. Submit the program and examine the output that it creates.

```
proc print data=orion.organization_dim;
   id Employee_ID;
   var Employee_Name Employee_Country Employee_Gender;
   title 'Listing of All Employees From Orion.Organization_Dim';
run;
```

b. Modify the program so that all occurrences of *Organization* and *Employee* are replaced with macro variable references called **dsn** and **var**, respectively. Submit the program and verify the output.

When substituting for the hardcoded **Employees** in the TITLE statement, be sure to keep the ending **s** as part of the title text.

c. Modify the value of **dsn** to *Customer* and **var** to *Customer*. Resubmit the program.

Challenge

10. Macro Variable References with Multiple Delimiters

a. Open the **m102e10** program shown below into the Editor window. This program analyzes the **orion.staff** data to find the employee with the most seniority within a job title. Submit the program and examine the output that it creates.

```
proc sort data=orion.staff out=staffhires;
   by Job_Title Emp_Hire_Date;
run;
data FirstHired;
   set staffhires;
   by Job_Title;
   if First.Job_Title;
run;
proc print data=FirstHired;
   id Job_Title;
   var Employee_ID Emp_Hire_Date;
   title "First Employee Hired within Each Job Title";
run;
```

b. Create a macro variable that acts as a toggle such that the program selects employees with either the most seniority or the least seniority within their job title. Make sure that the title reflects this difference.

Partial PROC PRINT Output

First Employee Hired within Each Job Title

Job_Title	Employee_ID	Emp_Hire_ Date
Account Manager	120746	01APR2002
Accountant I	120752	01AUG1975
Accountant II	120771	01DEC1976
Accountant III	120757	01JAN1974
Administration Manager	120104	01JAN1981
Applications Developer I	120797	01DEC1977
Applications Developer II	120796	01MAR1983
Applications Developer IV	120802	01JAN1978

Partial PROC PRINT Output

Last Employee Hired within Each Job Title

Job_Title	Employee_ID	Emp_Hire_ Date
Account Manager	120746	01APR2002
Accountant I	120761	01JUL2006
Accountant II	120754	01MAY2006
Accountant III	120755	01AUG1983
Administration Manager	121000	01DEC1993
Applications Developer I	120801	01JUL1999
Applications Developer II	120812	01AUG2001
Applications Developer IV	120794	01JUL2003

2.6 Macro Functions

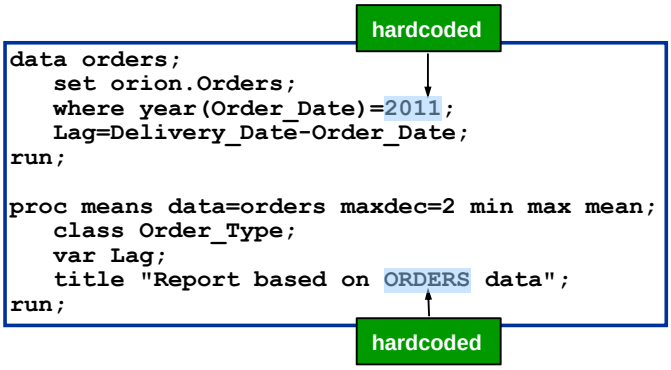
Objectives

- Use macro language functions to manipulate text.
- Use macro language functions to perform arithmetic operations.
- Use macro language functions to execute SAS functions.

116

Business Scenario

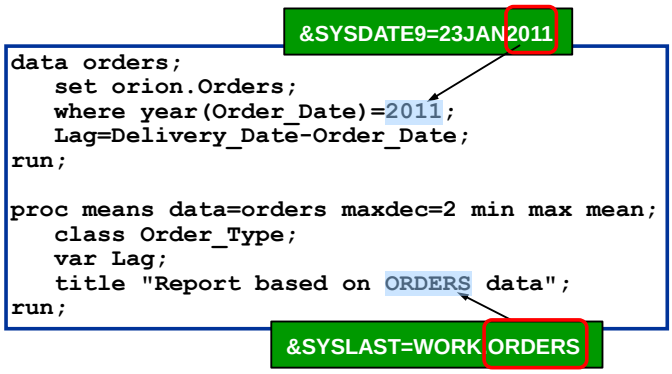
The production program below requires manual entry of the year and data set name. How do you eliminate manual data entry?



117

Task Requirements

Extract the necessary information from automatic macro variables.



118

2.09 Short Answer Poll

In a DATA step, what function would you use to extract the year portion of a character date?

```
data new;
   date='23JAN2011';
   year=???;
run;
```

119

Macro Functions

Macro functions manipulate macro variables.

SAS function

```
substr(date,6)
```

Macro function

```
%substr(&sysdate9,6)
```

- Macro functions are executed by the macro processor.

121

Macro Functions

Selected character string manipulation functions:

%UPCASE	Translates letters from lowercase to uppercase.
%SUBSTR	Extracts a substring from a character string.
%SCAN	Extracts a word from a character string.
%INDEX	Searches a character string for specified text.

Other macro functions:

%EVAL	Performs arithmetic and logical operations.
%SYSFUNC	Executes SAS functions.
%STR	Masks special characters.
%NRSTR	Masks special characters, including macro triggers.

122

%SUBSTR Function

Use the %SUBSTR function to extract the year portion of the **&SYSDATE9** macro variable.

```
17   %put &=sysdate9;
SYSDATE9=20APR2011
18   %put YEAR=%substr(&sysdate9,6);
YEAR=2011
```

%SUBSTR(*argument*, *position* <,*n*>)

- The %SUBSTR function returns the portion of *argument* beginning at *position* for a length of *n* characters.
- When *n* is not supplied, the %SUBSTR function returns the portion of *argument* beginning at *position* to the end of *argument*.

123

The values of *position* and *n* can also be the result of an arithmetic expression that yields an integer. For example,

```
%substr(&var,%length(&var)-1)
```

returns the last two characters of the value of the macro variable **var**.

Macro Functions

Arguments to macro string manipulation functions can be any text or macro triggers, or both, including:

Constant text	%put %upcase(angel);	ANGEL
Macro variable references	%let name=angel; %put %upcase(&name);	ANGEL
Macro functions	%put %upcase(%substr(&name,1,1));	A
Macro calls	%put %upcase(%mymacro);	unknown

Constant text arguments do not require quotation marks.

124

2.10 Multiple Choice Poll

What is the value of **x** after %LET statement execution?

a. A
b. B
c. C
d. D

```
%let x=%substr("ABCD",2,1);
```

125

%SCAN Function

To extract the data set name from the **&SYSLAST** macro variable, use the %SCAN macro function.

SAS Log

```
26   %put &=syslast;
SYSLAST=WORK.ORDERS
27   %put DSN=%scan(&syslast,2,.);
DSN=ORDERS
```

%SCAN(*argument*, *n* <,*delimiters*>)

The %SCAN function does the following:

- returns the *n*th word of *argument*, where words are strings of characters separated by delimiters
- uses a default set of delimiters if none are specified
- returns a null string if there are fewer than *n* words in *argument*

127

The following are default delimiters for the %SCAN function: **blank . (& ! $ *) ; - / , %**

It is not necessary to place *argument* and *delimiters* in quotation marks because they are always handled as character strings by the %SCAN function.

Business Scenario Solution

Step 1 Write and debug a program with hardcoded constants.

```
data orders;
   set orion.Orders;
   where year(Order_Date)=2011;
   Lag=Delivery_Date - Order_Date;
run;

proc means data=orders maxdec=2 min max mean;
   class Order_Type;
   var Lag;
   title "Report based on ORDERS data";
run;
```

m102d05a

128

Business Scenario Solution

Step 2 Use macro functions to extract the necessary information from automatic macro variables.

```
data orders;
   set orion.Orders;
   where year(Order_Date)=%substr(&sysdate9,6);
   Lag=Delivery_Date-Order_Date;
run;

proc means data=&syslast maxdec=2 min max mean;
   class Order_Type;
   var Lag;
   title "Report based on %scan(&syslast,2,.) data";
run;
```

Report based on ORDERS data

Analysis Variable : Lag

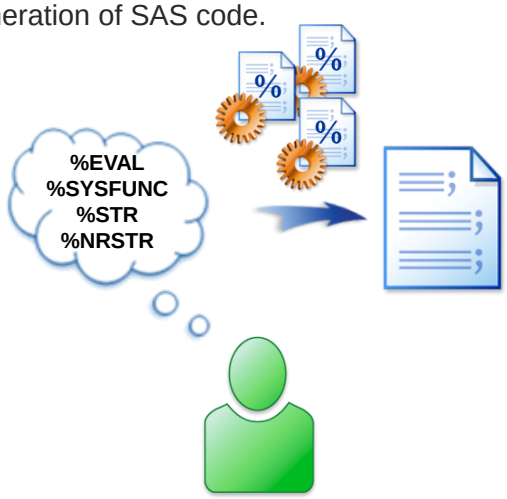
Order Type	N Obs	Minimum	Maximum	Mean
1	70	0.00	10.00	0.43
2	15	2.00	10.00	4.27
3	31	1.00	7.00	3.87

m102d05b

129

Business Scenario

Additional macro functions are available to automate the generation of SAS code.



130

%EVAL Function

The %EVAL function performs arithmetic and logical operations.

```
28   %put x=2+2;
x=2+2
29   %put x=%eval(2+2);
x=4
```

%EVAL(*expression*)

The %EVAL function does the following:

- truncates non-integer results
- returns a text result
- returns 1 (true) or 0 (false) for logical operations
- returns a null value and error message when non-integer values are used in arithmetic operations

131

%EVAL Function

Example: Compute the first year of a range based on the current date.

```
%let thisyr=%substr(&sysdate9,6);
%let lastyr=%eval(&thisyr-1);
proc means data=orion.order_fact maxdec=2 min max mean;
   class order_type;
   var total_retail_price;
   where year(order_date) between &lastyr and &thisyr;
   title1 "Orders for &lastyr and &thisyr";
   title2 "(as of &sysdate9)";
run;
```

Orders for 2010 and 2011
(as of 31DEC2011)

The MEANS Procedure

Analysis Variable : Total Retail Price for This Product

Order Type	N Obs	Minimum	Maximum	Mean
1	174	3.20	1136.20	126.74
2	62	6.20	1937.20	212.88
3	55	9.60	702.00	172.10

132 m102d06

%SYSFUNC Function

The %SYSFUNC macro function executes SAS functions.

```
31   %put &=syslast;
SYSLAST=WORK.ORDERS
32   %put DSN=%sysfunc(propcase(&syslast));
DSN=Work.Orders
```

%SYSFUNC(*SAS function*(*argument(s)*) <,*format*>)

- *SAS function(argument(s))* is the name of a SAS function and its corresponding arguments.
- The second argument is an optional format for the value returned by the first argument.

133

Most SAS functions can be used with %SYSFUNC. Exceptions include array processing (DIM, HBOUND, LBOUND), variable information (VNAME, VLABEL, MISSING), macro interface (RESOLVE, SYMGET), and data conversion functions (INPUT, PUT), as well as the IORCMSG, LAG, and DIF functions.

Because %SYSFUNC is a macro function, do not enclose character arguments in quotation marks as you do with SAS functions. Use commas to separate all arguments in SAS functions within %SYSFUNC. You ***cannot*** use argument lists preceded by the word OF.

%SYSFUNC executes one SAS function at a time. If you want to execute multiple SAS functions in a single macro statement or expression, use a separate %SYSFUNC for each SAS function, as shown below.

```
%put %sysfunc(year(%sysfunc(today())));
```

%SYSFUNC Function

The automatic macro variables **SYSDATE9** and **SYSTIME** can be used in titles.

```
title1 "&sysdate9";
title2 "&systime";
```

```
07MAR2011
13:39
```

SYSDATE9 and **SYSTIME** represent the date and time that the SAS session began.

134 m102d07

%SYSFUNC Function

Example: Generate titles that contain the current date and time, appropriately formatted.

```
title1 "%sysfunc(today(),weekdate.)";
title2 "%sysfunc(time(),timeAMPM8.)";
```

```
Monday, March 7, 2011
1:39 PM
```

135 m102d07

2.11 Short Answer Poll

What is the problem with this attempt to store a complete SAS statement in a macro variable?

```
%let statement=title "S&P 500";
```

136 m102d08

%STR Function

Example: Store a SAS statement in a macro variable.

```
%let statement=%str(title "S&P 500";);
```

%STR(*argument*)

138 m102d08

%STR Function

The %STR function *masks* (removes the normal meaning of) these special tokens:

Special Characters	**+ - * / , < > = ; ' "** **~ ^ ( ) # blank**
Mnemonics	**LT EQ GT LE GE NE AND** **OR NOT IN**

139

The %STR function does not mask **&** and **%** characters.

%STR Function

The following are true for the %STR function:

- masks tokens, so the macro processor does not interpret them as macro-level syntax
- enables macro triggers to work normally
- preserves leading and trailing blanks in its argument
- masks an unpaired quotation mark or parenthesis in its argument when the quotation mark or parenthesis is immediately preceded by a percent sign (%)

140

%STR Function

The %STR function enables macro triggers to work normally.

SAS Log

```
3    %let statement=%str(title "S&P 500";);
WARNING: Apparent symbolic reference P not resolved.
```

m102d08

141

%NRSTR Function

The %NRSTR function works the same as the %STR function, except it also masks macro triggers.

Example: Use %NRSTR to prevent attempted macro variable resolution.

```
%let statement=%nrstr(title "S&P 500";);
```

%NRSTR(argument)

m102d08

142

In addition to %STR and %NRSTR, several other macro quoting functions are available for specialized purposes. For further information, see “Macro Quoting” under “Understanding and Using the Macro Facility” under *SAS Macro Language: Reference* in Base SAS documentation.

2.12 Multiple Answer Poll

You can use macro functions to do which of the following:

a. manipulate text
b. execute SAS functions
c. manipulate SAS data set variables
d. perform arithmetic calculations

143

Exercises

Level 1

11. Using the %SUBSTR and %SCAN Functions

a. Submit a %LET statement to assign the value *Anthony Miller* to a macro variable named **fullname**.

b. Extract the first initial and last name, putting them together into a new macro variable as *A. Miller*. Use the %PUT statement to display the results.

12. Using the %SYSFUNC Function

Use the %PUT statement and the %SYSFUNC function to display the current date and time. Format the date with the MMDDYYP10. format and the time with the TIMEAMPM. format.

Level 2

13. Using Macro Functions

a. Open the **m102e13** program shown below into the Editor window. Submit the program and examine the output that it creates. Verify the titles.

```
%let d=&sysdate9;
%let t=&systime;
proc print data=orion.product_dim;
   where Product_Name contains "Jacket";
   var Product_Name Product_ID Supplier_Name;
   title1 "Product Names Containing 'Jacket'";
   title2 "Report produced &t &d";
run;
```

b. Submit a %LET statement to assign the value *R&D* to a macro variable named **product**. Reference the new macro variable in the WHERE statement and the TITLE1 statement. Submit the modified program.

PROC PRINT Output

```
                          Product Names Containing 'R&D'
                         Report produced 14:06 17FEB2009

          Obs    Product_Name                        Product_ID    Supplier_Name

          393    Top Men's R&D Ultimate Jacket    240300300070    Top Sports Inc
          395    Top R&D Long Jacket              240300300090    Top Sports Inc
```

Challenge

14. Manipulating Macro Variables

a. Assign your birthdate to a macro variable.

b. Issue a %PUT statement that writes the day of the week that you were born.

2.7 Solutions

Solutions to Exercises

1. Displaying Automatic Macro Variables

a. The _AUTOMATIC_ argument in the %PUT statement displays the values of all automatic macro variables in the SAS log. Many of the values shown are dependent on the host system.

```
%put _automatic_;
```

b. The value of **SYSLAST** is _NULL_ unless a data set has been created. Then it is the name of the last created data set. The value of **SYSUSERID** is dependent on the host system. The value of **SYSTIME** and **SYSDATE9** is the initialization time of your SAS session.

c. The values of **SYSTIME** and **SYSDATE** do not reflect the current time and date. Instead, they reflect the initialization time of the SAS session. Therefore, the values are not always accurate.

2. Using Automatic Macro Variables

a. Using the SORT procedure, sort the data set **orion.continent** by **Continent_Name**.

```
proc sort data=orion.continent out=sorted;
   by Continent_Name;
run;
```

b. The **SYSLAST** automatic macro variable contains the name of the most recently created data set.

```
proc print data=&syslast;
   title "&syslast";
run;
```

c. Submit the program and examine the results.

SAS Output

WORK.SORTED

Obs	Continent_ ID	Continent_Name
1	94	Africa
2	95	Asia
3	96	Australia/Pacific
4	93	Europe
5	91	North America

3. Using Automatic Macro Variables

a. The DATA step alters the value of **SYSLAST** to *WORK.NEW*.

```
data new;
   set orion.continent;
run;
```

b. The value of **SYSLAST** is still *WORK.NEW*. The PRINT procedure does not create a SAS data set. Therefore, it does not alter the value of **SYSLAST**.

```
proc print data=orion.continent;
run;
```

4. Using SAS Date Constants

a. Open the program into the Editor window.

b. Modify the program so that it subsets the data to return only the employees hired between January 1, 2007, and today. Use the automatic macro variable **SYSDATE9** to return today's date.

```
proc print data=orion.employee_payroll;
   format Birth_Date Employee_Hire_Date date9.;
   where Employee_Hire_Date between '01jan2007'd and "&sysdate"d;
run;
```

5. Defining and Using Macro Variables for Character Substitution

a. Submit the program below.

```
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "Gold";
   title "Gold Customers";
run;
```

Partial SAS Output

Gold Customers

Obs	Customer_Name	Customer_ Gender	Customer_ Age
2	Sandrina Stephano	F	28
3	Cornelia Krahl	F	33
7	Markus Sepke	M	19
11	Oliver S. Füßling	M	43

b. The macro variable **type** should contain the text string *Gold* without any surrounding quotation marks. To resolve the macro variable in the WHERE and TITLE statements, change the single quotation marks to double quotation marks. It produces the same output as before.

```
%let type=Gold;
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type";
   title "&type Customers";
run;
```

c. Turn on the SYMBOLGEN option.

```
options symbolgen;
%let type=Gold;
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type";
   title "&type Customers";
run;
```

Partial SAS Log

```
79   proc print data=orion.customer_dim;
80        var customer_name customer_gender customer_age;
81        where customer_group contains "&type";
SYMBOLGEN:  Macro variable TYPE resolves to Gold
SYMBOLGEN:  Macro variable TYPE resolves to Gold
82        title "&type Customers";
83   run;
```

d. Modify the value of **type** to *Internet*.

```
%let type=Internet;
proc print data=orion.customer_dim;
   var Customer_name Customer_Gender Customer_age;
   where Customer_Group contains "&type";
   title "&type Customers";
run;
```

Partial SAS Log

```
63   %let type=Internet;
64   proc print data=orion.customer_dim;
65      var customer_name customer_gender customer_age;
66      where customer_group contains "&type";
SYMBOLGEN:  Macro variable TYPE resolves to Internet
SYMBOLGEN:  Macro variable TYPE resolves to Internet
67      title "&type Customers";
68   run;

NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Internet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           5.04 seconds
      cpu time            0.01 seconds
```

e. Turn off the SYMBOLGEN option.

```
options nosymbolgen;
```

6. Defining and Using Macro Variables for Numeric Substitution

a. Edit the program to also subset the data for customers between the ages of 30 and 45.

```
%let type=Gold;
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type" and
         Customer_Age between 30 and 45;
   title "&type Customers between 30 and 45";
run;
```

b. Modify the program so that the values *30* and *45* are replaced by references to the macro variables **age1** and **age2**, respectively.

```
%let type=Gold;
%let age1=30;
%let age2=45;
proc print data=orion.customer_dim;
   var Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type" and
         Customer_Age between &age1 and &age2 ;
   title "&type Customers between &age1 and &age2";
run;
```

c. Include the appropriate system option to display resolved values of macro variables in the SAS log. Resubmit the program and examine the log.

```
options symbolgen;
```

d. Modify the value of **age1** and **age2**. Resubmit the program and examine the log.

Partial SAS Log

```
88   %let type=Gold;
89   %let age1=20;
90   %let age2=30;
91   proc print data=orion.customer_dim;
92      var customer_name customer_gender customer_age;
93      where customer_group contains "&type" and Customer_Age between &age1 and &age2 ;
SYMBOLGEN:  Macro variable TYPE resolves to Gold
SYMBOLGEN:  Macro variable AGE1 resolves to 20
SYMBOLGEN:  Macro variable AGE2 resolves to 30
SYMBOLGEN:  Macro variable TYPE resolves to Gold
SYMBOLGEN:  Macro variable AGE1 resolves to 20
SYMBOLGEN:  Macro variable AGE2 resolves to 30
94      title "&type Customers between &age1 and &age2";
95   run;

NOTE: There were 7 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Gold' and (Customer_Age>=20 and Customer_Age<=30);
```

e. Turn off the SYMBOLGEN option.

```
options nosymbolgen;
```

7. Deleting Macro Variables

a. Open the program and submit the code. %PUT _USER_ lists all user-defined macro variables in the SAS log.

```
%put _user_;
```

b. The %SYMDEL statement deletes user-defined macro variables.

```
%symdel type age1 age2 pet1 pet2;
```

Your solution could be different. It depends on the macro variables created during the current SAS session.

c. Use the %PUT statement to verify that the macro variable deletion is successful.

```
%put _user_;
```

8. Consecutive Macro Variable References

a. Open the program into the Editor window.

b. Modify the program so that the two occurrences of *AUG* and *2006* are replaced by references to the macro variables **month** and **year**, respectively.

```
%let month=AUG;
%let year=2006;
proc print data=orion.organization_dim;
   where Employee_Hire_Date="01&month&year"d;
   id Employee_ID;
   var Employee_Name Employee_Country Employee_Hire_Date;
   title "Personal Information for Employees Hired in "
         "&month &year";
run;
```

c. Modify the value of **month** to *JUL* and **year** to *2003*. Resubmit the program.

```
%let month=JUL;
%let year=2003;
proc print data=orion.organization_dim;
   where Employee_Hire_Date="01&month&year"d;
   id Employee_ID;
   var Employee_Name Employee_Country Employee_Hire_Date;
   title "Personal Information for Employees Hired in "
         "&month &year";
run;
```

9. Macro Variable References with Delimiters

a. Open the program into the Editor window.

b. Modify the program so that all occurrences of *Organization* and *Employee* are replaced with macro variable references called **dsn** and **var**, respectively. Submit the program and verify the output.

```
%let dsn=Organization;
%let var=Employee;
proc print data=orion.&dsn._dim;
   id &var._ID;
   var &var._Name &var._Country &var._Gender;
```

```
   title "Listing of All &var.s From Orion.&dsn._Dim";
run;
```

c. Modify the value of **dsn** to *Customer* and **var** to *Customer*. Resubmit the program

```
%let dsn=Customer;
%let var=Customer;
proc print data=orion.&dsn._dim;
   id &var._ID;
   var &var._Name &var._Country &var._Gender;
   title "Listing of All &var.s From Orion.&dsn._Dim";
run;
```

10. Macro Variable References with Multiple Delimiters

a. Open the program into the Editor window.

b. Using a macro variable, modify the program to return the employees with the most or least amount of seniority.

```
proc sort data=orion.staff out=staffhires;
   by Job_Title Emp_Hire_Date;
run;

%let seniority=First; /* Employees with most seniority */
/* %let seniority=Last; Employees with least seniority */

data &seniority.Hired;
   set staffhires;
   by Job_Title;
   if &seniority..Job_Title;
run;
proc print data=&seniority.Hired;
   id Job_Title;
   var Employee_ID Emp_Hire_Date;
   title "&seniority Employee Hired within Each Job Title";
run;
```

11. Using the %SUBSTR and %SCAN Functions

a. Submit a %LET statement to assign the value *Anthony Miller* to a macro variable named **fullname**.

```
%let fullname=Anthony Miller;
```

b. Extract the first initial and last name, putting them together into a new variable as *A. Miller*.

```
%let newname=%substr(&fullname,1,1). %scan(&fullname,2);
%put &newname;
```

Alternate Solution

```
%let initial=%substr(&fullname,1,1);
%let last=%scan(&fullname,2);
%let newname=&initial.. &last;
%put &newname;
```

12. Using the %SYSFUNC Function

Use the %PUT statement and the %SYSFUNC function to display the current date and time. Format the date with the MMDDYYP10. format and the time with the TIMEAMPM. format.

```
%put Today is %sysfunc(today(), mmddyyp10.) and time is
     %sysfunc(time(), timeampm.);
```

13. Using Macro Functions

a. Open the program into the Editor window.

b. The %NRSTR function is required to prevent macro variable resolution.

```
%let d=&sysdate9;
%let t=&systime;
%let product=%nrstr(R&D);
proc print data=orion.product_dim;
   where Product_Name contains "&product";
   var Product_Name Product_ID Supplier_Name;
   title1 "Product Names Containing '&product'";
   title2 "Report produced &t &d";
run;
```

14. Manipulating Macro Variables

a. Assign your birthdate to a macro variable.

```
%let birthdate=06jun1982;
```

b. Issue a %PUT statement that writes the day of the week that you were born.

```
%put %sysfunc(putn("&birthdate"d,downame.));
```

Solutions to Student Activities (Polls/Quizzes)

2.01 Short Answer Poll – Correct Answer

What are the two types of macro variables?

automatic and user-defined

11

2.02 Poll – Correct Answer

Submit the statement below.

```
%put _automatic_;
```

Does **SYSTIME** match the current time?

- ❍ Yes
- ❍ No

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

21

2.03 Short Answer Poll – Correct Answer

SAS Output

```
          Customer Country

Country     Frequency      Percent
-------------------------------------
AU                  8        10.39
CA                 15        19.48
DE                 10        12.99
IL                  5         6.49
TR                  7         9.09
US                 28        36.36
ZA                  4         5.19

Created &systime &sysday, &sysdate9
```

The word scanner does not tokenize literals enclosed in single quotation marks, so macro variables do not resolve.

38

m102d01

2.04 Multiple Choice Poll – Correct Answer

Macro variable references are resolved by which of the following?

a. SAS compiler
b. macro processor
c. word scanner

50

2.05 Short Answer Poll – Correct Answer

What is the value of the macro variable **z**?

```
%let x=15;
%let y=10;
%let z=&x-&y;
```

The value of z is 15-10.

82

2.06 Multiple Choice Poll – Correct Answer

Which WHERE statement is correct?

a. where City=&office;
b. where City='&office';
c. where City="&office";

91

2.07 Short Answer Poll – Correct Answer

What is the problem with this reference?

Partial SAS Log

```
1  %let graphics=g;
2  proc &graphicschart data=orion.y&year&month;
          -
          10
WARNING: Apparent symbolic reference GRAPHICSCHART not resolved.

ERROR 10-205: Expecting the name of the procedure to be executed.
```

SAS interprets the macro variable's name as GRAPHICSCHART because no delimiter separates the macro variable reference from the trailing text.

107

2.08 Short Answer Poll – Correct Answer

Modify the previous example to include a macro variable that stores a libref.

```
libname orion "&path";
proc gchart data=oriony2010jan;
   hbar week / sumvar=sale;
run;
proc gplot data=oriony2010jan;
   plot sale*day;
run;
```

missing period

What is the problem this time?

The period after &LIB is interpreted as a delimiter, so the period does not appear as text.

111

2.09 Short Answer Poll – Correct Answer

In a DATA step, what function would you use to extract the year portion of a character date?

```
data new;
   date='23JAN2011';
   year=substr(date,6);
run;
```

The SUBSTR function extracts selected characters from a character string.

120

2.10 Multiple Choice Poll – Correct Answer

What is the value of **x** after %LET statement execution?

a. A
b. B
c. C
d. D

```
%let x=%substr("ABCD",2,1);
```

126

2.11 Short Answer Poll – Correct Answer

What is the problem with this attempt to store a complete SAS statement in a macro variable?

```
%let statement=title "S&P 500";
```

The semicolon ends the %LET statement.

The stored value lacks a semicolon.

```
title "S&P 500"
```

missing semicolon

137

m102d08

2.12 Multiple Answer Poll – Correct Answers

You can use macro functions to do which of the following:

a. manipulate text
b. execute SAS functions
c. manipulate SAS data set variables
d. perform arithmetic calculations

144

Chapter 3 Macro Definitions

3.1 Defining and Calling a Macro

Objectives

- Define the purpose of a macro definition.
- Specify the steps for creating and storing a macro.
- Identify the steps for calling a macro.
- Investigate the program flow of a macro call.

3

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

4

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 (LET and SCAN, for example).

5 m103d01

Creating and Storing a Macro

Creating a macro requires the following steps:

Step 1 Create the macro definition.

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

Step 2 Submit the macro definition.

6

Creating and Storing a Macro

Submitting a macro definition causes the following actions:

1 The macro is compiled.

Macro language statements or expressions, if any, are

- checked for syntax errors
- compiled.

SAS statements and other text are

- not checked for syntax errors
- not compiled.

2 The macro is stored.

The compiled macro is stored, by default, in the temporary catalog **work.sasmacr**, with an entry type of MACRO.

7

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.

m103d01

8

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

3.01 Poll

Does the macro call below require a semicolon?

%Time

❍ Yes
❍ No

10

Macro Storage

To display a list of compiled macros in a SAS catalog, use the CONTENTS statement in PROC CATALOG.

```
proc catalog cat=work.sasmacr;
   contents;
   title "My Temporary Macros";
quit;
```

PROC CATALOG Output

```
                        My Temporary Macros

                  Contents of Catalog WORK.SASMACR

# Name Type         Create Date       Modified Date Description
-----------------------------------------------------------------
1 TIME MACRO 11JUN2010:15:55:59  11JUN2010:15:55:59
```

12 m103d02

Permanent Macro Storage

Storing a macro permanently requires two steps.

Step 1 Set the appropriate system options.

```
options mstored sasmstore=orion;
```

MSTORED enables storage of compiled macros in a permanent library.

SASMSTORE= designates a permanent library to store compiled macros.

Step 2 Specify the STORE and SOURCE options in the %MACRO statement.

```
%macro time / store source;
```

The SOURCE option stores the macro source code with the compiled code.

13

The following code copies macro source code saved with the SOURCE option to the SAS log:

```
%copy time / source;
```

Permanent Macro Storage

Store the **Time** macro permanently in the **orion** library.

```
libname orion "&path";
options mstored sasmstore=orion;
%macro time / store source;
   %put The current time is %sysfunc
        (time(),timeampm.).;
%mend time;
```

Call the **Time** macro in a ***new*** SAS session.

```
libname orion "&path";
options mstored sasmstore=orion;
%time
```

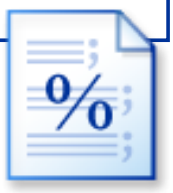
14 m103d03

The macro processor searches the **work.sasmacr** catalog first, followed by the SASMSTORE= catalog.

Business Scenario

To minimize typing, create the **Calc** macro below.

```
%macro calc;
   proc means data=orion.order_fact;
   run;
%mend calc;
```



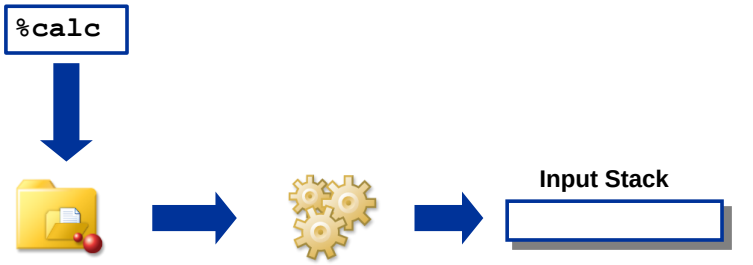
The **Calc** macro

- generates complete SAS statements and a complete SAS step
- contains no macro language statements.

15 m103d04a

General Program Flow

When the macro processor receives a macro call, it takes the following actions.



Searches the macro catalog for CALC.MACRO

Executes compiled macro language statements, if any

Sends SAS code and other text to the input stack for word scanning

16

continued...

General Program Flow

Pause

Pauses while the word scanner tokenizes inserted text, and SAS code, if any, compiles and executes

Resume

Resumes execution of macro language statements, if any

17

Defining and Calling a Macro

SAS Log

```
1293 %macro calc;
1294    proc means data=orion.order_fact;
1295    run;
1296 %mend calc;
NOTE: The macro CALC completed compilation without errors.
      5 instructions 124 bytes.
1297
1298 %calc

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set
      ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time      0.02 seconds
      cpu time       0.03 seconds
```

- PROC MEANS source code does not appear in the SAS log.

18 m103d04a

Defining and Calling a Macro

To view the text generated by macro execution, specify the MPRINT option.

SAS Log

OPTIONS MPRINT;

```
1300 options mprint;
1301 %calc
MPRINT(CALC):  proc means data=orion.order_fact;
MPRINT(CALC):  run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set
      ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time      0.02 seconds
      cpu time       0.03 seconds
```

- The default setting is NOMPRINT.

19 m103d04a

Macro-generated code is treated as a series of tokens. The MPRINT option writes each statement to a new line without indention.

Business Scenario

For greater flexibility, revise the **Calc** macro as shown below.

```
%macro calc;
   proc means data=
%mend calc;
```

- This **Calc** macro generates a partial SAS statement and a partial SAS step.

20 m103d04b

Defining and Calling a Macro

Call the revised **Calc** macro, followed by additional code to complete the PROC MEANS statement and step.

```
%macro calc;
   proc means data=
%mend calc;

%calc
orion.order_fact;
run;
```

21 m103d04b

Defining and Calling a Macro

SAS Log

```
1193 options mprint;
1194 %calc
MPRINT(CALC):  proc means data=
1195 orion.order_fact;
1196 run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set
      ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time      0.04 seconds
      cpu time       0.06 seconds
```

22 m103d04b

Defining and Calling a Macro

Add a semicolon after the macro call.

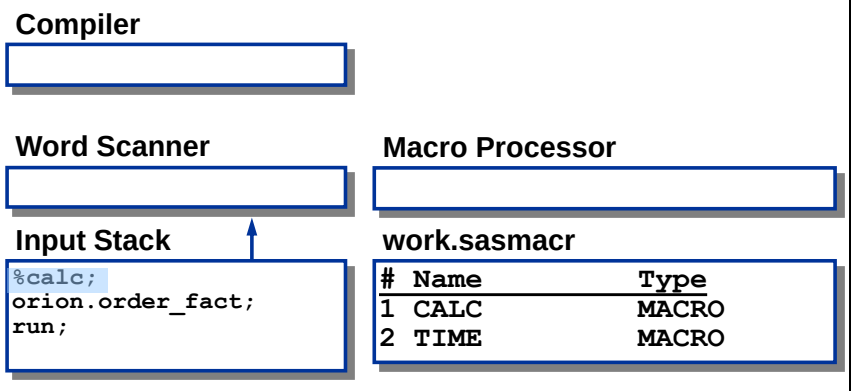
```
%macro calc;
   proc means data=
%mend calc;

%calc;
orion.order_fact;
run;
```

A semicolon following a macro call is not recommended and can cause problems.

23 m103d04b

Detail Program Flow

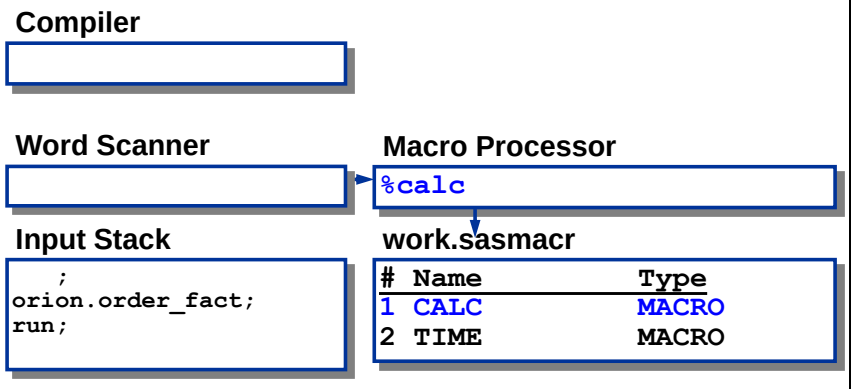
Add a semicolon after the macro call.



24 ...

Detail Program Flow

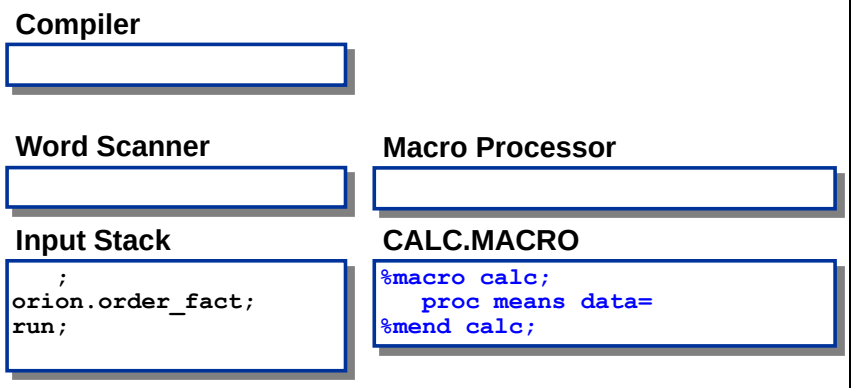
The macro call is passed to the macro processor as usual. The semicolon remains in the input stack.



25 ...

Detail Program Flow

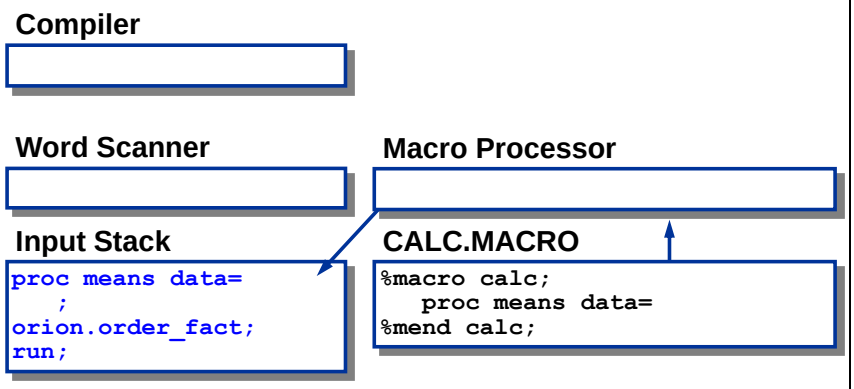
The macro processor opens CALC.MACRO.



26 ...

Detail Program Flow

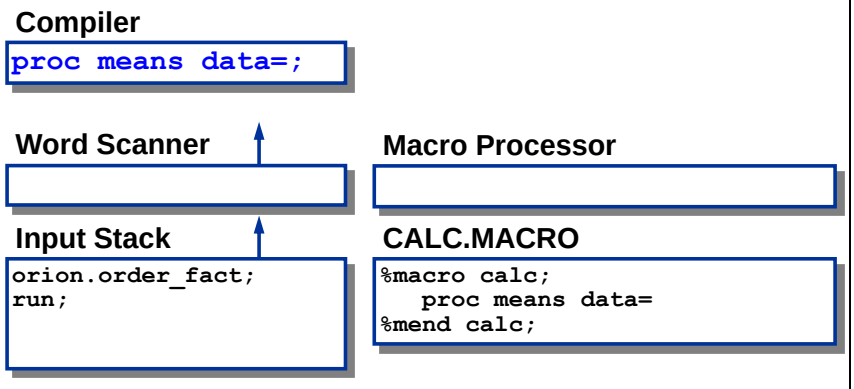
Macro text is placed on the input stack on top of the semicolon.



27 ...

Detail Program Flow

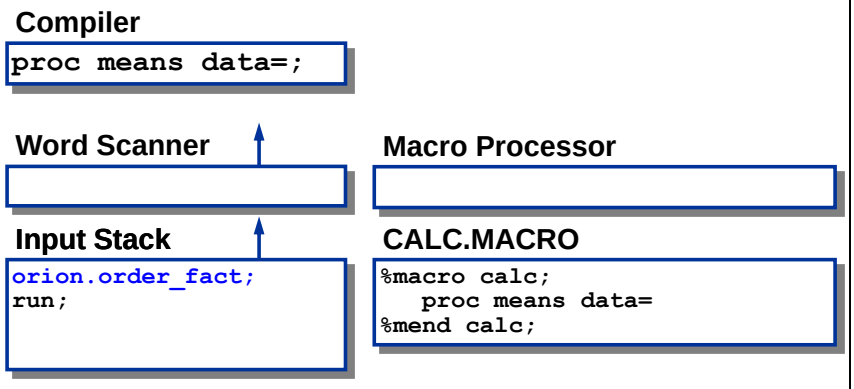
The PROC MEANS statement is passed to the compiler, followed by the semicolon, generating a syntax error.



28 ...

Detail Program Flow

A second syntax error is generated by the subsequent text.



29 ...

Defining and Calling a Macro

SAS Log

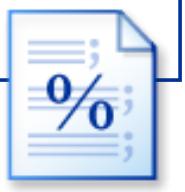
```
1204 %calc;
           -
           22
ERROR 22-322: Expecting a name.

MPRINT(CALC):  proc means data=
1205 orion.order_fact;
ERROR: File WORK.NAME.DATA does not exist.
1205 orion.order_fact;
     ----------------
     180
ERROR 180-322: Statement is not valid or it is used out of proper order.

1206 run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE MEANS used (Total process time):
      real time      0.01 seconds
      cpu time       0.01 seconds
```

30 m103d04b

Business Scenario

Revise the **Calc** macro again to enable easy and flexible selection of the ***data set name*** and ***variable name(s)***.

```
%macro calc;
   proc means data=      ;
      var    ;
   run;
%mend calc;
```



31 m103d04c

Defining the Macro

Use macro variable references within a macro definition for increased flexibility.

```
%macro calc;
   proc means data=&dsn;
      var &vars;
   run;
%mend calc;
```

32 m103d04c

Macro variable references within a macro definition resolve during macro execution, not compilation.

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.

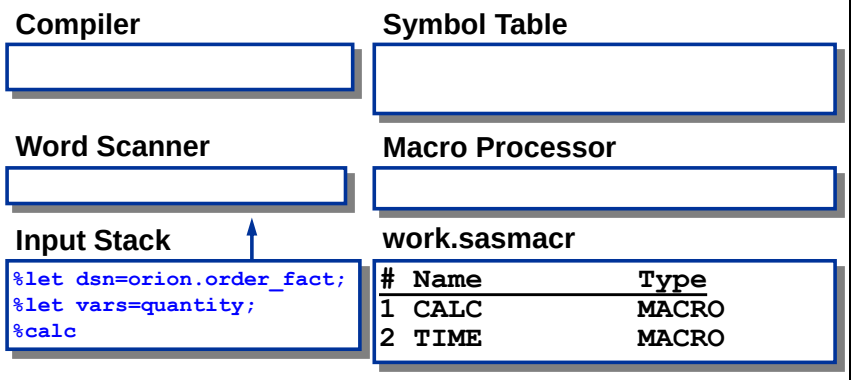
```
%let dsn=orion.order_fact;
%let vars=quantity;
%calc
```

33 m103d04c

Placing a semicolon after a macro call might insert an inappropriate semicolon into the resulting program, leading to errors during compilation or execution.

Detail Program Flow

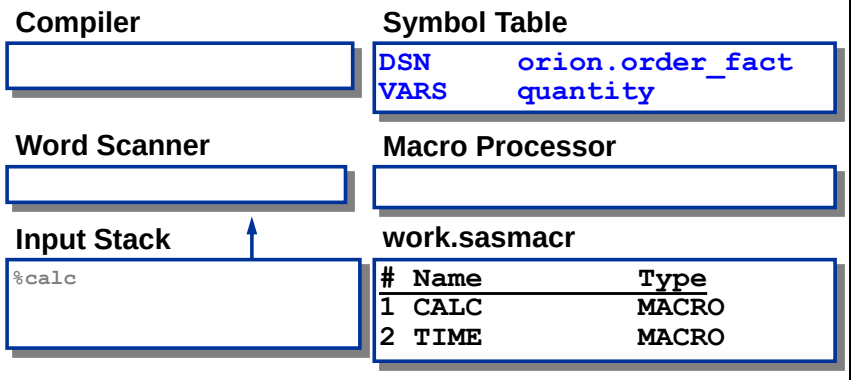
Example: Create macro variables and call the **Calc** macro.



34 ...

Detail Program Flow

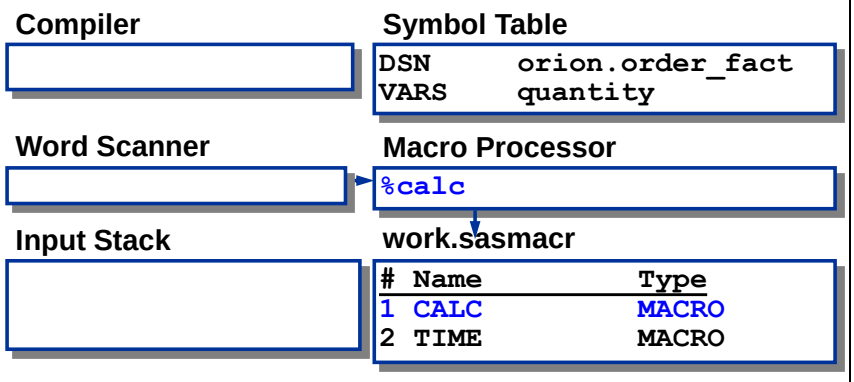
The macro processor executes the %LET statements and populates the symbol table.



35 ...

Detail Program Flow

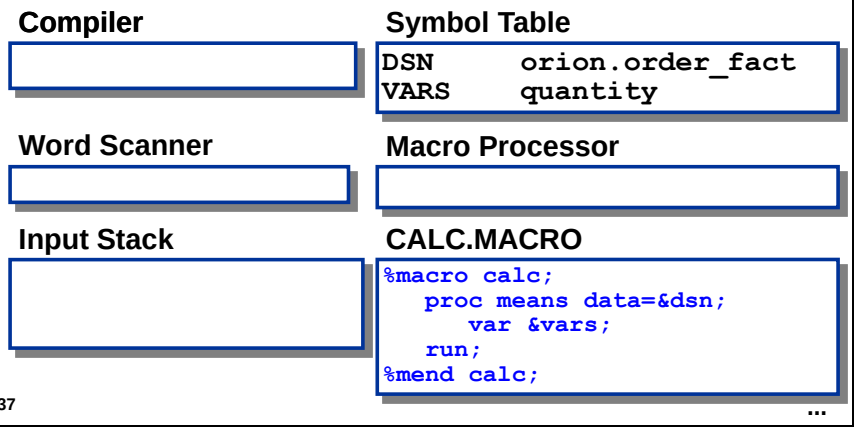
When the macro processor receives **%Calc**, it locates CALC.MACRO within the **work.sasmacr** catalog.



36 ...

Detail Program Flow

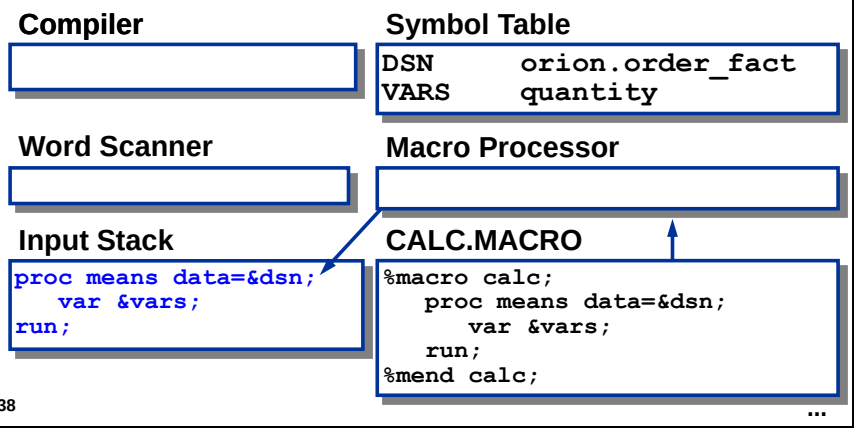
The macro processor opens CALC.MACRO. There are no macro language statements to execute.



37 ...

Detail Program Flow

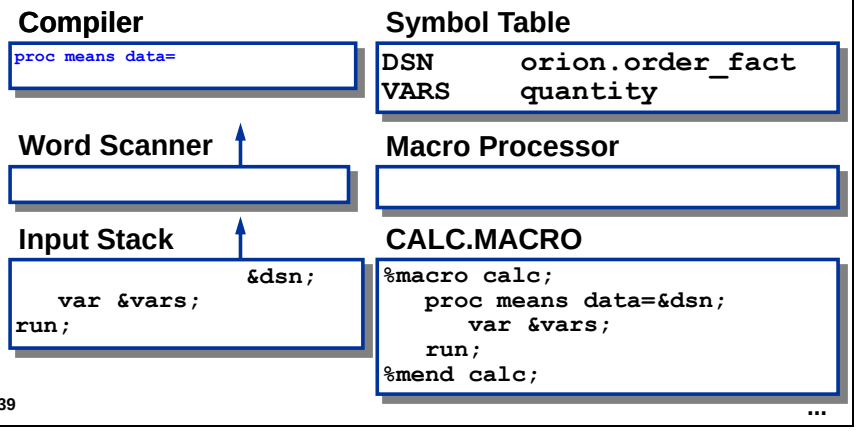
The macro processor places the macro text on the input stack.



38 ...

Detail Program Flow

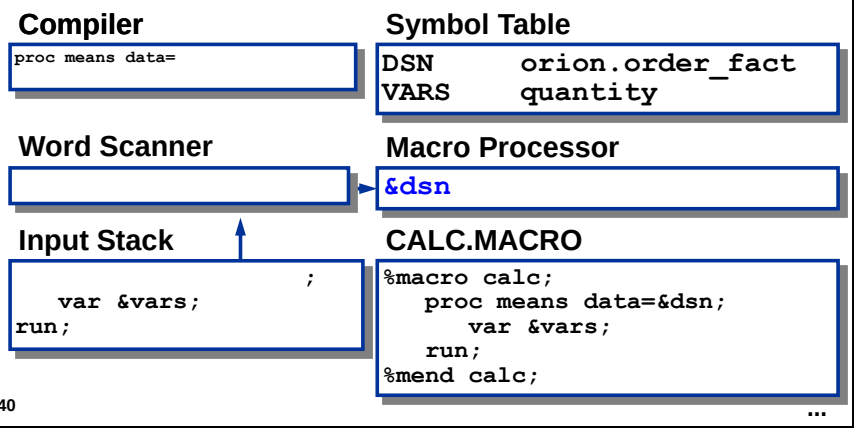
Macro activity pauses while the word scanner tokenizes SAS code and passes tokens to the compiler.



39 ...

Detail Program Flow

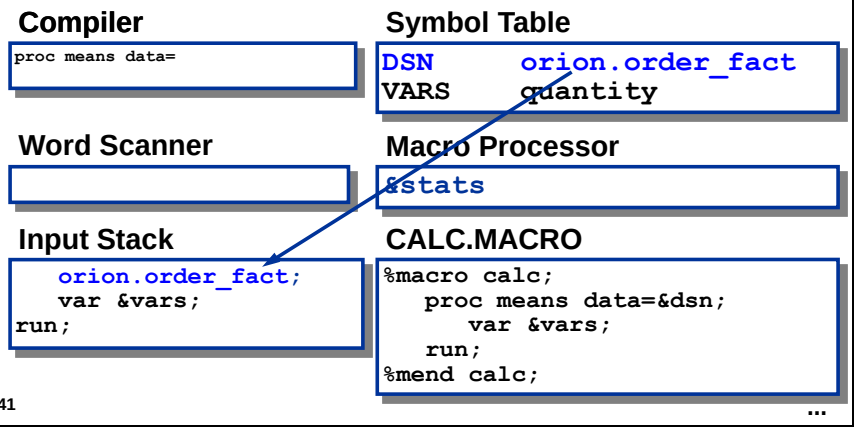
Macro variable references are passed to the macro processor.



40 ...

Detail Program Flow

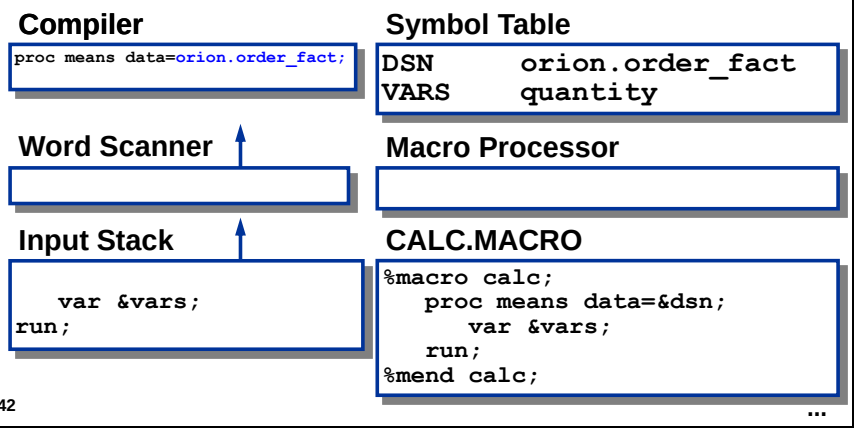
Symbolic substitution is performed.



41 ...

Detail Program Flow

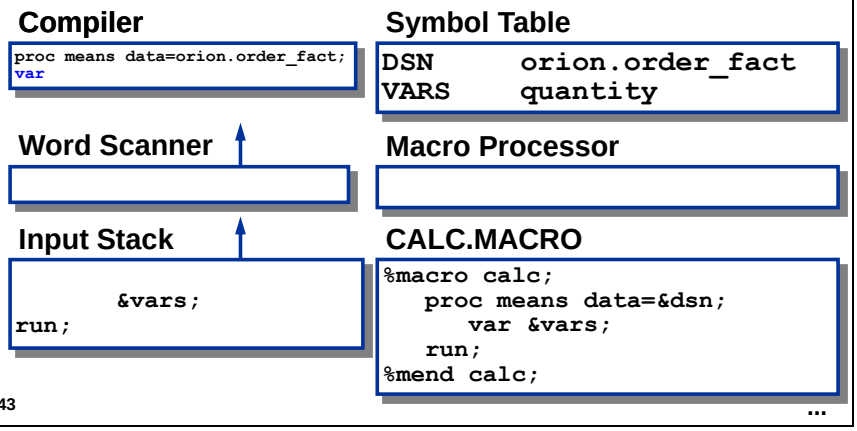
The word scanner tokenizes the resolved value and passes it to the compiler.



42 ...

Detail Program Flow

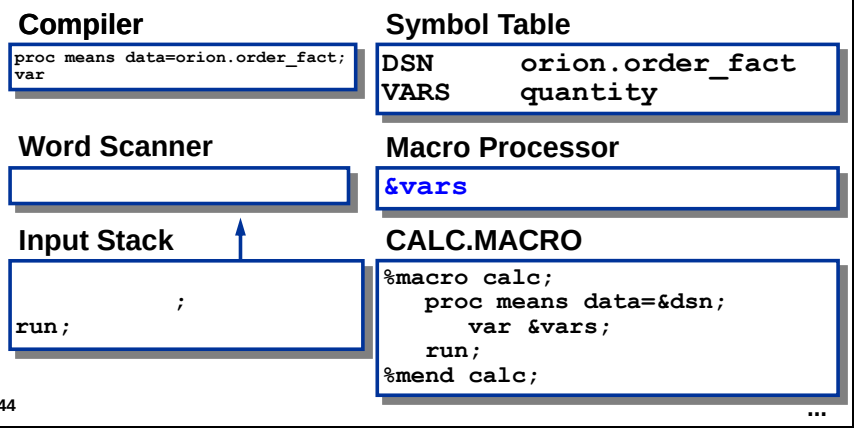
Word scanning continues.



43 ...

Detail Program Flow

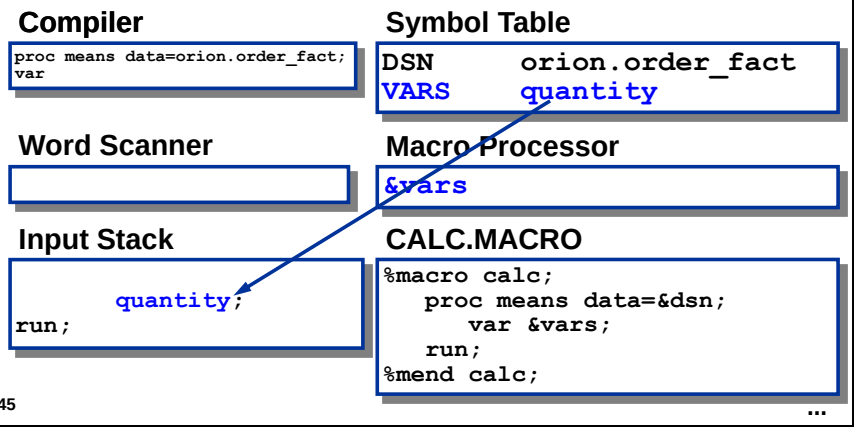
The macro variable reference is passed to the macro processor.



44 ...

Detail Program Flow

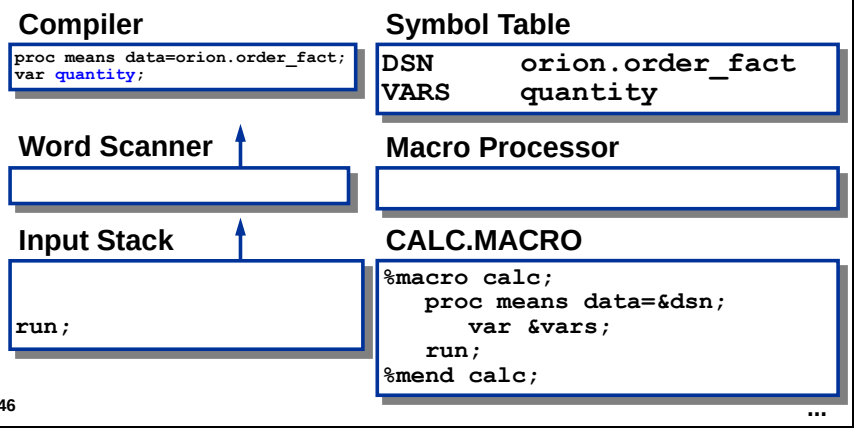
Symbolic substitution is performed.



45 ...

Detail Program Flow

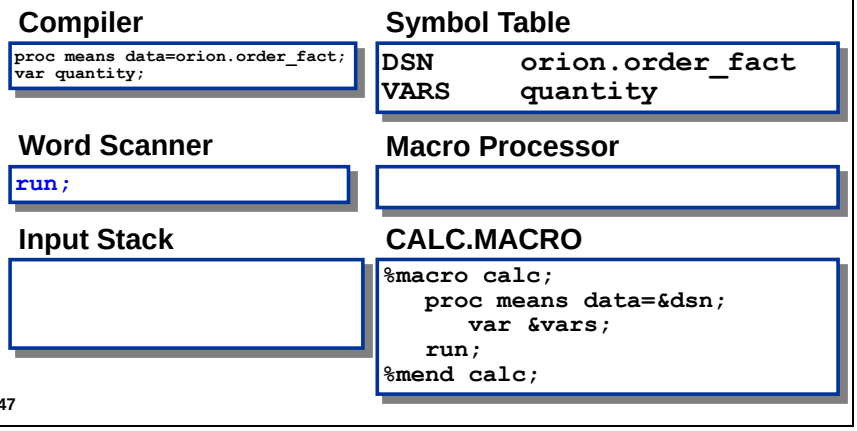
The word scanner tokenizes the resolved value and passes it to the compiler.



46 ...

Detail Program Flow

When a step boundary is encountered, SAS executes the compiled step as macro activity remains paused. Macro activity stops when the %MEND statement is encountered.



47

Macro Execution

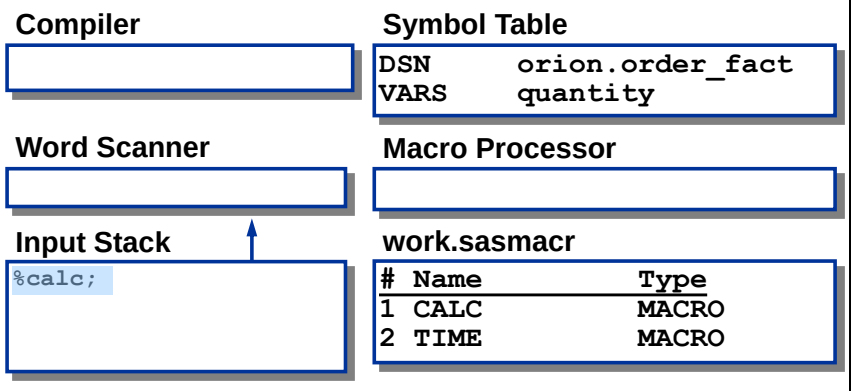
SAS Log

```
52  %let stats=min max;
53  %let vars=quantity;
54  %calc

NOTE: There were 617 observations read from the data set ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time      0.03 seconds
      cpu time       0.03 seconds
```

48 m103d04c

Detail Program Flow

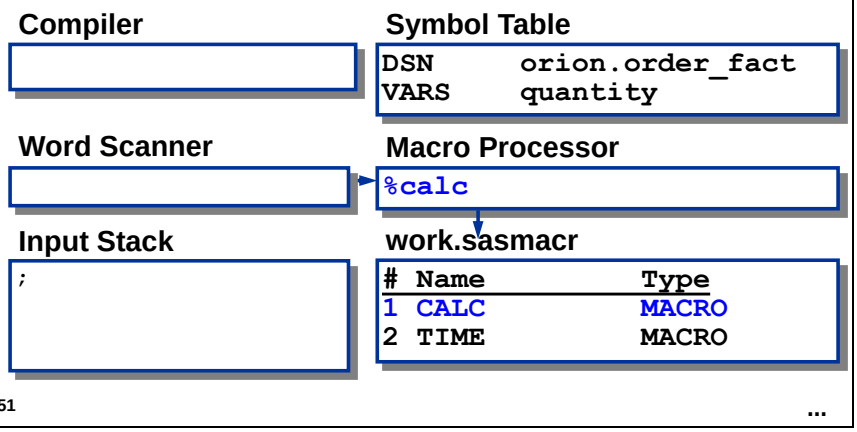
Add a semicolon to the macro call.



50 ...

Detail Program Flow

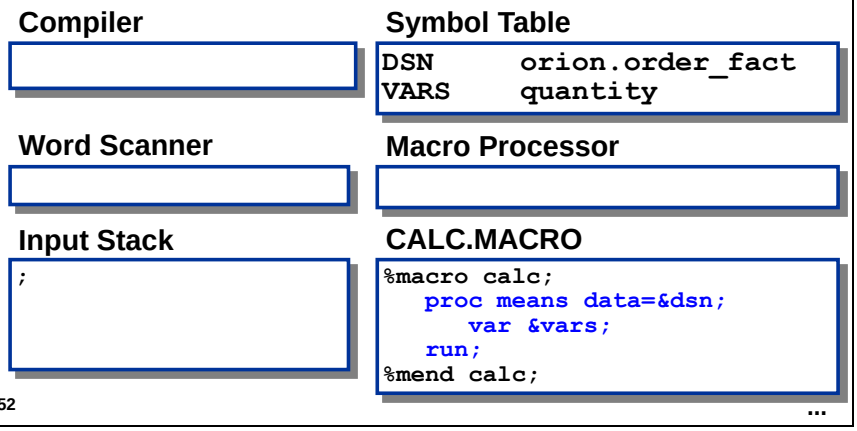
The macro call is passed to the macro processor as usual. The semicolon remains in the input stack.



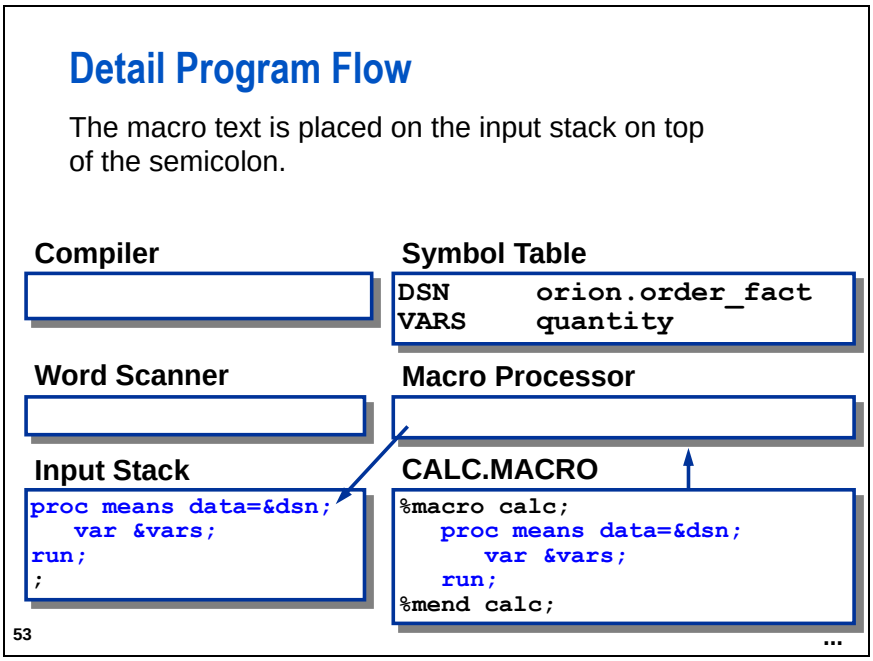
51 ...

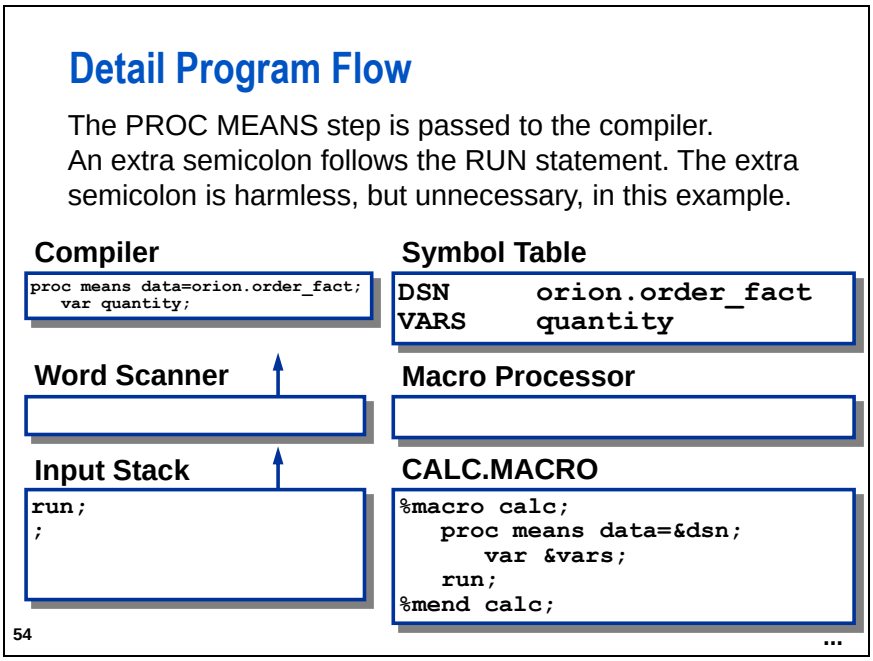
Detail Program Flow

The macro processor opens CALC.MACRO.



52 ...





Exercises

Level 1

1. Defining and Calling a Macro

a. Open the **m103e01** program shown below into the Editor window.

```
proc print data=orion.customer_dim;
   var Customer_Group Customer_Name Customer_Gender Customer_Age;
   where Customer_Group contains "&type";
   title "&type Customers";
run;
```

b. Convert the program into a macro named **Customers**. Set the appropriate system option to display a note in the SAS log when a macro definition has compiled. Submit the macro definition and examine the log.

c. Submit a %LET statement to assign the value *Gold* to the macro variable **type**. Call the macro and examine the log.

d. Change the value of **type** to *Internet*.

e. Activate the appropriate system option to display source code received by the SAS compiler. Call the macro again and examine the log.

Level 2

2. Storing a Macro

a. Open the **m103e02** program shown below into the Editor window.

```
%macro tut;
   king tut
%mend tut;
```

b. Submit the macro definition and check the SAS log.

c. In the SAS Explorer window, navigate to the **sasmacr** catalog within the **work** library to locate the **Tut** macro. In SAS Enterprise Guide, use PROC CATALOG.

d. Submit a %SYSMACDELETE statement to delete the **Tut** macro.

The %SYSMACDELETE statement was introduced in SAS 9.3.

Challenge

3. Calling a Macro from a TITLE Statement

a. Define a macro that issues the current time of day with the TIMEAMPM. format. Name the macro **Currtime**. Submit the macro definition.

b. Open the **m103e03** program shown below into the Editor window. Add a TITLE2 statement. Call the **Currtime** macro from the TITLE2 statement. Submit the program and examine the output.

```
proc print data=orion.customer_dim(obs=10);
   var Customer_Name Customer_Group;
   title 'Customer List';
run;
```

3.2 Macro Parameters

Objectives

- Define and call macros with parameters.
- Describe the difference between positional parameters and keyword parameters.

58

Review

Example: Notice macro variable references within the **Calc** macro.

```
%macro calc;
  proc means data=&dsn;
     var &vars;
  run;
%mend calc;
```

59 m103d04c

Business Scenario

Simplify the **Calc** macro so that it can be called with one line of code instead of three.

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

three lines of code

```
%calc(orion.order_fact,quantity)
```

one line of code

- You can define a macro with a parameter list.

61 m103d04c

Parameter Lists

A *parameter list* is a list of macro variables referenced within the macro. There are three types of parameter lists:

- positional
- keyword
- mixed

62

Positional Parameters

A *positional parameter* list defines macro parameters in a particular order. Parameter ***names*** are supplied when the macro is defined.

```
%macro calc(dsn,vars);
   proc means data=&dsn;
      var &vars;
   run;
%mend calc;
```

%MACRO *macro-name*(*parameter-1, … parameter-n*);
macro text
%MEND <*macro-name*>;

63 m103d05

Positional Parameters

Parameter ***values*** are supplied when the macro is called.

```
%macro calc(dsn,vars);
   proc means data=&dsn;
      var &vars;
   run;
%mend calc;

%calc(orion.order_fact,quantity)
```

%*macro-name*(*value-1*, … *value-n*)

- Parameter values must appear in the same order as their corresponding parameter names.

64 m103d05

To assign a null value to one or more positional parameters, use commas as placeholders for the omitted values.

Local Symbol Tables

When a macro with a parameter list is called, a local symbol table is created and populated with the macro parameters.

```
%calc(orion.order_fact,quantity)
```

Calc Local Table

DSN	orion.order_fact
VARS	quantity

Macro variables in a local table exist only during macro execution and can be referenced only within the macro.

- Local symbol tables are deleted after macro execution.

65

3.02 Multiple Choice Poll

A %LET statement outside a macro definition creates a macro variable in which of the following?

a. global symbol table
b. local symbol table

66

Positional Parameters

Example: Define and call a macro with positional parameters.

```
%macro count(opts, start, stop);
   proc freq data=orion.orders;
      where order_date between
            "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
   run;
%mend count;
options mprint;

%count(nocum,01jan2008,31dec2008)
%count(,01jul2008,31dec2008)
```

68 m103d06a

Macros with Positional Parameters

m103d06a

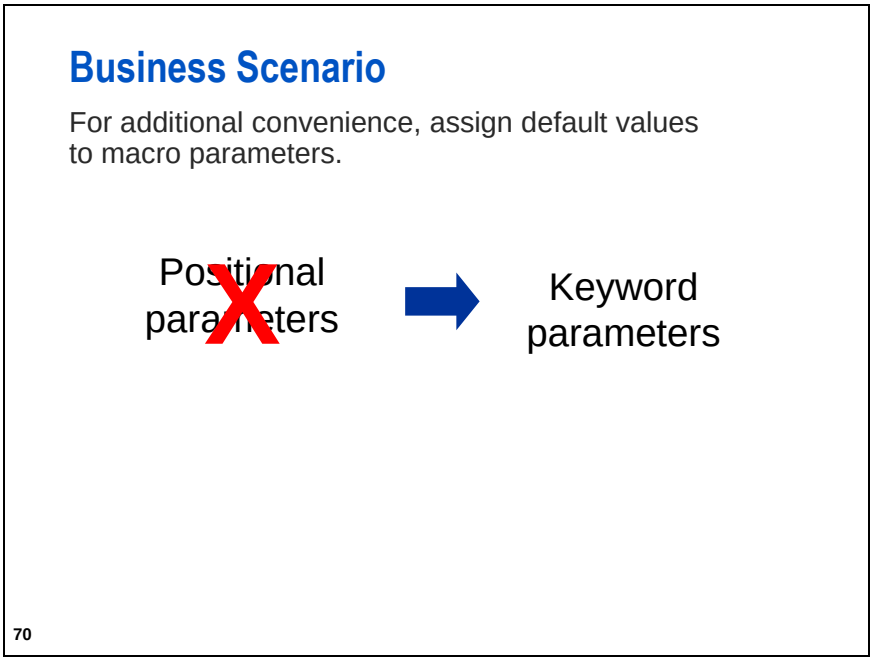
Use positional parameters to specify a range of dates and TABLE statement options for the FREQ procedure.

```
%macro count(opts, start, stop);
   proc freq data=orion.orders;
      where order_date between "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
   run;
%mend count;
options mprint;
%count(nocum,01jan2008,31dec2008)
%count(,01jul2008,31dec2008)
```

A null value is passed to the OPTS parameter in the second call.

Partial SAS Log

```
50   %count(nocum,01jan2008,31dec2008)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan2008"d and "31dec2008"d;
MPRINT(COUNT):   table order_type / nocum;
MPRINT(COUNT):   title1 "Orders from 01jan2008 to 31dec2008";
MPRINT(COUNT):   run;
NOTE: There were 87 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.28 seconds
      cpu time            0.11 seconds

51   %count(,01jul2008,31dec2008)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jul2008"d and "31dec2008"d;
MPRINT(COUNT):   table order_type / ;
MPRINT(COUNT):   title1 "Orders from 01jul2008 to 31dec2008";
MPRINT(COUNT):   run;
NOTE: There were 40 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JUL2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```



Keyword Parameters

Keyword parameters are assigned a default value after an equal sign (=).

```
%macro count(opts=,start=01jan08,stop=31dec08);
   proc freq data=orion.orders;
      where order_date between
            "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
   run;
%mend count;
```

%MACRO *macro-name*(*keyword=value*, …, *keyword=value*);
macro text
%MEND <*macro-name*>;

71 m103d06b

Keyword Parameters

Call a macro with keyword parameters.

```
%macro count(opts=,start=01jan08,stop=31dec08);
   proc freq data=orion.orders;
      where order_date between
            "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
   run;
%mend count;
options mprint;
%count(opts=nocum)
%count(stop=01jul08,opts=nocum nopercent)
%count()
```

%*macro-name*(*keyword=value*, …, *keyword=value*)

Keyword parameters can appear in any order and can be omitted from the call without placeholders. If omitted from the call, a keyword parameter receives its default value.

72 m103d06b

To omit every keyword parameter from a macro call, specify **%*macro-name*()**.

Specifying **%*macro-name*** without the parentheses might not execute the macro immediately.

Macros with Keyword Parameters

m103d06b

Alter the previous macro by using keyword parameters. Issue various calls to the macro.

```
%macro count(opts=,start=01jan08,stop=31dec08);
   proc freq data=orion.orders;
      where order_date between "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
   run;
%mend count;
options mprint;
%count(opts=nocum)
%count(stop=01jul08,opts=nocum nopercent)
%count()
```

Partial SAS Log

```
64   %count(opts=nocum)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "31dec08"d;
MPRINT(COUNT):   table order_type / nocum;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 31dec08";
MPRINT(COUNT):   run;
NOTE: There were 87 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

65   %count(stop=01jul08,opts=nocum nopercent)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "01jul08"d;
MPRINT(COUNT):   table order_type / nocum nopercent;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 01jul08";
MPRINT(COUNT):   run;
NOTE: There were 47 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='01JUL2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

66   %count()
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "31dec08"d;
MPRINT(COUNT):   table order_type / ;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 31dec08";
MPRINT(COUNT):   run;
NOTE: There were 87 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

3.03 Short Answer Poll

Submit program **m103a01**.

```
%macro dog(name=spot);
   %put My dog is &name;
%mend dog;

%dog()
```

Edit the macro call to omit the parentheses.

```
%dog
```

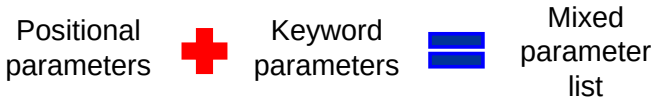
Submit the macro call again.

What do you see in the SAS log?

74

Business Scenario

A parameter list can contain a mix of positional and keyword parameters.



76

Mixed Parameter Lists

In a mixed parameter list, positional parameters are listed before keyword parameters in both the macro definition and the macro call.

```
%macro count(opts,start=01jan08,stop=31dec08);
   proc freq data=orion.orders;
      where order_date between
            "&start"d and "&stop"d;
      table order_type / &opts;
      title1 "Orders from &start to &stop";
  run;
%mend count;
options mprint;
%count(nocum)
%count(stop=30jun08,start=01apr08)
%count(nocum nopercent,stop=30jun08)
%count()
```

77 m103d06c

Macros with Mixed Parameter Lists

m103d06c

Alter the previous macro by using a mixed parameter list. Issue various calls to the macro.

```
%macro count(opts,start=01jan08,stop=31dec08);
proc freq data=orion.orders;
   where order_date between "&start"d and "&stop"d;
   table order_type / &opts;
   title1 "Orders from &start to &stop";
run;
%mend count;
options mprint;
%count(nocum)
%count(stop=30jun08,start=01apr08)
%count(nocum nopercent,stop=30jun08)
%count()
```

Partial SAS Log

```
76   %count(nocum)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "31dec08"d;
MPRINT(COUNT):   table order_type / nocum;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 31dec08";
MPRINT(COUNT):   run;
NOTE: There were 87 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.04 seconds
```

```
      cpu time            0.03 seconds


77   %count(stop=30jun08,start=01apr08)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01apr08"d and "30jun08"d;
MPRINT(COUNT):   table order_type / ;
MPRINT(COUNT):   title1 "Orders from 01apr08 to 30jun08";
MPRINT(COUNT):   run;
NOTE: There were 28 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01APR2008'D and order_date<='30JUN2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

78   %count(nocum nopercent,stop=30jun08)
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "30jun08"d;
MPRINT(COUNT):   table order_type / nocum nopercent;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 30jun08";
MPRINT(COUNT):   run;
NOTE: There were 47 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='30JUN2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

79   %count()
MPRINT(COUNT):   proc freq data=orion.orders;
MPRINT(COUNT):   where order_date between "01jan08"d and "31dec08"d;
MPRINT(COUNT):   table order_type / ;
MPRINT(COUNT):   title1 "Orders from 01jan08 to 31dec08";
MPRINT(COUNT):   run;
NOTE: There were 87 observations read from the data set ORION.ORDERS.
      WHERE (order_date>='01JAN2008'D and order_date<='31DEC2008'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

Exercises

Level 1

4. Defining and Using Macro Parameters

a. Open the **m103e04** program shown below into the Editor window.

```
%macro customers;
   proc print data=orion.customer_dim;
      var Customer_Name Customer_Gender Customer_Age;
      where Customer_Group contains "&type";
      title "&type Customers";
   run;
%mend customers;
```

b. Convert this macro without parameters into a macro with one positional parameter. Name the parameter based on macro variable references within the macro. Set the appropriate system option to display a note in the SAS log when a macro definition has compiled. Submit the macro definition to compile the macro.

c. Call the macro defined in the previous step with a value of *Gold* for the parameter.

d. Call the macro again, but with a parameter value of *Catalog*.

e. Change the positional parameter to a keyword parameter with a default value of *Club*. Submit the revised macro definition to compile the macro.

f. Call the macro defined in the previous step with a value of *Internet* for the keyword parameter.

g. Call the macro again, but enable the macro to use its default parameter value.

Level 2

5. Using a Macro to Generate PROC MEANS Code

a. Open the **m103e05** program shown below into the Editor window.

```
options nolabel;
title 'Order Stats';
proc means data=orion.order_fact maxdec=2 mean;
   var total_retail_price;
   class order_type;
run;
title;
```

b. Create a macro with keyword parameters that generalize the code so that the following attributes are controlled by macro variables. Choose default values for all parameters so that the code executes correctly.

- Statistics: any combination of N, NMISS, MIN, MEAN, MAX, RANGE, or a null value
- Decimal places: 0, 1, 2, 3, or 4
- Analysis variables: **total_retail_price** or **costprice_per_unit** (or both)
- Class variables: **order_type** or **quantity** (or both)

c. Execute the macro using the default parameter values.

d. Call the macro again, but override all default parameter values.

e. Call the macro again, but override only the default parameter values for statistics and decimal places.

Challenge

6. Using Parameters That Contain Special Characters

a. Open the **m103e06** program shown below into the Editor window. Submit the program to compile the macro.

```
%macro specialchars(name);
   proc print data=orion.employee_addresses;
      where Employee_Name="&name";
      var Employee_ID Street_Number Street_Name City State
          Postal_Code;
      title "Data for &name";
   run;
%mend specialchars;
```

b. Execute the macro with a parameter value of *Abbott, Ray*.

PROC PRINT Output

```
                                Data for Abbott, Ray

              Employee_    Street_                                          Postal_
       Obs       ID         Number      Street_Name         City       State  Code

         1      121044       2267     Edwards Mill Rd    Miami-Dade     FL    33135
```

3.3 Solutions

Solutions to Exercises

1. Defining and Calling a Macro

a. Open the program into the Editor window.

b. Convert the program into a macro named **Customers**. Set the MCOMPILENOTE= option and add %MACRO and %MEND statements to create a macro definition.

```
options mcompilenote=all;
%macro customers;
   proc print data=orion.customer_dim;
      var Customer_Name Customer_Gender Customer_Age;
      where Customer_Group contains "&type";
      title "&type Customers";
   run;
%mend customers;
```

SAS Log

```
1    options mcompilenote=all;
2    %macro customers;
3       proc print data=orion.customer_dim;
4          var Customer_Name Customer_Gender Customer_Age;
5          where Customer_Group contains "&type";
6          title "&type Customers";
7        run;
8    %mend customers;
NOTE: The macro CUSTOMERS completed compilation without errors.
      3 instructions 188 bytes.
```

c. Submit a %LET statement to assign the value *Gold* to the macro variable **type**. Call the macro by preceding its name with a percent sign.

```
%let type=Gold;
%customers
```

SAS Log

```
10   %let type=Gold;
11   %customers

NOTE: There were 21 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Gold';
NOTE: PROCEDURE PRINT used (Total process time):
       real time           0.67 seconds
       cpu time            0.18 seconds
```

d. Change the value of **type** to *Internet*.

```
%let type=Internet;
```

e. Set the appropriate system option to display source code received by the SAS compiler. Call the macro again and examine the log.

```
options mprint;
%customers
```

SAS Log

```
12   options mprint;
13   %customers
MPRINT(CUSTOMERS):   proc print data=orion.customer_dim;
MPRINT(CUSTOMERS):   var Customer_Name Customer_Gender Customer_Age;
MPRINT(CUSTOMERS):   where Customer_Group contains "Internet";
MPRINT(CUSTOMERS):   title "Internet Customers";
MPRINT(CUSTOMERS):   run;
NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Internet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

2. Storing a Macro

a. Open the program into the Editor window.

b. Submit the macro definition and check the SAS log.

c. Use the SAS Explorer window or PROC CATALOG to locate the stored macro.

```
proc catalog cat=work.sasmacr;
   contents;
quit;
```

d. Submit a %SYSMACDELETE statement.

```
%sysmacdelete tut;
```

3. Calling a Macro from a TITLE Statement

a. Define a macro that issues the current time of day with the TIMEAMPM. format. Name the macro **Currtime**. Submit the macro definition.

```
%macro currtime;
   %sysfunc(time(),timeAMPM.)
%mend currtime;
```

b. Open the **m103e03** program into the Editor window. Add a TITLE2 statement. Call the macro from the TITLE2 statement. Submit the program and examine the output.

```
proc print data=orion.customer_dim(obs=10);
   var Customer_Name Customer_Group;
   title 'Customer List';
   title2 "%currtime";
run;
```

4. Defining and Using Macro Parameters

a. Open the program into the Editor window.

b. The macro parameter name should be TYPE because the program contains the macro references **&type**. When you define positional parameters, enclose the parameter names in parentheses following the macro name.

```
options mcompilenote=all;
%macro customers(type);
   proc print data=orion.customer_dim;
      var Customer_Name Customer_Gender Customer_Age;
      where Customer_Group contains "&type";
      title "&type Customers";
   run;
%mend customers;
```

c. To execute the macro, use a percent sign followed by the name of the macro. To assign a value to a positional parameter, supply the desired value within parentheses following the macro name.

```
options mprint;
%customers(Gold)
```

SAS Log

```
178  %customers(Gold)
MPRINT(CUSTOMERS):   proc print data=orion.customer_dim;
MPRINT(CUSTOMERS):   var Customer_Name Customer_Gender Customer_Age;
MPRINT(CUSTOMERS):   where Customer_Group contains "Gold";
MPRINT(CUSTOMERS):   title "Gold Customers";
MPRINT(CUSTOMERS):   run;
NOTE: There were 21 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Gold';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.14 seconds
      cpu time            0.00 seconds
```

PROC PRINT Output

```
                        Gold Customers

                                 Customer_    Customer_
Obs    Customer_Name              Gender         Age

  2    Sandrina Stephano             F            28
  3    Cornelia Krahl                F            33
  7    Markus Sepke                  M            19
 11    Oliver S. Füßling             M            43
 17    Cynthia Martinez              F            48
 21    Alphone Greenwald             M            23
 24    Dianne Patchin                F            28
 25    Annmarie Leveille             F            23
 26    Gert-Gunter Mendler           M            73
 31    Carsten Maestrini             M            63
 32    James Klisurich               M            38
 35    Viola Folsom                  F            38
 40    Kyndal Hooks                  F            43
 46    Ramesh Trentholme             M            58
 48    Avni Umran                    M            28
 53    Sanelisiwe Collier            F            19
 57    Rita Lotz                     F            43
 58    Bill Cuddy                    M            21
 62    Susan Krasowski               F            48
 64    Avinoam Tuvia                 M            23
 75    Angel Borwick                 F            38
```

d. The macro definition does not need to be resubmitted with each macro call. The macro call does not end with a semicolon.

```
%customers(Catalog)
```

SAS Log

```
179  %customers(Catalog)
MPRINT(CUSTOMERS):   proc print data=orion.customer_dim;
MPRINT(CUSTOMERS):   var Customer_Name Customer_Gender Customer_Age;
MPRINT(CUSTOMERS):   where Customer_Group contains "Catalog";
MPRINT(CUSTOMERS):   title "Catalog Customers";
MPRINT(CUSTOMERS):   run;
NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Catalog';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

PROC PRINT Output

```
                      Catalog Customers

                                Customer_    Customer_
Obs    Customer_Name             Gender         Age

  8    Ulrich Heyde                 M            68
 13    Tulio Devereaux              M            58
 14    Robyn Klem                   F            48
 15    Cynthia Mccluney             F            38
 16    Candy Kinsey                 F            73
```

```
 20    Phenix Hill             M            43
 59    Avinoam Zweig           M            48
 67    Lauren Marx             F            38
```

e. When you define keyword parameters, an equal sign (=) must follow the name of each parameter. A default value for each parameter can be specified following the equal sign.

```
%macro customers(type=Club);
   proc print data=orion.customer_dim;
      var Customer_Name Customer_Gender Customer_Age;
      where Customer_Group contains "&type";
      title "&type Customers";
   run;
%mend customers;
```

f. To assign a value to a keyword parameter, specify the name of the parameter followed by an equal sign (=), followed by the desired value.

```
%customers(type=Internet)
```

SAS Log

```
180   %customers(type=Internet)
MPRINT(CUSTOMERS):   proc print data=orion.customer_dim;
MPRINT(CUSTOMERS):   var Customer_Name Customer_Gender Customer_Age;
MPRINT(CUSTOMERS):   where Customer_Group contains "Internet";
MPRINT(CUSTOMERS):   title "Internet Customers";
MPRINT(CUSTOMERS):   run;
NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Internet';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

PROC PRINT Output

```
                  Internet Customers

                            Customer_    Customer_
Obs    Customer_Name         Gender         Age

  8    Ulrich Heyde             M            68
 13    Tulio Devereaux          M            58
 14    Robyn Klem               F            48
 15    Cynthia Mccluney         F            38
 16    Candy Kinsey             F            73
 20    Phenix Hill              M            43
 59    Avinoam Zweig            M            48
 67    Lauren Marx              F            38
```

g. To request that all default parameter values be used, follow the macro call with empty parentheses.

```
%customers()
```

SAS Log

```
189  %customers()
MPRINT(CUSTOMERS):   proc print data=orion.customer_dim;
MPRINT(CUSTOMERS):   var Customer_Name Customer_Gender Customer_Age;
```

```
MPRINT(CUSTOMERS):   where Customer_Group contains "Club";
MPRINT(CUSTOMERS):   title "Club Customers";
MPRINT(CUSTOMERS):   run;
NOTE: There were 69 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE customer_group contains 'Club';
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
```

5. Using a Macro to Generate PROC MEANS Code

a. Open the program into the Editor window.

b. Create a macro with keyword parameters that generalize the code so that the following attributes are controlled by macro variables. Choose default values for all parameters so that the code executes correctly.

```
%macro orderstats
 (var=total_retail_price,class=order_type,stats=mean,decimals=2);
    options nolabel;
    title 'Order Stats';
    proc means data=orion.order_fact maxdec=&decimals &stats;
       var &var;
        class &class;
    run;
    title;
%mend orderstats;
```

c. Execute the macro using the default parameter values.

```
%orderstats()
```

SAS Log

```
85   %orderstats()
MPRINT(ORDERSTATS):   options nolabel;
MPRINT(ORDERSTATS):   title 'Order Stats';
MPRINT(ORDERSTATS):   proc means data=orion.order_fact maxdec=2 mean;
MPRINT(ORDERSTATS):   var total_retail_price;
MPRINT(ORDERSTATS):   class order_type;
MPRINT(ORDERSTATS):   run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time          0.04 seconds
      cpu time           0.04 seconds

MPRINT(ORDERSTATS):   title;
```

d. Call the macro again, but override all default parameter values.

```
%orderstats(var=costprice_per_unit, class=quantity, stats=min mean max, decimals=0)
```

SAS Log

```
 86   %orderstats(var=costprice_per_unit, class=quantity, stats=min mean max, decimals=0)
MPRINT(ORDERSTATS):   options nolabel;
MPRINT(ORDERSTATS):   title 'Order Stats';
MPRINT(ORDERSTATS):   proc means data=orion.order_fact maxdec=0 min mean max;
```

```
MPRINT(ORDERSTATS):   var costprice_per_unit;
MPRINT(ORDERSTATS):   class quantity;
MPRINT(ORDERSTATS):   run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds

MPRINT(ORDERSTATS):   title;
```

e. Call the macro again, but override only the default parameter values for statistics and decimal places.

```
%orderstats(stats=min mean max, decimals=0)
```

SAS Log

```
87   %orderstats(stats=min mean max, decimals=0)
MPRINT(ORDERSTATS):   options nolabel;
MPRINT(ORDERSTATS):   title 'Order Stats';
MPRINT(ORDERSTATS):   proc means data=orion.order_fact maxdec=0 min mean max;
MPRINT(ORDERSTATS):   var total_retail_price;
MPRINT(ORDERSTATS):   class order_type;
MPRINT(ORDERSTATS):   run;

NOTE: Multiple concurrent threads will be used to summarize data.
NOTE: There were 617 observations read from the data set ORION.ORDER_FACT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds

MPRINT(ORDERSTATS):   title;
```

6. Using Parameters That Contain Special Characters

a. Open the program into the Editor window. Submit the program to compile the macro.

b. The %STR function is required to prevent a special character such as a comma from being misinterpreted as a parameter delimiter on the macro call.

```
options mprint;
%specialchars(%str(Abbott, Ray))
```

SAS Log

```
63   %specialchars(%str(Abbott, Ray))
MPRINT(SPECIALCHARS):   proc print data=orion.employee_addresses;
MPRINT(SPECIALCHARS):   where Employee_Name="Abbott, Ray";
MPRINT(SPECIALCHARS):   var Employee_ID Street_Number Street_Name City State Postal_Code;
MPRINT(SPECIALCHARS):   title "Data for Abbott, Ray";
MPRINT(SPECIALCHARS):   run;
NOTE: There were 1 observations read from the data set ORION.EMPLOYEE_ADDRESSES.
      WHERE Employee_Name='Abbott, Ray';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           2.51 seconds
      cpu time            0.00 seconds
```

Solutions to Student Activities (Polls/Quizzes)

3.01 Poll – Correct Answer

Does the macro call below require a semicolon?

%Time

❍ Yes
❍ No

A macro call is not a statement. A semicolon is not required and can cause problems.

11

3.02 Multiple Choice Poll – Correct Answer

A %LET statement outside a macro definition creates a macro variable in which of the following?

a. global symbol table
b. local symbol table

67

3.03 Short Answer Poll – Correct Answer

Edit the macro call to omit the parentheses.

```
%dog
```

Submit the macro call again.

What do you see in the SAS log?

SAS windowing environment: The macro call does not execute without parentheses.

SAS Enterprise Guide: The macro call executes without parentheses.

75

Chapter 4 DATA Step and SQL Interfaces

4.1 Creating Macro Variables in the DATA Step

Objectives

- Create macro variables during DATA step execution.
- Describe the difference between the SYMPUTX routine and the %LET statement.

3

Business Scenario

Automate production of the report below, with a footnote that indicates whether there are any Internet orders.

Orders for 2-2011

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	05FEB2011	1	1	$117.60
2	07FEB2011	1	2	$656.60
3	07FEB2011	1	2	$129.00
4	09FEB2011	1	2	$36.20
5	16FEB2011	1	1	$29.40
6	28FEB2011	1	5	$192.00

No Internet Orders

Internet orders have an **Order_Type** value of *3*.

4

Many applications require macro variables to store values based on data, programming logic, or expressions.

DATA Step Interface

```
%let month=2;
%let year=2011;

data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         %let foot=No Internet Orders;
      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;

proc print data=orders;
   title "Orders for &month-&year";
   footnote "&foot";
run;
```

5

m104d01a

DATA Step Interface

Why is the footnote incorrect?

Orders for 2-2011

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	05FEB2011	1	1	$117.60
2	07FEB2011	1	2	$656.60
3	07FEB2011	1	2	$129.00
4	09FEB2011	1	2	$36.20
5	16FEB2011	1	1	$29.40
6	28FEB2011	1	5	$192.00

Some Internet Orders

m104d01a

6

DATA Step Interface

Word scanning begins. Macro triggers are encountered.

```
%let month=2;
%let year=2011;
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         %let foot=No Internet Orders;
      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;
```

Symbol Table	
month	2
year	2011

7 ...

DATA Step Interface

Compiling begins. Macro variable references are resolved.

```
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=2011 and month(order_date)=2;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         %let foot=No Internet Orders;
      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;
```

Symbol Table	
month	2
year	2011

8 ...

DATA Step Interface

The macro trigger is passed to the macro processor.

```
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=2011 and month(order_date)=2;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         %let foot=No Internet Orders;
      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;
```

Symbol Table

month	2
year	2011
foot	No Internet Orders

9 ...

DATA Step Interface

The macro variable **foot** is assigned.

```
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=2011 and month(order_date)=2;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;

      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;
```

Symbol Table

month	2
year	2011
foot	No Internet Orders

10 ...

DATA Step Interface

The macro trigger overwrites the previous value.

```
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=2011 and month(order_date)=2;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;

      end;
      else do;
         %let foot=Some Internet Orders;
      end;
   end;
run;
```

Symbol Table	
month	2
year	2011
foot	Some Internet Orders

11 ...

%LET statements execute at word-scanning time. SAS statements are sent to the compiler.

DATA Step Interface

The compilation phase is complete. Ready for execution.

```
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=2011 and month(order_date)=2;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;

      end;
      else do;

      end;
   end;
run;
```

Symbol Table	
month	2
year	2011
foot	Some Internet Orders

Nothing in this DATA step affects the value of FOOT.

The value of **foot** remains *Some Internet Orders*.

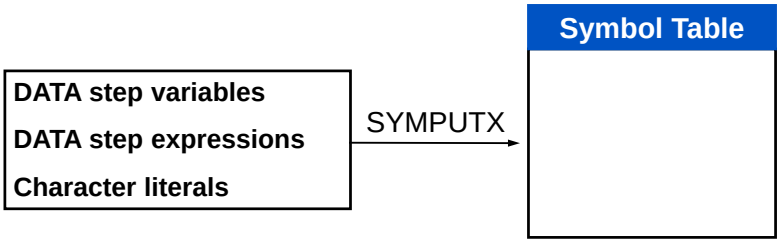
12

SYMPUTX Routine

The SYMPUTX routine assigns to a macro variable any value available to the DATA step during execution time.

It can create macro variables with the following:

- static values
- dynamic (data dependent) values
- dynamic (data dependent) names



13

The SYMPUTX routine was introduced in SAS®9. In prior versions of SAS, the SYMPUT routine was available.

Completed Business Scenario

The SYMPUTX routine is a DATA step statement that assigns a text value to a macro variable.

```
%let month=2;
%let year=2011;

data orders;
   keep order_date order_type quantity total_retail_price;
   set orders.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         call symputx('foot', 'No Internet Orders');
      end;
      else do;
         call symputx('foot', 'Some Internet Orders');
      end;
   end;
run;
```

CALL SYMPUTX(*macro-variable, value*);

fixed macro variable name

fixed macro variable value

no macro triggers

14 m104d01b

Completed Business Scenario

The footnote is correct.

Orders for 2-2011

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	05FEB2011	1	1	$117.60
2	07FEB2011	1	2	$656.60
3	07FEB2011	1	2	$129.00
4	09FEB2011	1	2	$36.20
5	16FEB2011	1	1	$29.40
6	28FEB2011	1	5	$192.00

No Internet Orders

15 m104d01b

Macro variables created by the SYMPUTX routine are available following DATA step execution and can be referenced after a step boundary.

SYMPUTX Routine

m104d01b

Conditionally assign a text value to a macro variable **foot** based on DATA step values. Reference this macro variable later in the program.

```
%let month=2;
%let year=2011;
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then do;
      put Number=;
      if Number=0 then do;
         call symputx('foot', 'No Internet Orders');
      end;
      else do;
         call symputx('foot', 'Some Internet Orders');
      end;
   end;
run;

proc print data=orders;
   title "Orders for &month-&year";
```

```
   footnote "&foot";
run;
```

The value assigned to the macro variable **foot** is set dynamically to *No Internet Orders* or *Some Internet Orders*, based on DATA step execution-time logic.

PROC PRINT Output

```
                    Orders for 2-2011

            Order_    Order_               Total_Retail_
   Obs        Date      Type     Quantity       Price

    1     05FEB2011       1          1           $117.60
    2     07FEB2011       1          2           $656.60
    3     07FEB2011       1          2           $129.00
    4     09FEB2011       1          2            $36.20
    5     16FEB2011       1          1            $29.40
    6     28FEB2011       1          5           $192.00



                    No Internet Orders
```

4.01 Multiple Choice Poll

What is the value of **foot** after execution of the DATA step?

a. No Internet orders
b. Some Internet orders

```
data _null_;
   call symputx('foot','No Internet orders');
   %let foot=Some Internet orders;
run;
```

17

Business Scenario

Enhance the footnote on the previous report to include the total number of Internet orders.

Orders for 1-2011

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	02JAN2011	3	2	$195.60
2	03JAN2011	1	6	$160.80
3	04JAN2011	1	2	$306.20
4	06JAN2011	3	3	$37.80
5	13JAN2011	1	2	$362.60
6	23JAN2011	1	1	$72.60
7	24JAN2011	1	2	$258.20
8	24JAN2011	1	2	$81.20
9	24JAN2011	1	3	$358.20
10	25JAN2011	3	1	$102.40
11	25JAN2011	3	1	$113.20
12	28JAN2011	3	2	$174.40
13	29JAN2011	2	1	$37.40

5 Internet Orders

19 m104d01c

SYMPUTX Routine

Copy the value of a DATA step variable into a macro variable.

```
%let month=1;
%let year=2011;

data orders;
   keep order_date order_type quantity total_retail_price;
   set orders.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then call symputx('num', Number);
run;

proc print data=orders;
   title "Orders for &month-&year";
   footnote "&num Internet Orders";
run;
```

- A maximum of 32,767 characters can be assigned to a macro variable.
- Values of numeric variables are converted automatically to character using the BEST. format, with a width up to 32 characters.
- Leading and trailing blanks are removed from both SYMPUTX arguments.

20 m104d01c

The SYMPUT routine, which remains available, has syntax and functionality that is similar to that of the SYMPUTX routine. The SYMPUT routine does not trim leading and trailing blanks automatically from either argument. Therefore, it is often necessary to use TRIM and LEFT functions within the SYMPUT routine. When assigning numeric values to macro variables, the SYMPUT routine writes numeric-to-character conversion notes to the log.

SYMPUTX Routine

m104d01c

```
%let month=1;
%let year=2011;

data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then call symputx('num', Number);
run;

proc print data=orders;
   title "Orders for &month-&year";
   footnote "&num Internet Orders";
run;
```

PROC PRINT Output

```
                    Orders for 1-2011

            Order_    Order_                Total_Retail_
    Obs       Date      Type     Quantity        Price

      1    02JAN2011       3          2           $195.60
      2    03JAN2011       1          6           $160.80
      3    04JAN2011       1          2           $306.20
      4    06JAN2011       3          3            $37.80
      5    13JAN2011       1          2           $362.60
      6    23JAN2011       1          1            $72.60
      7    24JAN2011       1          2           $258.20
      8    24JAN2011       1          2            $81.20
      9    24JAN2011       1          3           $358.20
     10    25JAN2011       3          1           $102.40
     11    25JAN2011       3          1           $113.20
     12    28JAN2011       3          2           $174.40
     13    29JAN2011       2          1            $37.40

                    5 Internet Orders
```

Business Scenario

Further enhance the footnotes by adding the average Internet order amount and last Internet order date.

Orders for 1-2011

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	02JAN2011	3	2	$195.60
2	03JAN2011	1	6	$160.80
3	04JAN2011	1	2	$306.20
4	06JAN2011	3	3	$37.80
5	13JAN2011	1	2	$362.60
6	23JAN2011	1	1	$72.60
7	24JAN2011	1	2	$258.20
8	24JAN2011	1	2	$81.20
9	24JAN2011	1	3	$358.20
10	25JAN2011	3	1	$102.40
11	25JAN2011	3	1	$113.20
12	28JAN2011	3	2	$174.40
13	29JAN2011	2	1	$37.40

Average Internet Order: $125
Last Internet Order: 01/28/2011

22 m104d01d

The data set is sorted by **Order_Date**.

SYMPUTX Routine

m104d01d

```
%let month=1;
%let year=2011;
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then do;
      Number+1;
      Amount+total_retail_price;
      Date=order_date;
      retain date;
      end;
   if final then do;
      if number=0 then do;
          call symputx('dat', 'N/A');
          call symputx('avg', 'N/A');
          end;
      else do;
         call symputx('dat', put(date,mmddyy10.));
         call symputx('avg', put(amount/number,dollar8.));
        end;
      end;
run;
```

```
proc print data=orders;
   title "Orders for &month-&year";
   footnote1 "Average Internet Order: &avg";
   footnote2 "Last Internet Order: &dat";
run;
```

The PUT function returns a character string by writing a value with a specified format.

You can use the PUT function to do the following:

- format the result of a numeric expression
- perform explicit numeric-to-character conversion

General form of the PUT function:

PUT(*source*, *format***)**

source is a constant, variable, or expression (numeric or character).

format is any SAS format or user-defined format. It determines the width of the resulting string and whether the string is right-aligned or left-aligned. The type for *format* must match the type for *source*.

PROC PRINT Output

Orders for 1-2007

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
1	02JAN2011	3	2	$195.60
2	03JAN2011	1	6	$160.80
3	04JAN2011	1	2	$306.20
4	06JAN2011	3	3	$37.80
5	13JAN2011	1	2	$362.60
6	23JAN2011	1	1	$72.60
7	24JAN2011	1	2	$258.20
8	24JAN2011	1	2	$81.20
9	24JAN2011	1	3	$358.20
10	25JAN2011	3	1	$102.40
11	25JAN2011	3	1	$113.20
12	28JAN2011	3	2	$174.40
13	29JAN2011	2	1	$37.40

Average Internet Order: $125
Last Internet Order: 01/28/2011

4.02 Short Answer Poll

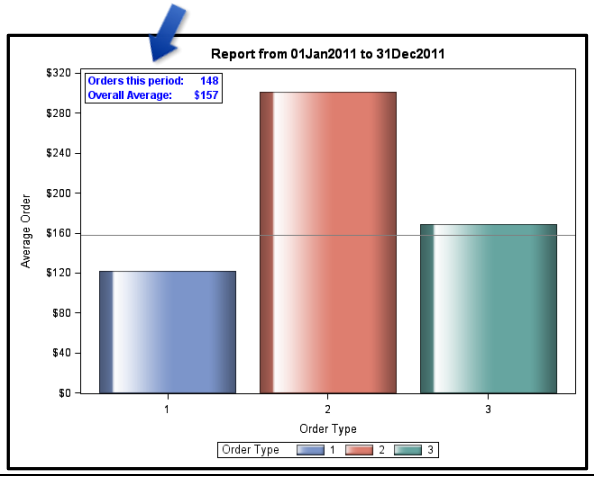
This CALL SYMPUTX statement creates a macro variable named **date**. Complete the call to assign the value of the variable **current**, formatting the result as a date such as 21NOV2012.

```
call symputx('date',                    );
```

24

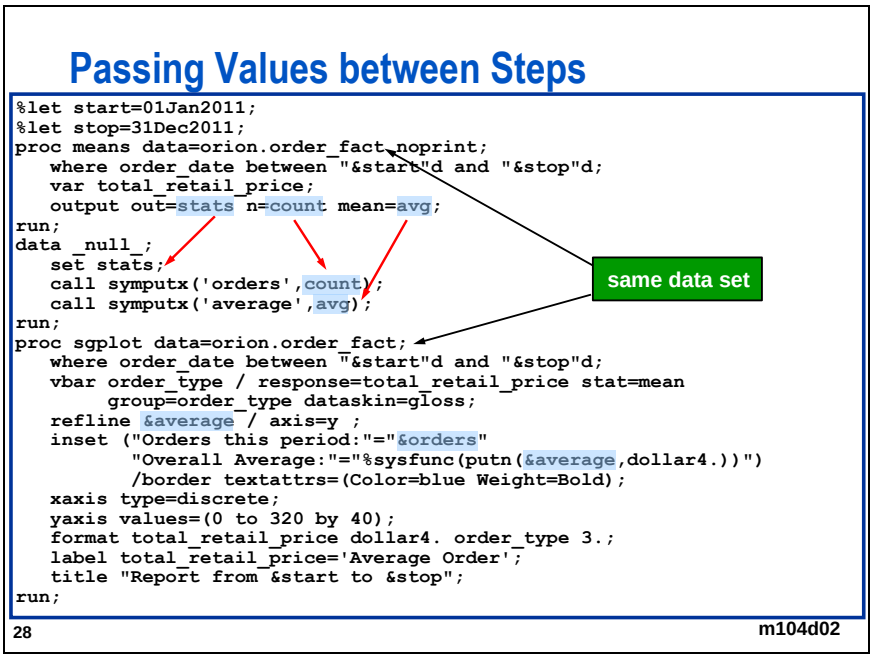
Business Scenario

Based on user-selected time periods, dynamically compute statistics for automatic inclusion within the graph below.



27

m104d02



The PUTN function returns a character string by writing a value with a numeric format.

Passing Values between Steps

m104d02

```
%let start=01Jan2011;
%let stop=31Dec2011;

proc means data=orion.order_fact noprint;
   where order_date between "&start"d and "&stop"d;
   var total_retail_price;
   output out=stats n=count mean=avg;
run;

data _null_;
   set stats;
   call symputx('orders',count);
   call symputx('average',avg);
run;

proc sgplot data=orion.order_fact;
   where order_date between "&start"d and "&stop"d;
   vbar order_type / response=total_retail_price stat=mean
        group=order_type dataskin=gloss;
   refline &average / axis=y ;
   inset ("Orders this period:" = "&orders"
          "Overall Average:" = "%sysfunc(putn(&average,dollar4.))")
      /border textattrs=(Color=blue Weight=Bold);
```

```
   xaxis type=discrete;
   yaxis values=(0 to 320 by 40);
   format total_retail_price dollar4. order_type 3.;
   label total_retail_price='Average Order';
   title1  "Report from &start to &stop";
run;
```

Exercises

Level 1

1. Creating Macro Variables with the SYMPUTX Routine

a. Open the **m104e01** program shown below into the Editor window. Submit the program and examine the output that it creates.

```
data staff;
   keep employee_ID job_title salary gender;
   set orion.staff;
   where job_title contains 'Audit';
run;

proc print data=staff;
   sum salary;
   title 'Audit Staff';
run;
```

b. Add a %LET statement before the DATA step to create the macro variable **job** with the value *Audit*. Replace hardcoded values of *Audit* with references to the macro variable **job**. Resubmit the program. It should produce the same output as before.

c. Change the value of the macro variable **job** from *Audit* to *Analyst*. Resubmit the program and examine the new results.

d. Modify the DATA step to create the macro variable **avg** to store the average salary. Reference the macro variable **avg** in a FOOTNOTE statement. Format the average salary with a dollar sign, comma, and no decimal places.

PROC PRINT Output

```
                            Analyst Staff

Obs     Employee_ID    Job_Title                     Salary    Gender

 1          120263    Financial Analyst III        $42,605      M
 2          120264    Financial Analyst II         $37,510      F
 3          120710    Business Analyst II          $54,840      M
 4          120711    Business Analyst III         $59,130      F
 5          120775    HR Analyst II                $41,580      F
 6          120779    HR Analyst II                $43,690      F
 7          121148    Business Analyst II          $52,930      M
                                                 ============
                                                   $332,285
```

```
Average Salary: $47,469
```

Level 2

2. Creating Macro Variables with the SYMPUTX Routine

a. Open the **m104e02** program shown below into the Editor window. This program creates a summary data set named **customer_sum** that summarizes **Total_Retail_Price** by **Customer_ID** and sorts the data set by descending **CustTotalPurchase**. Submit the program (part **a**) and examine the output that it creates.

```
proc means data=orion.order_fact nway noprint;
   var Total_Retail_Price;
   class Customer_ID;
   output out=customer_sum sum=CustTotalPurchase;
run;

proc sort data=customer_sum;
   by descending CustTotalPurchase;
run;

proc print data=customer_sum(drop=_type_);
run;
```

b. Create a macro variable named **top** that contains the ID number for the top customer. Then modify the program (part **b**) to print only the orders for Orion's top customer.

Partial PROC PRINT Output

Orders for Customer 16 - Orion's Top Customer

Order_ID	Order_ Type	Order_ Date	Delivery_ Date
1230450371	2	24MAR2007	26MAR2007
1231305521	2	27AUG2007	04SEP2007
1231305531	2	27AUG2007	29AUG2007
1234538390	2	12JAN2009	14JAN2009
1234588648	2	17JAN2009	19JAN2009

Challenge

3. Creating Macro Variables with the SYMPUTX Routine

a. Open the **m104e03** program shown below into the Editor window. Submit the program and examine the output that it creates.

```
proc means data=orion.order_fact nway noprint;
   var Total_Retail_Price;
   class Customer_ID;
   output out=customer_sum sum=CustTotalPurchase;
run;
```

```
proc sort data=customer_sum ;
   by descending CustTotalPurchase;
run;

proc print data=customer_sum(drop=_type_);
run;
```

b. Using the **customer_sum** data set, create a single macro variable, **top3**, that contains the customer IDs of the top three customers by revenue.

✎ **Customer_ID** is a numeric variable.

c. Using the **orion.customer_dim** data set, print a listing of the top three customers.

PROC PRINT Output

Top 3 Customers

Customer_ID	Customer_Name	Customer_Type
10	Karen Ballinger	Orion Club members high activity
16	Ulrich Heyde	Internet/Catalog Customers
45	Dianne Patchin	Orion Club Gold members low activity

4.2 Indirect References to Macro Variables

Objectives

- Reference macro variables indirectly.
- Create a series of macro variables using the SYMPUTX routine.

33

Business Scenario

Create an order history for a given customer. Report titles should display customer name and number.

```
            Customer Number: 9
          Customer Name: Cornelia Krahl

          Order_   Order_              Total_Retail_
  Obs      Date     Type     Quantity      Price

  160   15APR2008    3          1           $29.40
  273   07JUN2009    3          2           $16.00
  288   10AUG2009    3          3        $1,542.60
  289   10AUG2009    3          2          $550.20
  316   02DEC2009    3          2           $39.20
  326   25DEC2009    3          1          $514.20
```

34

Table Lookup Application

Step 1 Hardcode the program, including customer name and number.

```
proc print data=orion.order_fact;
   where customer_ID=9;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: 9";
   title2 "Customer Name: Cornelia Krahl";
run;
```

35

m104d03a

Table Lookup Application

Step 2 Create and reference a macro variable for the customer number.

```
%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: Cornelia Krahl";
run;
```

How can you reference the customer name in TITLE2 without hardcoding it?

36 m104d03b

Table Lookup Application

The **orion.customer** data set contains customer names and ID numbers. Customer ID numbers are unique.

Orion.Customer

Obs	Customer_ID	Customer_Name	Country	Gender	Birth_ Date
1	4	James Kvarniq	US	M	27JUN1974
2	5	Sandrina Stephano	US	F	09JUL1979
3	9	Cornelia Krahl	DE	F	27FEB1974
4	10	Karen Ballinger	US	F	18OCT1984
5	11	Elke Wallstab	DE	F	16AUG1974
6	12	David Black	US	M	12APR1969
7	13	Markus Sepke	DE	M	21JUL1988
8	16	Ulrich Heyde	DE	M	16JAN1939
9	17	Jimmie Evans	US	M	17AUG1954
10	18	Tonie Asmussen	US	M	02FEB1954
11	19	Oliver S. Füßling	DE	M	23FEB1964
12	20	Michael Dineley	US	M	17APR1959

37

Table Lookup Application

Step 3 Add a DATA step to create a macro variable with the customer's name.

Reference the macro variable in TITLE2.

```
%let custID=9;

data _null_;
   set orion.customer;
   where customer_ID=&custID;
   call symputx('name', Customer_Name);
run;

proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name";
run;
```

same WHERE statement

38 m104d03c

4.03 Short Answer Poll

How many rows are selected by the DATA step WHERE statement in the preceding program, repeated below?

```
%let custID=9;
data _null_;
   set orion.customer;
   where customer_ID=&custID;
   call symputx('name', Customer_Name);
run;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name";
run;
```

39 m104d03c

Table Lookup Application

To select ***all*** customers, eliminate the WHERE statement from the DATA step.

```
%let custID=9;
data _null_;
   set orion.customer;
   call symputx('name', Customer_Name);
run;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name";
run;
```

What is the problem this time?

41

Table Lookup Application

Because only one macro variable is created by the SYMPUTX routine, its value is overwritten with each iteration of the DATA step.

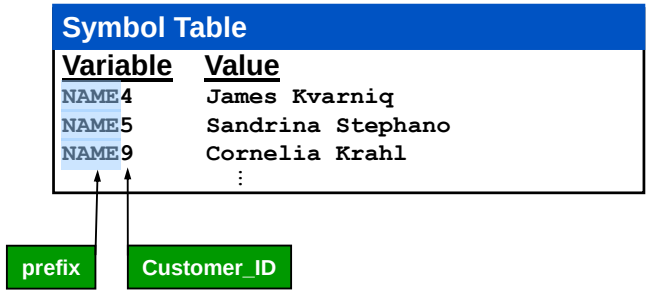
```
%let custID=9;
data _null_;
   set orion.customer;
   call symputx('name', Customer_Name);
run;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name";
run;
```

Unique macro variable names are required.

42

Creating a Series of Macro Variables

Derive unique macro variable names by appending the customer ID number to a fixed prefix.



43

Creating a Series of Macro Variables

Step 4 Create a series of macro variables to store customer names.

```
data _null_;
   set orion.customer;
   call symputx('name'||left(Customer_ID),
        Customer_Name);
run;
```

Generate unique macro variable names with a DATA step character variable or expression as the first argument to the SYMPUTX routine.

44 m104d03d

Creating a Series of Macro Variables

Partial SAS Log

```
639627  %put _user_;
GLOBAL NAME10 Karen Ballinger
GLOBAL NAME1033 Selim Okay
GLOBAL NAME11 Elke Wallstab
GLOBAL NAME1100 Ahmet Canko
GLOBAL NAME111 Karolina Dokter
GLOBAL NAME11171 Bill Cuddy
GLOBAL NAME12 David Black
GLOBAL NAME12386 Avinoam Zweig
GLOBAL NAME13 Markus Sepke
GLOBAL NAME14104 Avinoam Zweig
GLOBAL NAME14703 Eyal Bloch
GLOBAL NAME16 Ulrich Heyde
GLOBAL NAME1684 Carglar Aydemir
GLOBAL NAME17 Jimmie Evans
GLOBAL NAME17023 Susan Krasowski
GLOBAL NAME171 Robert Bowerman
GLOBAL NAME18 Tonie Asmussen
GLOBAL NAME183 Duncan Robertshawe
GLOBAL NAME19 Oliver S. Füßling
```

45

Creating a Series of Macro Variables

You can now reference the correct name without rerunning the DATA step.

Symbol Table

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

```
%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name9";
run;
```

46

m104d03e

4.04 Short Answer Poll

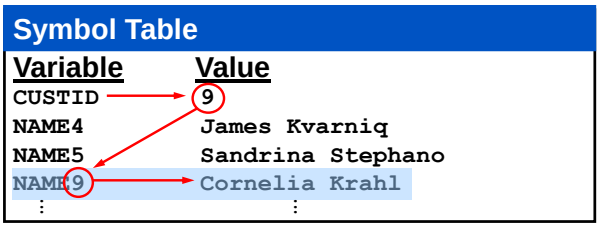
Modify program **m104a02** to create an order history for **custID 4**. How many program changes are required?

```
%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name9";
run;
```

47 m104a02

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

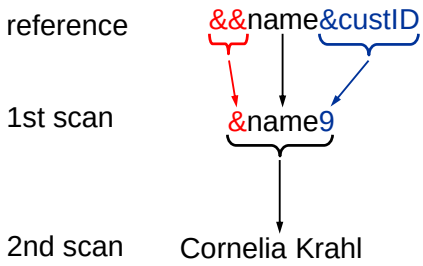
Step 5 Use an indirect reference.

```
%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &&name&custID";
run;
```

51 m104d03f

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

Completed Business Scenario

PROC PRINT Output

Customer Number: 9
Customer Name: Cornelia Krahl

Obs	Order_ Date	Order_ Type	Quantity	Total_Retail_ Price
160	15APR2004	3	1	$29.40
273	07JUN2005	3	2	$16.00
288	10AUG2005	3	3	$1,542.60
289	10AUG2005	3	2	$550.20
316	02DEC2005	3	2	$39.20
326	25DEC2005	3	1	$514.20

54

4.05 Short Answer Poll

Submit program **m104a03**.

```
%let custid=9;
%let name9=Joe;
%put &name&custid;
```

How many times are the macro variables scanned in the %PUT statement?

Is this successful?

m104a03

55

Indirect References to Macro Variables

m104d04

```
data _null_;
   set orion.customer;
   call symputx('name'||left(Customer_ID), customer_Name);
run;

%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &&name&custID";
run;
```

Partial SAS Log

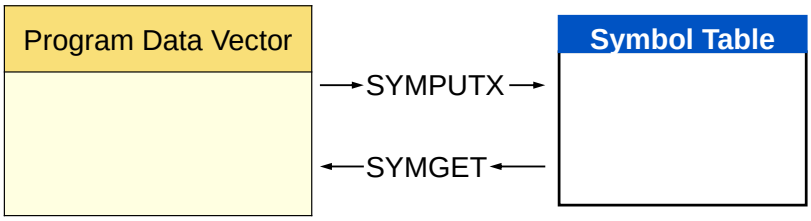
```
451  %let custID=9;
452  proc print data=orion.order_fact;
453      where customer_ID=&custID;
SYMBOLGEN:  Macro variable CUSTID resolves to 9
454      var order_date order_type quantity total_retail_price;
SYMBOLGEN:  Macro variable CUSTID resolves to 9
455      title1 "Customer Number: &custID";
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable CUSTID resolves to 9
SYMBOLGEN:  Macro variable NAME9 resolves to Cornelia Krahl
456      title2 "Customer Name: &&name&custID";
457  run;

NOTE: There were 6 observations read from the data set ORION.ORDER_FACT.
      WHERE customer_ID=9;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

PROC PRINT Output

```
                          Customer Number: 9
                     Customer Name: Cornelia Krahl

                  Order_    Order_                Total_Retail_
           Obs      Date     Type     Quantity        Price

           160    15APR2008      3          1           $29.40
           273    07JUN2009      3          2           $16.00
           288    10AUG2009      3          3        $1,542.60
           289    10AUG2009      3          2          $550.20
           316    02DEC2009      3          2           $39.20
           326    25DEC2009      3          1          $514.20
```

SYMGET Function

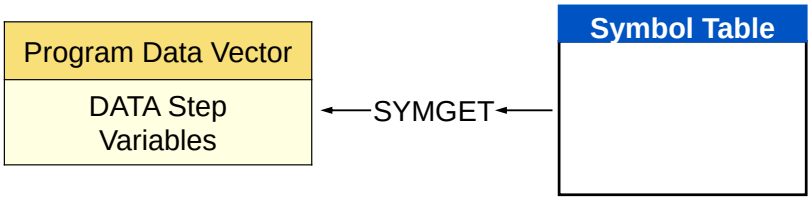
	Create macro variables	Resolve macro variables
Word scanning time	%LET	&macvar
Execution time	CALL SYMPUTX	SYMGET(macvar)



63

SYMGET Function

The SYMGET function retrieves a macro variable's value during DATA step execution.



The SYMGET function is covered in SAS® Macro Language 2: Advanced Techniques.

64

Exercises

Level 1

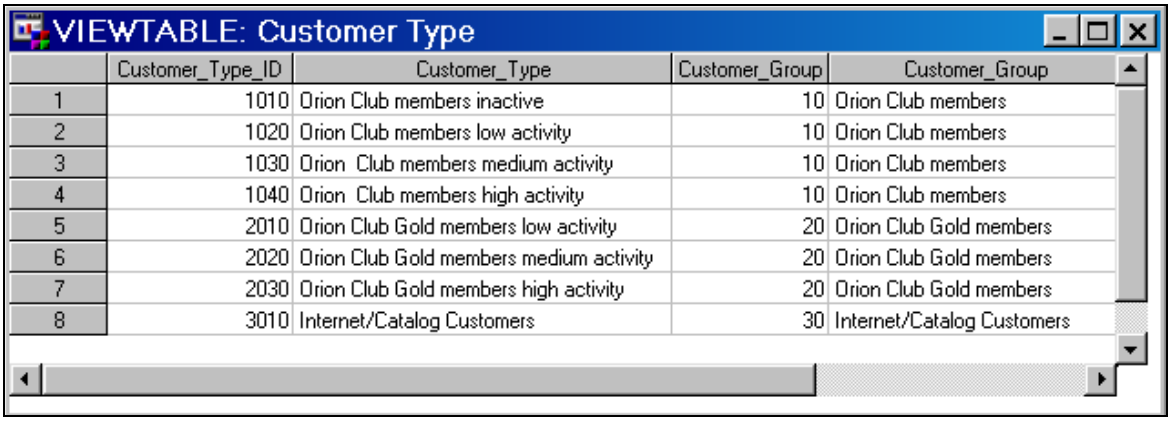
4. Creating a Series of Macro Variables with the SYMPUTX Routine

a. Open the **m104e04** program shown below into the Editor window.

```
%macro memberlist(id=1020);
   %put _user_;
   title "A List of &id";
   proc print data=orion.customer;
      var Customer_Name Customer_ID Gender;
      where Customer_Type_ID=&id;
   run;
%mend memberlist;

%memberlist()
```

b. The **orion.customer_type** data set contains the variable **Customer_Type_ID**, which uniquely identifies the customer membership level and activity level. Add a DATA step to create a series of macro variables named **type*xxxx***, where ***xxxx*** is the value of **Customer_Type_ID**. The value of each **type** macro variable should be the value of **Customer_Type**. Place the DATA step before the macro definition.

Listing of **orion.customer_type**



VIEWTABLE: Customer Type

	Customer_Type_ID	Customer_Type	Customer_Group	Customer_Group
1	1010	Orion Club members inactive	10	Orion Club members
2	1020	Orion Club members low activity	10	Orion Club members
3	1030	Orion Club members medium activity	10	Orion Club members
4	1040	Orion Club members high activity	10	Orion Club members
5	2010	Orion Club Gold members low activity	20	Orion Club Gold members
6	2020	Orion Club Gold members medium activity	20	Orion Club Gold members
7	2030	Orion Club Gold members high activity	20	Orion Club Gold members
8	3010	Internet/Catalog Customers	30	Internet/Catalog Customers

c. Modify the TITLE statement so that it displays the appropriate customer type. Use an indirect macro variable reference to one of the **type** variables based on the current value of ID. Submit the modified program.

Partial PROC PRINT Output

```
                    A List of Orion Club members low activity

                Obs    Customer_Name           Customer_ID    Gender

                  1    James Kvarniq                 4         M
                 10    Tonie Asmussen               18         M
                 19    Alvan Goheen                 34         M
```

d. Call the macro again, but with a parameter value of *2030*.

Partial PROC PRINT Output

```
                    A List of Orion Club Gold members high activity

                Obs    Customer_Name           Customer_ID    Gender

                 11    Oliver S. Füßling            19         M
```

```
21    Alphone Greenwald              39      M
25    Annmarie Leveille              49      F
26    Gert-Gunter Mendler            50      M
```

Level 2

5. Using Indirect References in a Macro Call

a. Open the **m104e05** program shown below into the Editor window. Submit the program and examine the results.

```
data _null_;
   set orion.customer_type;
   call symputx('type'||left(Customer_Type_ID), Customer_Type);
run;

%put _user_;

%macro memberlist(custtype);
   proc print data=orion.customer_dim;
      var Customer_Name Customer_ID Customer_Age_Group;
      where Customer_Type="&custtype";
      title "A List of &custtype";
   run;
%mend memberlist;
```

b. Create a macro variable named **num** with the value *2010*. Call the **Memberlist** macro. Pass the appropriate parameter to the **Memberlist** macro, such that **custtype** resolves to *Orion Club Gold members low activity* on the macro call.

PROC PRINT Output

```
                A List of Orion Club Gold members low activity

                                                          Customer_
              Obs    Customer_Name          Customer_ID    Age_Group

                7    Markus Sepke                    13    15-30 years
               24    Dianne Patchin                  45    15-30 years
               53    Sanelisiwe Collier            2550    15-30 years
               58    Bill Cuddy                   11171    15-30 years
               75    Angel Borwick                70201    31-45 years
```

Challenge

6. Using a Table Lookup Application

a. Using **orion.country**, create a series of macro variables in which the name of the macro variable is the country abbreviation (**Country**) and the value of the macro variable is the country name (**Country_Name**).

b. Open the **m104e06** program shown below into the Editor window.

```
%let code=AU;
proc print data=Orion.Employee_Addresses;
   var Employee_Name City;
   where Country="&code";
   title "A List of xxxxx Employees";
run;
```

c. Use indirect macro variable referencing to replace the ***xxxxx*** with the appropriate country name.

Partial PROC PRINT Output

```
                          A List of Australia Employees

                      Obs    Employee_Name            City

                        2    Aisbitt, Sandy           Melbourne
                       17    Bahlman, Sharon          Sydney
                       18    Baker, Gabriele          Sydney
                       22    Baran, Shanmuganathan    Sydney
                       23    Barbis, Viney            Sydney
                       24    Barcoe, Selina           Melbourne
                       25    Barreto, Geok-Seng       Sydney
                       31    Billington, Kareen       Sydney
                       34    Blanton, Brig            Melbourne
                       37    Body, Meera              Sydney
                       48    Buddery, Jeannette       Sydney
                       52    Cantatore, Lorian        Sydney
```

4.3 Creating Macro Variables in SQL

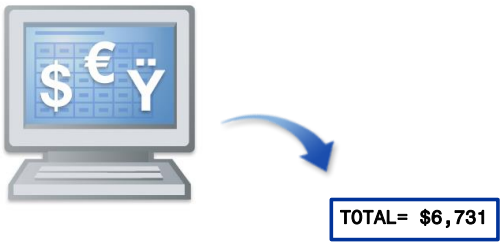
Objectives

- Create macro variables during PROC SQL execution.
- Store several values in one macro variable using the SQL procedure.

67

Business Scenario

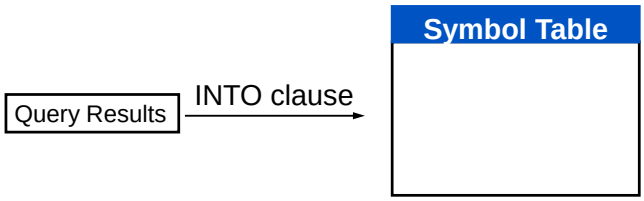
Create a macro variable that contains the total price of all 2011 Internet orders.



68

INTO Clause

The INTO clause creates macro variables from query results.



69

INTO Clause

This INTO clause creates a single macro variable named **total**.

```
proc sql noprint;
   select sum(total_retail_price) format=dollar8.
      into :total
      from orion.order_fact
      where year(order_date)=2011 and order_type=3;
quit;

%put &=total;
```

SELECT *col1*, *col2*, ...
INTO *:mvar1*, *:mvar2*,...
FROM *table-expression*
WHERE *where-expression*
ORDER BY *col1*, *col2*, . . . ;

Partial SAS Log

```
TOTAL= $6,731
```

This form of the INTO clause does not trim leading or trailing blanks.

70 m104d05a

A %LET statement can be used to trim leading and trailing blanks created by the INTO clause:

```
%let macrovariable=&macrovariable;
```

INTO Clause

The INTO clause can create multiple macro variables.

Example: Create macro variables with the date and amount of the top three sales from 2011.

```
title 'Top 2011 Sales';

proc sql outobs=3 double;
   select total_retail_price, order_date format=mmddyy10.
      into :price1-:price3, :date1-:date3
      from orion.order_fact
      where year(order_date)=2011
      order by total_retail_price desc;
quit;
```

71 m104d05b

INTO Clause

SQL Result

Top 2011 Sales

Total_Retail_ Price	Order_Date
$1,937.20	06/20/2011
$1,066.40	11/01/2011
$760.80	12/12/2011

Macro variables: price1⇨ price2⇨ price3⇨

Macro variables: ⇦date1 ⇦date2 ⇦date3

Partial SAS Log

```
1529 %put &price1 &date1, &price2 &date2, &price3 &date3;
$1,937.20 06/20/2011, $1,066.40 11/01/2011, $760.80 12/12/2011
```

72 m104d05b

INTO Clause

Example: Create a macro variable with a list of all customer countries. Delimit the country codes with a comma and a space.

```
proc sql noprint;
   select distinct country
      into :countries separated by ', '
      from orion.customer;
quit;

%put &=Countries;
```

INTO : *macro-variable* SEPARATED BY '*delimiters*'

Partial SAS Log

```
COUNTRIES=AU, CA, DE, IL, TR, US, ZA
```

The SEPARATED BY argument stores multiple values in a single macro variable.

73 m104d05c

The DATA step below creates the same macro variable.

```
proc sort data=orion.customer(keep=country) nodupkey out=allcountries;
   by country;
run;

data _null_;
   set allcountries end=eof;
   length countries $ 50;
   retain countries;
   countries=catx(', ', countries, country);
```

```
   if eof then call symputx('countries', countries);
run;
```

4.06 Multiple Choice Poll

Which technique creates macro variables during execution time?

a. %LET statement
b. SYMPUTX routine
c. INTO clause
d. both b and c

74

SQL Procedure

Example: Display user-defined macro variables in a report.

```
proc sql flow;
   select name, value
      from dictionary.macros
      where scope='GLOBAL'
      order by name;
quit;
```

Partial Output

```
Macro Variable
Name                 Macro Variable Value
-------------------------------------------------------------

AVERAGE              157.49094595
AVG                  $125
COUNTRIES            AU, CA, DE, IL, TR, US, ZA
CUSTID               9
DAT                  01/28/2011
DATE1                06/20/2011
DATE2                11/01/2011
DATE3                12/12/2011
MONTH                1
NAME                 Cornelia Krahl
ORDERS               148
...
```

76 m104d05d

The **dictionary.macros** table is one of several PROC SQL read-only DICTIONARY tables that store SAS metadata. For additional information, see "Accessing SAS System Information Using DICTIONARY Tables" under "Programming with the SQL Procedure" in the *SQL Procedure User's Guide*.

An alternative to the SQL query above is the two-step program below that uses **sashelp.vmacro** in place of **dictionary.macros**.

```
proc sort data=sashelp.vmacro
           (keep=name value scope
           where=(scope='GLOBAL'))
           out=mymacrovariables(keep=name value)
      sortseq=linguistic(numeric_collation=on);
   by name;
run;

proc print data=mymacrovariables;
   title 'Sorted list of global macro variables';
run;
```

SQL Procedure

Example: Create a utility macro to display user-defined macro variables in a report.

```
%macro putALL;
   proc sql flow;
      select name, value
         from dictionary.macros
         where scope='GLOBAL'
         order by name;
   quit;
%mend putALL;
```

```
%putALL
```

77 m104d05d

INTO Clause

Example: Create a macro variable with a list of all user-defined macro variable names. Delimit the names with spaces.

```
proc sql noprint;
   select name into: vars separated by ' '
      from dictionary.macros
      where scope='GLOBAL';
quit;
```

Partial SAS Log

```
705 %put &vars;
SQLOBS SQLOOPS PRICE1 MONTH CUSTID SYS_SQL_IP_ALL DATE1 COUNTRIES
DAT DATE2 YEAR TOTAL DATE3 NAME ORDERS START SQLXOBS SQLRC AVG STOP
AVERAGE SQLEXITCODE PRICE2 PRICE3
```

78

INTO Clause

Example: Create a utility macro that deletes all user-defined macro variables.

```
%macro deleteALL;

   proc sql noprint;
      select name into: vars separated by ' '
         from dictionary.macros
         where scope='GLOBAL';
   quit;

   %symdel &vars;

%mend deleteALL;

%deleteALL
```

79 m104d05e

Exercises

Level 1

7. Creating Macro Variables Using SQL

a. Open the **m104e07** program shown below into the Editor window.

```
%let start=01Jan2011;
%let stop=31Jan2011;

proc means data=orion.order_fact noprint;
   where order_date between "&start"d and "&stop"d;
   var Quantity Total_Retail_Price;
   output out=stats mean=Avg_Quant Avg_Price;
run;

data _null_;
   set stats;
   call symputx('Quant',put(Avg_Quant,4.2));
   call symputx('Price',put(Avg_Price,dollar7.2));
run;

proc print data=orion.order_fact noobs n;
   where order_date between "&start"d and "&stop"d;
   var Order_ID Order_Date Quantity Total_Retail_Price;
   sum Quantity Total_Retail_Price;
   format Total_Retail_Price dollar6.;
   title1 "Report from &start to &stop";
   title3 "Average Quantity: &quant";
   title4 "Average Price: &price";
run;
```

b. Submit the program and view the results.

Report from 01Jan2011 to 31Jan2011

Average Quantity: 2.15
Average Price: $173.89

Order_ID	Order_ Date	Quantity	Total_ Retail_ Price
1241054779	02JAN2011	2	$196
1241063739	03JAN2011	6	$161
1241066216	04JAN2011	2	$306
1241086052	06JAN2011	3	$38
1241147641	13JAN2011	2	$363
1241235281	23JAN2011	1	$73
1241244297	24JAN2011	2	$258
1241244297	24JAN2011	2	$81
1241244297	24JAN2011	3	$358
1241263172	25JAN2011	1	$102
1241263172	25JAN2011	1	$113
1241286432	28JAN2011	2	$174
1241298131	29JAN2011	1	$37
		28	$2,261

N = 13

c. Delete the macro variables **quant** and **price** from the symbol table.

d. Replace the PROC MEANS step and the DATA step with a PROC SQL step.

e. Resubmit the PROC PRINT step and verify that the output is the same.

Level 2

8. Creating a List of Values in a Macro Variable Using SQL

a. Open the **m104e08** program into the Editor window. Create a macro variable named **top3** with the customer ID numbers of the top three customers by **Total_Retail_Price**. Separate the ID numbers with a comma and a blank. Use the OUTOBS= option.

```
proc sql;
   select customer_id, sum(Total_Retail_Price) as total
      from orion.order_fact
      group by Customer_ID
      order by total descending;
quit;
```

The GROUP BY clause summarizes the data by customer ID number.

The ORDER BY clause sorts the data in descending order.

b. Submit the program and review the results, which are shown below.

```
                              Top 3 Customers

          Customer_ID     Customer_Name                Customer_Type

                   10    Karen Ballinger    Orion  Club members high activity
                   16    Ulrich Heyde       Internet/Catalog Customers
                   45    Dianne Patchin     Orion Club Gold members low activity
```

Challenge

9. Creating Multiple Macro Variables Using SQL

a. The **orion.customer_type** data set contains the variable **Customer_Type_ID**, which holds the unique customer type codes. Use the SQL procedure to create a series of macro variables named **CTYPE1** through **CTYPE*xx***, where ***xx*** resolves to the number of rows that the query returns.

You need two queries, one to return the number of rows that the query returns and the other to create **CTYPE1** through **CTYPE*xx***.

b. Open the program **m104e09** to display only the macro variables that begin with **CTYPE**.

```
proc sql;
   select name, value
      from dictionary.macros
      where name like "CTYPE%";
quit;
```

PROC SQL Output

```
             Macro Variable beginning with CTYPE

Macro Variable Name               Macro Variable Value
_____________________________________________________
CTYPE1                            1010
CTYPE2                            1020
CTYPE3                            1030
CTYPE8                            3010
CTYPE4                            1040
CTYPE5                            2010
CTYPE6                            2020
CTYPE7                            2030
```

4.4 Solutions

Solutions to Exercises

1. Creating Macro Variables with the SYMPUTX Routine

a. Open the program into the Editor window.

b. Add a %LET statement before the DATA step to create the macro variable **job** with the value *Audit*. Replace hardcoded values of *Audit* with references to the macro variable **job**. Resubmit the program. It should produce the same output as before.

c. Change the value of **job** to *Analyst*. Resubmit the program and examine the new results.

d. Modify the DATA step to create the macro variable **avg** to store the average salary. Reference the macro variable **avg** in a FOOTNOTE statement. Format the average salary with a dollar sign, comma, and no decimal places.

```
%let job=Analyst;

data staff;
   keep employee_ID job_title salary gender;
   set orion.staff end=last;
   where job_title contains "&job";
   total+salary;
   count+1;
   if last then call symputx('avg', put(total/count,dollar9.));
run;

proc print data=staff;
   sum salary;
   title "&job Staff";
   footnote "Average Salary: &avg";
run;
```

2. Creating Macro Variables with the SYMPUTX Routine

a. Open the program into the Editor window.

b. Create a macro variable named **top** that contains the ID number of the top customer. Then modify the program (part **b**) to print only the orders for Orion's top customer.

```
data _null_;
   set customer_sum (obs=1);
   call symputx('top', Customer_ID);
run;

proc print data=orion.orders noobs;
   where Customer_ID =&top;
   var Order_ID Order_Type Order_Date Delivery_Date;
   title "Orders for Customer &top - Orion's Top Customer";
run;
```

3. Creating Macro Variables with the SYMPUTX Routine

a. Open the program into the Editor window.

b. Using the **customer_sum** data set, create a single macro variable, **top3**, that contains the customer IDs of the top three customers by revenue.

```
data _null_;
   set customer_sum(obs=3) end=last;
   length top3 $50;
   retain top3;
   top3=catx(' ',top3, Customer_ID);
   /* Alternative Solution for the CATX Function    */
   /* top3=trim(top3)||' '||left(Customer_ID);      */
   if  last then call symputx('top3', top3);
run;
```

c. Using the **orion.customer_dim** data set, print a listing of the top three customers.

```
proc print data=orion.customer_dim noobs;
   where Customer_ID in (&top3);
   var Customer_ID Customer_Name Customer_Type;
   title 'Top 3 Customers';
run;
```

4. Creating a Series of Macro Variables with the SYMPUTX Routine

a. Open the program into the Editor window.

b. Concatenating the character value *type* with the value of the **Customer_Type_ID** variable specifies the name of each macro variable. Because the **Customer_Type_ID** variable is numeric, the LEFT function is required to remove the leading blanks introduced by the automatic numeric-to-character conversion that occurs as part of the SYMPUTX routine. The %PUT statement displays the names and values of all user-created macro variables.

c. Because each macro variable that contains the customer type has a common root at the start of its name (**type**) and a suffix that corresponds to the value of the ID macro variable, two ampersands are used in front of the complete reference.

```
data _null_;
   set orion.customer_type;
   call symputx('type'||left(Customer_Type_ID), Customer_Type);
```

```
   *Alternative solution using the CATS function;
   *call symputx(cats('type',Customer_Type_ID), Customer_Type);
run;

%put _user_;

%macro memberlist(id=1020);
   title "A List of &&type&id";
   proc print data=orion.customer;
     var Customer_Name Customer_ID Gender;
     where Customer_Type_ID=&id;
   run;
%mend memberlist;

%memberlist()
```

SAS Log

```
39   %put _user_;
GLOBAL TYPE1010 Orion Club members inactive
GLOBAL TYPE1020 Orion Club members low activity
GLOBAL TYPE2010 Orion Club Gold members low activity
GLOBAL TYPE1030 Orion  Club members medium activity
GLOBAL TYPE2020 Orion Club Gold members medium activity
GLOBAL TYPE3010 Internet/Catalog Customers
                    GLOBAL TYPE1040 Orion  Club members high activity
GLOBAL TYPE2030 Orion Club Gold members high activity

40   options symbolgen;
41   %let id=1020;
42   proc print data=orion.customer;
43       var Customer_Name Customer_ID Gender;
44       where Customer_Type_ID=&id;
SYMBOLGEN:  Macro variable ID resolves to 1020
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable ID resolves to 1020
SYMBOLGEN:  Macro variable TYPE1020 resolves to Orion Club members low activity
45       title "A List of &&type&id";
46   run;

NOTE: There were 17 observations read from the data set ORION.CUSTOMER.
      WHERE Customer_Type_ID=1020;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.17 seconds
      cpu time            0.00 seconds
```

d. Call the macro again, but with a parameter value of *2030*.

```
%memberlist(id=2030)
```

Partial SAS Log

```
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable ID resolves to 2030
SYMBOLGEN:  Macro variable TYPE2030 resolves to Orion Club Gold members high activity
SYMBOLGEN:  Macro variable ID resolves to 2030
```

```
NOTE: There were 10 observations read from the data set ORION.CUSTOMER.
      WHERE Customer_Type_ID=2030;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           1.43 seconds
      cpu time            0.00 seconds
```

5. Using Indirect References in a Macro Call

a. Open the program into the Editor window.

b. Create a macro variable named **num** with the value of *2010*. Execute the macro so that the value of **custtype** resolves to *Orion Club members low activity* in the macro call.

```
%let num=2010;

%memberlist(&&type&num)
```

Partial SAS Log

```
18   %let num=2010;
19   %memberlist(&&type&num)
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable NUM resolves to 2010
SYMBOLGEN:  Macro variable TYPE2010 resolves to Orion Club Gold members low activity
SYMBOLGEN:  Macro variable CUSTTYPE resolves to Orion Club Gold members low activity
SYMBOLGEN:  Macro variable CUSTTYPE resolves to Orion Club Gold members low activity
NOTE: There were 5 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE Customer_Type='Orion Club Gold members low activity';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           2.24 seconds
      cpu time            0.00 seconds
```

6. Using a Table Lookup Application

a. Using **orion.country**, create a series of macro variables in which the name of the macro variable is the country abbreviation (**Country**) and the value of the macro variable is the country name (**Country_Name**).

```
data _null_;
   set orion.country;
   call symputx(Country, Country_Name);
run;
%put _user_;
```

Partial SAS Log

```
639645  %put _user_;
GLOBAL AU Australia
GLOBAL CA Canada
GLOBAL DE Germany
GLOBAL IL Israel
GLOBAL TR Turkey
GLOBAL US United States
GLOBAL ZA South Africa
```

b. Open the program into the Editor window.

c. Use indirect macro variable referencing to replace the ***xxxxx*** with the appropriate country name.

```
%let code=AU;
proc print data=orion.employee_addresses;
   var Employee_Name City;
   where Country="&code";
   title "A List of &&&code Employees";
run;
```

7. Creating Macro Variables Using SQL

a. Open the program into the Editor window.

b. Submit the program and view the results.

c. Delete the macro variables **quant** and **price**.

```
%symdel quant price;
```

d. Replace the PROC MEANS step and the DATA step with a PROC SQL step.

```
proc sql noprint;
   select mean(Quantity) format=4.2,
          mean(Total_Retail_Price) format=dollar7.2
      into :quant, :price
      from orion.order_fact
      where order_date between "&start"d and "&stop"d;
quit;
```

e. Resubmit the PROC PRINT step and verify that the output is the same.

8. Creating a List of Values in a Macro Variable Using SQL

a. Open the **m104e08** program into the Editor window and modify the SQL procedure to create a macro variable named **top3** that contains the customer ID numbers of the top three customers by **Total_Retail_Price** in the **orion.order_fact** data set. Separate each of the values with a comma and a blank. Use the OUTOBS= option to limit the number of output rows.

```
proc sql noprint outobs=3;
   select customer_id, sum(Total_Retail_Price) as total
          into :top3 separated by ', '
      from orion.order_fact
      group by Customer_ID
      order by total descending;
```

b. Submit the program and review the results, which are shown in part **b** of the exercise.

9. Creating Multiple Macro Variables Using SQL

a. The first query creates the **numobs** macro variable that stores how many records are returned by the query. This is the same as the number of macro variables in each series.

A special form of the INTO clause is useful for creating a series of macro variables from multiple rows of an SQL query.

```
proc sql noprint;
   select count(*) into :numobs
      from orion.customer_type;
```

```
      %let numobs=&numobs;
   select Customer_Type_ID into :ctype1-:ctype&numobs
      from orion.customer_type;
quit;
```

b. Submit the program and review the results, which are shown in part **b** of the exercise.

Solutions to Student Activities (Polls/Quizzes)

4.01 Multiple Choice Poll – Correct Answer

What is the value of **foot** after execution of the DATA step?

a. No Internet orders
b. Some Internet orders

```
data _null_;
   call symputx('foot','No Internet orders');
   %let foot=Some Internet orders;
run;
```

1. **Word scanning begins. DATA step compilation begins.**
2. **%LET encountered. Macro trigger executes.**
3. **Step boundary. DATA step executes. SYMPUTX executes.**

18

4.02 Short Answer Poll – Correct Answer

This CALL SYMPUTX statement creates a macro variable named **date**. Complete the call to assign the value of the variable **current**, formatting the result as a date such as 21NOV2012.

```
call symputx('date',put(current,date9.));
```

25

4.03 Short Answer Poll – Correct Answer

How many rows are selected by the DATA step WHERE statement in the preceding program, repeated below?

one row

```
%let custID=9;
data _null_;
   set orion.customer;
   where customer_ID=&custID;
   call symputx('name', Customer_Name);
run;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name";
run;
```

Each time you select a customer number, the DATA step rereads the entire customer data set to select a subset of one customer.

40 m104d03c

4.04 Short Answer Poll – Correct Answer

Modify program **m104a02** to create an order history for **custID 4**. How many program changes are required?

```
%let custID=9;
proc print data=orion.order_fact;
   where customer_ID=&custID;
   var order_date order_type quantity total_retail_price;
   title1 "Customer Number: &custID";
   title2 "Customer Name: &name9";
run;
```

Change

Change

Two program changes are required.

48 m104a02

4.05 Short Answer Poll – Correct Answer

Submit program **m104a03**.

```
%let custid=9;
%let name9=Joe;
%put &name&custid;
```

How many times are the macro variables scanned in the %PUT statement?

One time

Is this successful?

No. It generates the following warning:

```
WARNING: Apparent symbolic reference NAME not resolved.
```

56 m104a03

4.06 Multiple Choice Poll – Correct Answer

Which technique creates macro variables during execution time?

a. %LET statement
b. SYMPUTX routine
c. INTO clause
d. both b and c

75

Chapter 5 Macro Programs

5.1 Conditional Processing

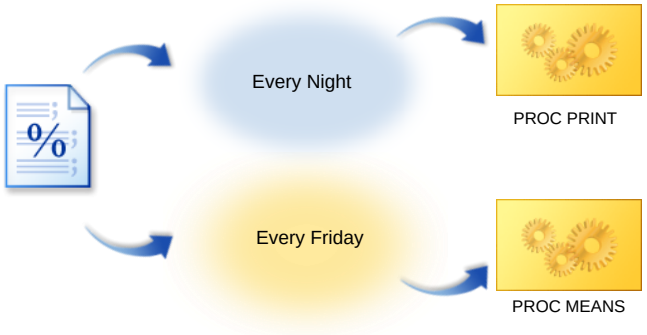
Objectives

- Conditionally process SAS code within a macro program.
- Monitor macro execution.
- Insert entire steps, entire statements, and partial statements into a SAS program.

3

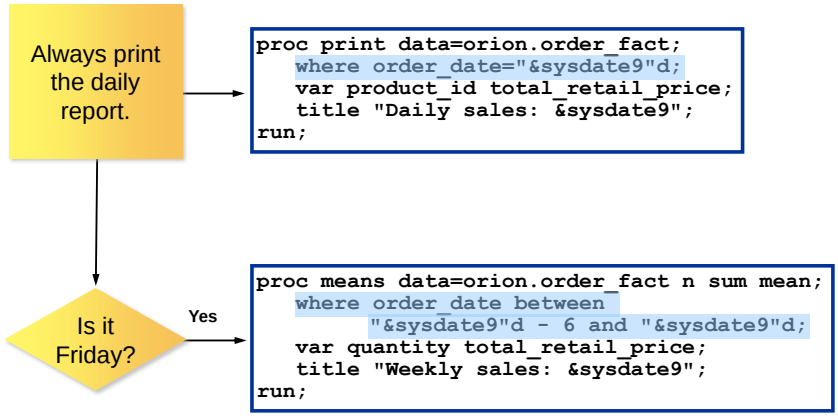
Business Scenario

A daily sales report is generated every night. Every Friday, a weekly report is generated. Determine the best method to automate these reports.



4

Conditional Processing



5

Solutions

Three methods:

Method 1	Create multiple macros, including a driver macro.
Method 2	Create a single macro with %DO and %END statements.
Method 3	Create a single macro with %INCLUDE statements.

6

Processing Complete Steps

Method 1 Create separate macros for the **Daily** and **Weekly** programs.

```
%macro daily;
   proc print data=orion.order_fact;
      where order_date="&sysdate9"d;
      var product_id total_retail_price;
      title "Daily sales: &sysdate9";
   run;
%mend daily;

%macro weekly;
   proc means data=orion.order_fact n sum mean;
      where order_date between
          "&sysdate9"d-6 and "&sysdate9"d;
      var quantity total_retail_price;
      title "Weekly sales: &sysdate9";
   run;
%mend weekly;
```

m105d01a

7

continued...

Processing Complete Steps

Method 1 Create a driver macro that always calls the **Daily** macro and conditionally calls the **Weekly** macro.

```
%macro reports;
   %daily
   %if &sysday=Friday %then %weekly;
%mend reports;
```

%IF *expression* **%THEN** *action*;
%ELSE *action*;

- Character constants are
  - not quoted
  - case sensitive.
- The %ELSE statement is optional.
- %IF-%THEN and %ELSE statements can be used inside a macro definition only.

m105d01a

8

Macro Expressions

%IF *expression*

	Macro Expressions	SAS Expressions
Arithmetic operators	√	√
Logical operators (do not precede AND or OR with %)	√	√
Comparison operators (symbols and mnemonics)	√	√
Case sensitivity	√	√
Special WHERE operators		
Quotation marks		√
Ranges such as 1<=x<=10		√
IN operator	√	√
IN operator: parentheses required		√

9

Special WHERE operators include CONTAINS, IS NULL, IS MISSING, LIKE, BETWEEN-AND, SAME-AND, and =* (sounds like).

Conditional Processing

%IF ... **%THEN** *action*;
%ELSE *action*;

These *actions* can follow keywords %THEN and %ELSE:

- a macro language statement
- a macro variable reference
- a macro call
- any text

10

Monitoring Macro Execution

The MLOGIC system option displays macro execution messages in the SAS log.

Partial SAS Log

```
494  %macro reports;
495     %daily
496     %if &sysday=Friday %then %weekly;
497  %mend reports;
498
499  options mlogic;
500  %reports
MLOGIC(REPORTS):  Beginning execution.
MLOGIC(DAILY):  Beginning execution.
MLOGIC(DAILY):  Ending execution.
MLOGIC(REPORTS):  %IF condition &sysday=Friday is TRUE
MLOGIC(WEEKLY):  Beginning execution.
MLOGIC(WEEKLY):  Ending execution.
MLOGIC(REPORTS):  Ending execution.
```

OPTIONS MLOGIC;

The default setting is NOMLOGIC.

11 m105d01a

The SYMBOLGEN option can be used to debug %IF expressions.

5.01 Short Answer Poll

Submit the program **m105a01**. What error do you see in the log?

```
%macro reports;
   %daily
   %if &sysday=Friday then %weekly;
%mend reports;
```

12

Processing Complete Steps

Method 2 Create a single macro with %DO and %END statements to generate text that contains semicolons.

```
%macro reports;
   proc print data=orion.order_fact;
      where order_date="&sysdate9"d;
      var product_id total_retail_price;
      title "Daily sales: &sysdate9";
   run;
 %if &sysday=Friday %then %do;
   proc means data=orion.order_fact n sum mean;
      where orde
           "&sy                    9"d;
      var quant
      title "We
   run;
 %end;
%mend reports;
```

%IF *expression* **%THEN %DO;**
statement; *statement*;...
%END;
%ELSE %DO;
statement; *statement*;...
%END;

m105d01b

14

Processing Complete Steps

Method 3 Create a single macro with %INCLUDE statements.

%INCLUDE *file-specification* < / SOURCE2 >;

```
%macro reports;
   %include "&path\daily.sas";
   %if &sysday=Friday %then %do;
      %include "&path\weekly.sas";
   %end;
%mend reports;
```

The %INCLUDE statement

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

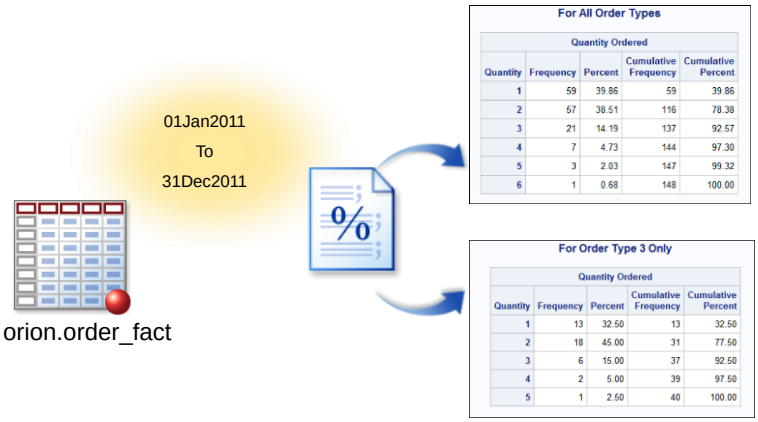
The SOURCE2 option displays inserted SAS statements in the SAS log.

m105d01c

15

Business Scenario

Generate frequency reports for all order types or a specific order type.



16

Processing Complete Statements

Example: Insert individual statements within a PROC step.

```
%macro count(type=,start=01jan2011,stop=31dec2011);
   proc freq data=orion.order_fact;
      where order_date between "&start"d and "&stop"d;
      tables quantity;
      title1 "Orders from &start to &stop";
      %if &type=  %then %do;
         title2 "For All Order Types";
      %end;
      %else %do;
         title2 "For Order Type &type Only";
         where same and order_type=&type;
      %end;
   run;
%mend count;
options mprint mlogic;
%count()
%count(type=3)
```

17

m105d02

Processing Complete Statements

SAS Log

```
802  %count()
MLOGIC(COUNT):  Beginning execution.
MLOGIC(COUNT):  Parameter TYPE has value
MLOGIC(COUNT):  Parameter START has value 01jan2011
MLOGIC(COUNT):  Parameter STOP has value 31dec2011
MPRINT(COUNT):   proc freq data=orion.order_fact;
MPRINT(COUNT):   where order_date between "01jan2011"d and "31dec2011"d;
MPRINT(COUNT):   tables quantity;
MPRINT(COUNT):   title1 "Orders from 01jan2011 to 31dec2011";
MLOGIC(COUNT):  %IF condition &type= is TRUE
MPRINT(COUNT):   title2 "For All Order Types";
MPRINT(COUNT):   run;

NOTE: There were 148 observations read from the data set ORION.ORDER_FACT.
      WHERE (order_date>='01JAN2011'D and order_date<='31DEC2011'D);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

MLOGIC(COUNT):  Ending execution.
```

18 m105d02

Processing Complete Statements

SAS Log

```
803  %count(type=3)
MLOGIC(COUNT):  Beginning execution.
MLOGIC(COUNT):  Parameter TYPE has value 3
MLOGIC(COUNT):  Parameter START has value 01jan2011
MLOGIC(COUNT):  Parameter STOP has value 31dec2011
MPRINT(COUNT):   proc freq data=orion.order_fact;
MPRINT(COUNT):   where order_date between "01jan2011"d and "31dec2011"d;
MPRINT(COUNT):   tables quantity;
MPRINT(COUNT):   title1 "Orders from 01jan2011 to 31dec2011";
MLOGIC(COUNT):  %IF condition &type= is FALSE
MPRINT(COUNT):   title2 "For Order Type 3 only";
MPRINT(COUNT):   where same and order_type=3;
NOTE: Where clause has been augmented.
MPRINT(COUNT):   run;

NOTE: There were 40 observations read from the data set ORION.ORDER_FACT.
      WHERE (order_date>='01JAN2011'D and order_date<='31DEC2011'D) and
      (order_type=3);
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

MLOGIC(COUNT):  Ending execution.
```

19 m105d02

SAME-AND is a WHERE statement operator that supplements an existing WHERE statement without respecifying the original WHERE statement.

Processing Complete Statements

Example: Insert individual statements within a DATA step.

```
%macro cust(place);
  %let place=%upcase(&place);
  data customers;
    set orion.customer;
    %if &place=US %then %do;
        where country='US';
        keep customer_name customer_address country;
    %end;
    %else %do;
        where country ne 'US';
        keep customer_name customer_address country location;
        length location $ 12;
        if      country="AU" then location='Australia';
        else if country="CA" then location='Canada';
        else if country="DE" then location='Germany';
        else if country="IL" then location='Israel';
        else if country="TR" then location='Turkey';
        else if country="ZA" then location='South Africa';
    %end;
  run;
%mend cust;
%cust(us)
%cust(international)
```

20 m105d03

The %UPCASE function converts values to uppercase.

%UPCASE(*argument***)**

Because macro comparisons are case sensitive, %UPCASE can eliminate case sensitivity when a macro program checks a parameter value.

Processing Complete Statements

SAS Log

```
828  %cust(us)
MLOGIC(CUST):  Beginning execution.
MLOGIC(CUST):  Parameter PLACE has value us
MLOGIC(CUST):  %LET (variable name is PLACE)
MPRINT(CUST):   data customers;
MPRINT(CUST):   set orion.customer;
MLOGIC(CUST):  %IF condition &place=US is TRUE
MPRINT(CUST):   where country='US';
MPRINT(CUST):   keep customer_name customer_address country;
MPRINT(CUST):   run;

NOTE: There were 28 observations read from the data set ORION.CUSTOMER.
      WHERE country='US';
NOTE: The data set WORK.CUSTOMERS has 28 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

MLOGIC(CUST):  Ending execution.
```

21 m105d03

Processing Complete Statements

SAS Log

```
829  %cust(international)
MLOGIC(CUST):  Beginning execution.
MLOGIC(CUST):  Parameter PLACE has value international
MLOGIC(CUST):  %LET (variable name is PLACE)
MPRINT(CUST):   data customers;
MPRINT(CUST):   set orion.customer;
MLOGIC(CUST):  %IF condition &place=US is FALSE
MPRINT(CUST):   where country ne 'US';
MPRINT(CUST):   keep customer_name customer_address country location;
MPRINT(CUST):   length location $ 12;
MPRINT(CUST):   if country="AU" then location='Australia';
MPRINT(CUST):   else if country="CA" then location='Canada';
MPRINT(CUST):   else if country="DE" then location='Germany';
MPRINT(CUST):   else if country="IL" then location='Israel';
MPRINT(CUST):   else if country="TR" then location='Turkey';
MPRINT(CUST):   else if country="ZA" then location='South Africa';
MPRINT(CUST):   run;

NOTE: There were 49 observations read from the data set ORION.CUSTOMER.
      WHERE country not = 'US';
NOTE: The data set WORK.CUSTOMERS has 49 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

MLOGIC(CUST):  Ending execution.
```

22 m105d03

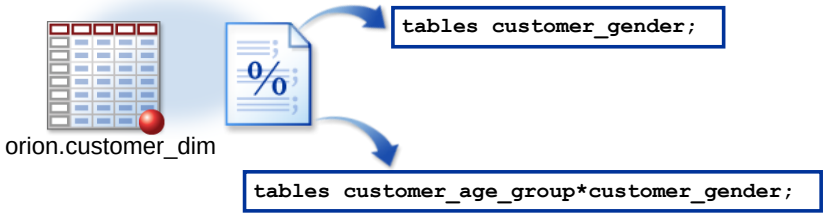
Idea Exchange

What is the difference between %IF-%THEN and IF-THEN?

23

Business Scenario

Create a macro that conditionally generates a one-way frequency report or a two-way frequency report.



25

Processing Partial Statements

Conditionally insert text into the middle of a statement.

Example: Generate either a one-way or two-way frequency table, depending on parameter values.

```
%macro counts(rows);
   title 'Customer Counts by Gender';
   proc freq data=orion.customer_dim;
      tables
   %if &rows ne  %then &rows *;
      customer_gender;
   run;
%mend counts;

%counts()
%counts(customer_age_group)
```

26 m105d04

Processing Partial Statements

SAS Log

```
1798  %counts()
MPRINT(COUNTS):   title 'Customer Counts by Gender';
MPRINT(COUNTS):   proc freq data=orion.customer_dim;
MPRINT(COUNTS):   tables customer_gender ;
MPRINT(COUNTS):   run;

NOTE: There were 77 observations read from the data set ORION.CUSTOMER_DIM.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

1799  %counts(customer_age_group)
MPRINT(COUNTS):   title 'Customer Counts by Gender';
MPRINT(COUNTS):   proc freq data=orion.customer_dim;
MPRINT(COUNTS):   tables customer_age_group * customer_gender ;
MPRINT(COUNTS):   run;

NOTE: There were 77 observations read from the data set ORION.CUSTOMER_DIM.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

27 m105d04

Exercises

Level 1

1. Conditionally Processing Complete Statements

a. Open the **m105e01** program shown below into the Editor window.

```
%macro listing(custtype);
   proc print data=orion.customer noobs;
   run;
%mend listing;
%listing(2010)
```

b. Modify the macro to test the CUSTTYPE parameter. If the value is null, insert the following statements into the PROC PRINT step:

```
var Customer_ID Customer_Name Customer_Type_ID;
title "All Customers";
```

If the value is not null, insert the following statements into the PROC PRINT step:

```
where Customer_Type_ID=&custtype;
var Customer_ID Customer_Name;
title "Customer Type: &custtype";
```

c. Resubmit the macro definition and call the macro using a null value and a valid value of CUSTTYPE.

Level 2

2. Debugging a Macro

a. Open the **m105e02** program shown below into the Editor window.

```
%macro day;
   %if &sysday=SATURDAY
      %then %put Yes;
      %else %put Sorry;
%mend day;

options nomlogic nosymbolgen ;

%day
```

b. Change SATURDAY to today's value and submit the program.

c. Did the log say Yes? If not, take the following steps, in sequence, until it does.

1) Activate the MLOGIC option, resubmit the program, check the log, and change the day as necessary.

2) Activate the SYMBOLGEN option, resubmit the program, check the log, and change the day as necessary.

Challenge

3. Debugging a Macro

a. Open the **m105e03** program shown below into the Editor window.

```
%macro where(state);
   %if &state=OR
      %then %put Oregon;
      %else %put Wherever;
%mend where;

%where(CA)
```

b. Submit the program. Examine the log and correct the error.

5.2 Parameter Validation

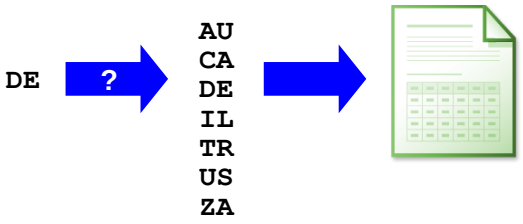
Objectives

- Perform parameter validation with the OR operator.
- Perform parameter validation with the IN operator.
- Perform data-driven parameter validation.

31

Business Scenario

Validate a parameter against a list of valid **COUNTRY** values. Explore three possible methods.



32

Parameter Validation

Method 1 Use OR operators for parameter validation.

```
%macro customers(place);
   %let place=%upcase(&place);
   %if &place=AU
   or  &place=CA
   or  &place=DE
   or  &place=IL
   or  &place=TR
   or  &place=US
   or  &place=ZA %then %do;
       proc print data=orion.customer;
          var customer_name customer_address country;
          where upcase(country)="&place";
          title "Customers from &place";
       run;
   %end;
   %else %put ERROR: No customers from &place..;
%mend customers;
%customers(de)
%customers(aa)
```

33 m105d05a

Parameter Validation

SAS Log

```
955  %customers(de)

NOTE: There were 10 observations read from the data set ORION.CUSTOMER.
      WHERE UPCASE(country)='DE';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

956  %customers(aa)
ERROR: No customers from AA.
```

34 m105d05a

5.02 Short Answer Poll

Instead of using multiple OR operators, which SAS comparison operator compares a value to a list of values?

```
if Country='FR'
or Country='CA'
or Country='DE'
or Country='ZA' then do;
```

35

Parameter Validation

Method 2 Use the IN operator for parameter validation.

```
%macro customers(place) / minoperator;
   %let place=%upcase(&place);
   %if &place in AU CA DE IL TR US ZA %then %do;
      proc print data=orion.customer;
         var customer_name customer_address country;
         where upcase(country)="&place";
         title "Customers from &place";
      run;
   %end;
   %else %put Sorry, no customers from &place..;
%mend customers;

%customers(de)
%customers(aa)
```

- The MINOPERATOR option is required.

37 m105d05b

- The macro IN operator does not require parentheses.

- When using NOT with the IN operator, NOT must precede the IN expression. Parentheses are required, as shown below.

```
%if not(&macvar in &valuelist) %then … ;
```

5.03 Short Answer Poll

What can be done if the value list is extremely long or changes frequently, or both?

```
%macro customers(place) / minoperator;
   %let place=%upcase(&place);
   %if &place in AU CA DE IL TR US ZA %then %do;
      proc print data=orion.customer;
         var customer_name customer_address country;
         where upcase(country)="&place";
         title "Customers from &place";
      run;
   %end;
   %else %put Sorry, no customers from &place..;
%mend customers;

%customers(de)
%customers(aa)
```

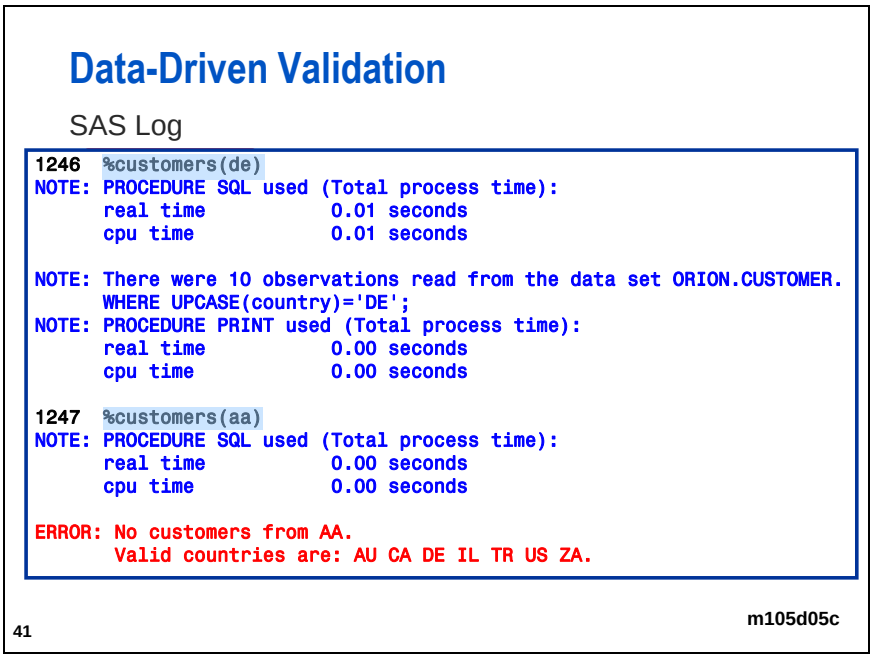
38 m105d05b

Data-Driven Validation

Method 3 Use data-driven parameter validation.

```
%macro customers(place) / minoperator;
   %let place=%upcase(&place);
   proc sql noprint;
      select distinct country into :list separated by ' '
             from orion.customer;
   quit;
   %if &place in &list %then %do;
      proc print data=orion.customer;
         var customer_name customer_address country;
         where upcase(country)="&place";
         title "Customers from &place";
      run;
   %end;
   %else %do;
      %put ERROR: No customers from &place..;
      %put ERROR- Valid countries are: &list..;
   %end;
%mend customers;
```

40 m105d05c



Exercises

Level 1

4. Validating a Macro Parameter

a. Open the **m105e04** program shown below into the Editor window and submit it.

```
%macro custtype(type);
   %let type=%upcase(&type);
    proc print data=orion.customer_dim;
       var Customer_Group Customer_Name Customer_Gender
           Customer_Age;
       where upcase(Customer_Group) contains "&type";
       title "&type Customers";
    run;
%mend custtype;
%custtype(internet)
```

b. Modify the macro to use %IF-%THEN and %ELSE statements to validate the TYPE parameter. The macro should submit the PROC PRINT step only if the TYPE parameter is *GOLD* or *INTERNET*. If the TYPE parameter is not correct, the macro should write this message to the SAS log:

```
ERROR: Invalid TYPE: xxxx.
ERROR: Valid TYPE values are INTERNET or GOLD.
```

The value *xxxx* is the TYPE parameter.

Be sure to set the appropriate option to activate the IN operator.

c. Resubmit the macro definition and call the macro using valid and invalid parameter values.

d. Modify the macro to test whether TYPE is null. If so, do not execute PROC PRINT. Instead, the macro should write this message to the SAS log:

```
ERROR: Missing TYPE.
ERROR: Valid values are INTERNET or GOLD.
```

The macro should first check for a null parameter value. If TYPE is not null, the macro should convert TYPE to uppercase and test for valid values of *GOLD* or *INTERNET*.

e. Resubmit the macro definition with a null parameter value, valid values in uppercase, lowercase, and mixed case, and an invalid value.

Level 2

5. Validating a Macro Parameter Using Data-Driven Techniques

a. Open the **m105e05** program shown below into the Editor window and submit it.

```
%macro listing(custtype);
   %if &custtype= %then %do;
      proc print data=orion.customer noobs;
         var Customer_ID Customer_Name Customer_Type_ID;
         title "All Customers";
      run;
   %end;
   %else %do;
       proc print data=orion.customer noobs;
          where Customer_Type_ID=&custtype;
          var Customer_ID Customer_Name;
          title "Customer Type: &custtype";
       run;
   %end;
%mend listing;

%listing(1020)
%listing()
```

b. Modify the macro definition to validate CUSTTYPE against a data-driven list.

1) Use the SQL procedure to create the macro variable **idlist** that contains the unique values of the variable **Customer_Type_ID** in the **orion.customer_type** data set.

2) If the CUSTTYPE parameter is not missing, validate it against **idlist**.

The macro IN operator cannot check for a null value. Check that a value is not null before using the IN operator to check whether the value is valid.

3) If CUSTTYPE is in **idlist**, execute PROC PRINT. Otherwise, do not execute PROC PRINT. Instead, the macro should write this message to the SAS log:

Partial SAS Log

```
ERROR: Value for CUSTTYPE is invalid.
       Valid values are 1010 1020 1030 1040 2010 2020 2030 3010
```

c. Resubmit the macro definition and call the macro with a null parameter value, a valid value, and an invalid value.

Challenge

6. Validating a Macro Parameter

a. Open the **m105e06** program shown below into the Editor window and submit it.

```
%macro salarystats(decimals=2,order=internal);
   options nolabel;
   title 'Salary Stats';
   proc means data=orion.staff maxdec=&decimals order=&order;
      where job_title contains 'Sales';
      var salary;
      class job_title;
   run;
title;
%mend salarystats;
```

b. Modify the macro to validate parameters according to the following requirements:

Macro Variable	Possible Values
decimals	*0* to *4*
order	*INTERNAL*, *FREQ*

1) Create an additional macro variable named **numerrors** that accumulates the number of parameter errors.

2) Execute PROC MEANS only if **numerrors** is zero.

To express NOT IN, use the syntax below.

```
%IF NOT(&macvar IN &value-list) %THEN … ;
```

c. The macro should write the following messages to the SAS log when parameters are invalid:

Partial SAS Log

```
211  %salarystats(decimals=5,order=fudge)
ERROR: Invalid DECIMALS parameter: 5.
       Valid DECIMALS values are 0 to 4.
ERROR: Invalid ORDER parameter: FUDGE.
       Valid ORDER values are INTERNAL or FREQ.
ERROR: 2 errors.  Code not submitted.
```

5.3 Iterative Processing

Objectives

- Execute macro language statements iteratively.
- Generate SAS code iteratively.

45

Business Scenario

Macro applications might require iterative processing.

The iterative %DO statement can execute macro language statements and generate SAS code.

46

SYMPUTX Routine (Review)

Example: Create a numbered series of macro variables.

```
data _null_;
   set orion.country end=no_more;
   call symputx('Country'||left(_n_),country_name);
   if no_more then call symputx('numrows',_n_);
run;
```

SAS Log

```
639680  %put _user_;
GLOBAL COUNTRY1 Australia
GLOBAL COUNTRY2 Canada
GLOBAL COUNTRY3 Germany
GLOBAL COUNTRY4 Israel
GLOBAL COUNTRY5 Turkey
GLOBAL COUNTRY6 United States
GLOBAL COUNTRY7 South Africa
GLOBAL NUMROWS 7
```

47 m105d06

The PROC SQL step below creates the same series of macro variables. In SAS 9.3, you can use a hyphen in the INTO clause to specify a range without an upper bound.

```
proc sql noprint;
   select country_name into :country1-
     from orion.country;
   %let numrows=&sqlobs;
quit;
```

The automatic macro variable **SQLOBS** is populated with the number of rows processed by the SELECT statement.

Simple Loops

Example: Display a series of macro variables in the SAS log by repeatedly executing %PUT within a macro loop.

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

%DO *index-variable*=*start* **%TO** *stop* <**%BY** *increment*>**;**
text
%END;

SAS Log

```
200  %putloop
Country1 is Australia
Country2 is Canada
Country3 is Germany
Country4 is Israel
Country5 is Turkey
Country6 is United States
Country7 is South Africa
```

48 m105d07

No code is sent to the compiler when the macro executes. The %PUT statements are executed by the macro processor.

Simple Loops

%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 created in the local symbol table if it does not already exist in another symbol table.
- *start*, *stop*, and *increment* values can be any valid macro expressions that resolve to integers.
- The %BY clause is optional. (The default *increment* is 1.)
- *text* can be any of the following:
  - constant text
  - macro variables or expressions
  - macro statements
  - macro calls

49

5.04 Multiple Choice Poll

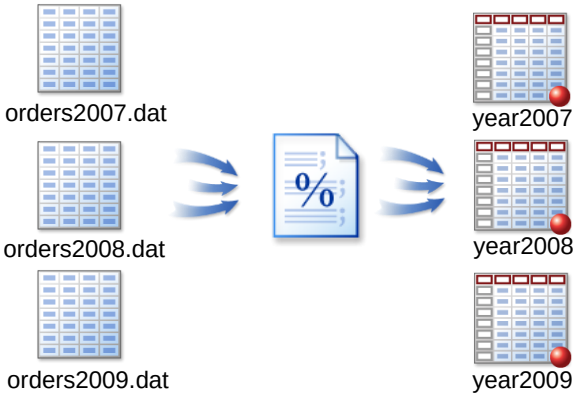
Which statement correctly creates an index variable named **i**?

a. %do &i=1 %to 10;
b. %do &i=1 to 10;
c. %do i=1 %to 10;
d. %do i=1 to 10;

50

Business Scenario

Import a series of text files into corresponding SAS data sets.



52

Generating Repetitive Code

Example: Iteratively generate complete SAS steps.

```
%macro readraw(first=2007,last=2011);
  %do year=&first %to &last;
    data year&year;
      infile "&path\orders&year..dat";
      input order_ID order_type order_date : date9.;
    run;
  %end;
%mend readraw;

options mlogic mprint;

%readraw(first=2008,last=2010)
```

53 m105d08

Generating Repetitive Code

Partial SAS Log

```
MLOGIC(READRAW):  %DO loop index variable YEAR is now 2009; loop will iterate again.
MPRINT(READRAW):   data year2009;
MPRINT(READRAW):   infile "s:\workshop\orders2009.dat";
MPRINT(READRAW):   input order_ID order_type order_date : date9.;
MPRINT(READRAW):   run;

NOTE: The infile "s:\workshop\orders2009.dat" is:
      File Name=s:\workshop\orders2009.dat,
      RECFM=V,LRECL=256

NOTE: 70 records were read from the infile "s:\workshop\orders2009.dat".
      The minimum record length was 22.
      The maximum record length was 22.
NOTE: The data set WORK.YEAR2009 has 70 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

MLOGIC(READRAW):  %DO loop index variable YEAR is now 2010; loop will iterate again.
MPRINT(READRAW):   data year2010;
MPRINT(READRAW):   infile "s:\workshop\orders2010.dat";
MPRINT(READRAW):   input order_ID order_type order_date : date9.;
MPRINT(READRAW):   run;
```

54 m105d08

Generating Data-Dependent Code

Example: Create a separate data set for each value of a selected variable in a selected data set.

```
%split(data=orion.customer, var=country)
```

Partial SAS Log

```
MPRINT(SPLIT):   data AU CA DE IL TR US ZA ;
MPRINT(SPLIT):   set orion.customer;
MPRINT(SPLIT):   select(country);
MPRINT(SPLIT):   when("AU") output AU;
MPRINT(SPLIT):   when("CA") output CA;
MPRINT(SPLIT):   when("DE") output DE;
MPRINT(SPLIT):   when("IL") output IL;
MPRINT(SPLIT):   when("TR") output TR;
MPRINT(SPLIT):   when("US") output US;
MPRINT(SPLIT):   when("ZA") output ZA;
MPRINT(SPLIT):   otherwise;
MPRINT(SPLIT):   end;
MPRINT(SPLIT):   run;
```

55

Generating Data-Dependent Code

Step 1 Store unique data values into macro variables.

```
%macro split (data=, var=);
  proc sort data=&data(keep=&var) out=values nodupkey;
     by &var;
  run;
  data _null_;
     set values end=last;
     call symputx('val'||left(_n_),&var);
     if last then call symputx('count',_n_);
  run;
  %put _local_;
%mend split;

%split(data=orion.customer, var=country)
```

56 m105d09a

Values of the selected variable must represent valid SAS names. The NOTNAME function detects the position of invalid SAS name characters within a character string.

The statement below writes a macro's local symbol table to the SAS log.

```
%put _local_;
```

Generating Data-Dependent Code

Partial SAS Log

```
SPLIT COUNT 7
SPLIT DATA orion.customer
SPLIT VAL1 AU
SPLIT VAL2 CA
SPLIT VAL3 DE
SPLIT VAL4 IL
SPLIT VAL5 TR
SPLIT VAL6 US
SPLIT VAL7 ZA
SPLIT VAR country
```

57 m105d09a

Generating Data-Dependent Code

Step 2 Use loops to generate the DATA step.

```
%macro split (data=, var=);
  proc sort data=&data(keep=&var) out=values nodupkey;
     by &var;
  run;
  data _null_;
     set values end=last;
     call symputx('val'||left(_n_),&var);
     if last then call symputx('count',_n_);
  run;
  data
     %do i=1 %to &count;
        &&val&i
     %end;
  ;
     set &data;
        select(&var);
        %do i=1 %to &count;
           when("&&val&i") output &&val&i;
        %end;
        otherwise;
        end;
  run;
%mend split;
```

58 m105d09b

Generating Data-Dependent Code

Partial SAS Log

```
MPRINT(SPLIT):   data AU CA DE IL TR US ZA ;
MPRINT(SPLIT):   set orion.customer;
MPRINT(SPLIT):   select(country);
MPRINT(SPLIT):   when("AU") output AU;
MPRINT(SPLIT):   when("CA") output CA;
MPRINT(SPLIT):   when("DE") output DE;
MPRINT(SPLIT):   when("IL") output IL;
MPRINT(SPLIT):   when("TR") output TR;
MPRINT(SPLIT):   when("US") output US;
MPRINT(SPLIT):   when("ZA") output ZA;
MPRINT(SPLIT):   otherwise;
MPRINT(SPLIT):   end;
MPRINT(SPLIT):   run;

NOTE: There were 77 observations read from the data set ORION.CUSTOMER.
NOTE: The data set WORK.AU has 8 observations and 12 variables.
NOTE: The data set WORK.CA has 15 observations and 12 variables.
NOTE: The data set WORK.DE has 10 observations and 12 variables.
NOTE: The data set WORK.IL has 5 observations and 12 variables.
NOTE: The data set WORK.TR has 7 observations and 12 variables.
NOTE: The data set WORK.US has 28 observations and 12 variables.
NOTE: The data set WORK.ZA has 4 observations and 12 variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.10 seconds
```

59 m105d09b

5.05 Short Answer Poll

Given the symbol table below, what is the value of &&VAL&COUNT?

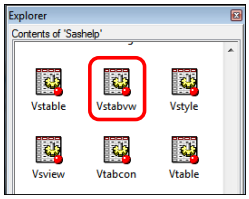
```
SPLIT COUNT 7
SPLIT DATA orion.customer
SPLIT VAL1 AU
SPLIT VAL2 CA
SPLIT VAL3 DE
SPLIT VAL4 IL
SPLIT VAL5 TR
SPLIT VAL6 US
SPLIT VAL7 ZA
SPLIT VAR country
```

60

Business Scenario

Print every data set in a library.

```
%printlib(lib=orion)
```



sashelp.vstabvw

libname	memname	memtype
ORION	CITY	DATA
ORION	CONTINENT	DATA
ORION	COUNTRY	DATA
ORION	COUNTY	DATA
ORION	CUSTOMER	DATA
ORION	CUSTOMER_DIM	DATA
ORION	CUSTOMER_TYPE	DATA
ORION	DISCOUNT	DATA
ORION	EMPLOYEE_ADDRESSES	DATA
ORION	EMPLOYEE_PAYROLL	DATA
ORION	LOOKUP_ORDER_TYPE	DATA
ORION	ORDERS	DATA
ORION	ORDER_FACT	DATA
ORION	ORDER_ITEM	DATA
ORION	ORGANIZATION_DIM	DATA
ORION	PRODUCT_DIM	DATA
ORION	STAFF	DATA
ORION	STATE	DATA

62

Generating Data-Dependent Code

SAS metadata is available in the **sashelp.vstabvw** dynamic view.

```
proc print data=sashelp.vstabvw;
   where libname="ORION";
   title "SASHELP.VSTABVW";
run;
```

Partial PROC PRINT Output

SASHELP.VSTABVW

Obs	libname	memname	memtype
336	ORION	CITY	DATA
337	ORION	CONTINENT	DATA
338	ORION	COUNTRY	DATA
339	ORION	COUNTY	DATA
340	ORION	CUSTOMER	DATA
341	ORION	CUSTOMER_DIM	DATA

63 m105d10

Generating Data-Dependent Code

Step 1 Store data set names into macro variables.

```
%macro printlib(lib=WORK);
  %let lib=%upcase(&lib);
  data _null_;
    set sashelp.vstabvw end=final;
    where libname="&lib";
    call symputx('dsn'||left(_n_),memname);
    if final then call symputx('totaldsn',_n_);
  run;
  %put _local_;
%mend printlib;

%printlib(lib=orion)
```

64 m105d11a

The statement below writes a macro's local symbol table to the SAS log.

```
%put _local_;
```

Generating Data-Dependent Code

Partial SAS Log

```
PRINTLIB DSN1 AGESMOD
PRINTLIB DSN10 COUPONS
PRINTLIB DSN11 CUSTOMER
PRINTLIB DSN12 CUSTOMERDIM
PRINTLIB DSN13 CUSTOMERDIMMORE
PRINTLIB DSN14 CUSTOMERTYPE
PRINTLIB DSN15 CUSTOMER_DIM
PRINTLIB DSN16 CUSTOMER_TYPE
PRINTLIB DSN17 DAILY_SALES
PRINTLIB DSN18 DISCOUNT
PRINTLIB DSN19 EMPLOYEEADDRESSES
PRINTLIB DSN2 BIZLIST
PRINTLIB DSN20 EMPLOYEEDONATIONS
PRINTLIB DSN21 EMPLOYEEORGANIZATION
PRINTLIB DSN22 EMPLOYEEPAYROLL
PRINTLIB DSN23 EMPLOYEEPHONES
PRINTLIB DSN24 EMPLOYEE_PAYROLL
PRINTLIB DSN25 EUROPECUSTOMERS
```

65 m105d11a

Generating Data-Dependent Code

Step 2 Use a macro loop to print every data set in the library.

```
%macro printlib(lib=WORK,obs=5);
  %let lib=%upcase(&lib);
  data _null_;
    set sashelp.vstabvw end=final;
    where libname="&lib";
    call symputx('dsn'||left(_n_),memname);
    if final then call symputx('totaldsn',_n_);
  run;
  %do i=1 %to &totaldsn;
    proc print data=&lib..&&dsn&i(obs=&obs);
       title "&lib..&&dsn&i Data Set";
    run;
  %end;
%mend printlib;
%printlib(lib=orion)
```

66 m105d11b

Generating Data-Dependent Code

Partial SAS Log

```
MPRINT(PRINTLIB):   proc print data=ORION.CUSTOMER(obs=5);
MPRINT(PRINTLIB):   title "ORION.CUSTOMER Data Set";
MPRINT(PRINTLIB):   run;
NOTE: There were 5 observations read from the data set ORION.CUSTOMER.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MPRINT(PRINTLIB):   proc print data=ORION.CUSTOMER_DIM(obs=5);
MPRINT(PRINTLIB):   title "ORION.CUSTOMER_DIM Data Set";
MPRINT(PRINTLIB):   run;
NOTE: There were 5 observations read from the data set ORION.CUSTOMER_DIM.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MPRINT(PRINTLIB):   proc print data=ORION.CUSTOMER_TYPE(obs=5);
MPRINT(PRINTLIB):   title "ORION.CUSTOMER_TYPE Data Set";
MPRINT(PRINTLIB):   run;
NOTE: There were 5 observations read from the data set ORION.CUSTOMER_TYPE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

67 m105d11b

Exercises

Level 1

7. Using Macro Loops and Indirect References

a. Open the **m105e07** program shown below into the Editor window.

```
proc means data=orion.order_fact sum mean maxdec=2;
   where Order_Type=1;
   var Total_Retail_Price CostPrice_Per_Unit;
   title "Order Type: 1";
run;
```

b. Define a macro that generates a separate PROC MEANS step for each of the order types in the **orion.order_fact** data set. The values of **Order_Type** are *1*, *2*, and *3*.

c. The **orion.lookup_order_type** data set contains the variable **START**, which represents each order type, and the variable **LABEL**, which describes each order type.

Partial Listing of **orion.lookup_order_type**

```
                                  START        LABEL

                                    1      Retail Sale
                                    2      Catalog Sale
                                    3      Internet Sale
```

Modify the following DATA step to create a series of macro variables named **type1**-**type*n***, where *n* is the number of observations in the **orion.lookup_order_type** data set. Each macro variable should receive the value of the data set variable **LABEL**. Place the modified DATA step into the macro definition.

```
data _null_;
   set orion.lookup_order_type;
run;
```

c. Modify the macro to do the following:

- create a macro variable **endloop** and populate it with the number of observations in the **orion.lookup_order_type** data set
- use the **endloop** macro variable as the stop value for the macro DO loop
- use an indirect reference to **type*x*** in the TITLE statement

Level 2

8. Generating Data-Dependent Steps

a. Open the **m105e08** program shown below into the Editor window. This program creates a summary data set named **customer_freq** that summarizes the variable **Total_Retail_Price** by **Customer_ID**, and then sorts by descending **sum**. CALL SYMPUTX creates a series of macro variables named **top1** through **top*x***, where *x* is the value of the OBS parameter.

```
%macro tops(obs=3);
    proc means data=orion.order_fact sum nway noprint;
       var Total_Retail_Price;
       class Customer_ID;
       output out=customer_freq sum=sum;
    run;

    proc sort data=customer_freq;
       by descending sum;
    run;
```

```
    data _null_;
       set customer_freq(obs=&obs);
       call symputx('top'||left(_n_), Customer_ID);
    run;
%mend tops;

%tops()
%tops(obs=5)
```

b. Modify the macro to print a listing of the top *x* customers from the **orion.customer_dim** data set. Display the variables **Customer_ID**, **Customer_Name**, and **Customer_Type**. Use a macro loop to dynamically generate values for the WHERE statement based on the macro variables **top1** through **top*x***.

Partial SAS Log

```
MPRINT(TOPS):   proc print data=orion.customer_dim noobs;
MPRINT(TOPS):   where Customer_ID in ( 16 10 45);
MPRINT(TOPS):   var Customer_ID Customer_Name Customer_Type;
MPRINT(TOPS):   title "Top 3 Customers";
MPRINT(TOPS):   run;
```

```
MPRINT(TOPS):   proc print data=orion.customer_dim noobs;
MPRINT(TOPS):   where Customer_ID in ( 16 10 45 2806 195);
MPRINT(TOPS):   var Customer_ID Customer_Name Customer_Type;
MPRINT(TOPS):   title "Top 5 Customers";
MPRINT(TOPS):   run;
```

Challenge

9. Generating Multiple Macro Calls

a. Open the **m105e09** program shown below into the Editor window. Submit the program and review the result.

```
%macro memberlist(custtype);
   proc print data=Orion.Customer_dim;
      var Customer_Name Customer_ID Customer_Age_Group;
      where Customer_Type="&custtype";
      title "A List of &custtype";
   run;
%mend memberlist;

%macro listall;
   data _null_;
      set orion.customer_type end=final;
      call symputx('type'||left(_n_), Customer_Type);
      if final then call symputx('n',_n_);
   run;
   %put _user_;
%mend listall;

%listall
```

b. Modify the **Listall** macro to call the **Memberlist** macro. The result of the macro call should create a PROC PRINT step for each customer type. Use a macro loop and indirect references to generate the appropriate macro calls.

5.4 Global and Local Symbol Tables

Objectives

- Explain the difference between global and local symbol tables.
- Describe how the macro processor decides which symbol table to use.
- Describe the concept of nested macros and the hierarchy of symbol tables.

71

Global Symbol Table (Review)

The *global symbol table* is

- created during SAS initialization
- initialized with automatic macro variables
- deleted at the end of the session.

72

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

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 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

Use local variables instead of global variables whenever possible. Memory used by a local table can be reused when the table is deleted following macro execution.

Local Symbol Table

Variable	Value
parameter1	**value1**
parameter2	**value2**
.	.
.	.
.	.
uservar1	**value1**
uservar2	**value2**

77

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

The DATA step SYMPUT routine can also create local macro variables, but it does not remove leading and trailing blanks from the macro variable's value.

%LOCAL Statement

General form of the %LOCAL statement:

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

- The %LOCAL statement adds one or more macro variables to the local symbol table with null values.
- It has no effect on variables already in the local table.
- It can appear only inside a macro definition.

79

%LOCAL Statement

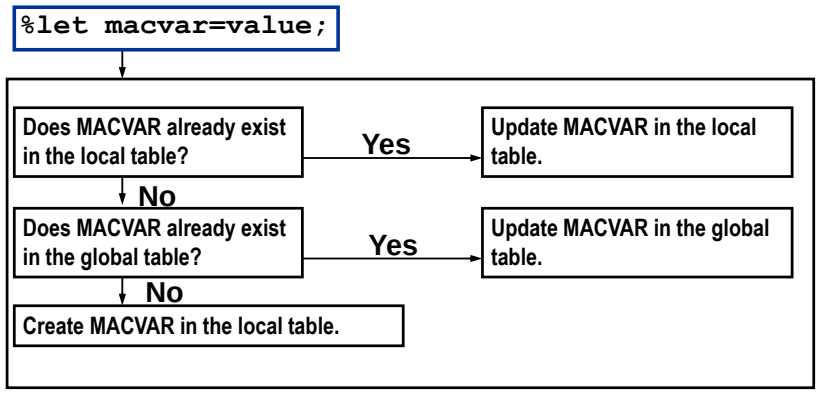
Declare the index variable of a macro loop as a local variable to prevent accidental contamination of a like-named macro variable in the global table or another local table.

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

80 m105d13

Rules for Creating and Updating Variables

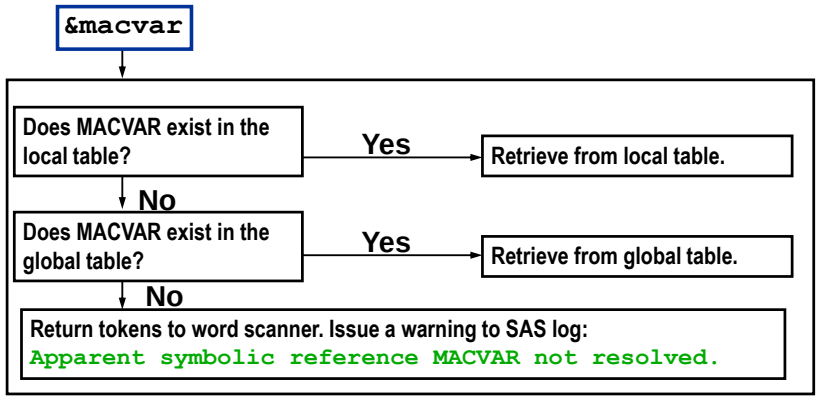
When the macro processor receives a request to create or update a macro variable during macro execution, the macro processor follows these rules:



81

Rules for Resolving Variables

To resolve a macro variable reference during macro execution, the macro processor follows these rules:



82

5.06 Short Answer Poll

How many local symbol tables are created when macro **A** is called and begins to execute?

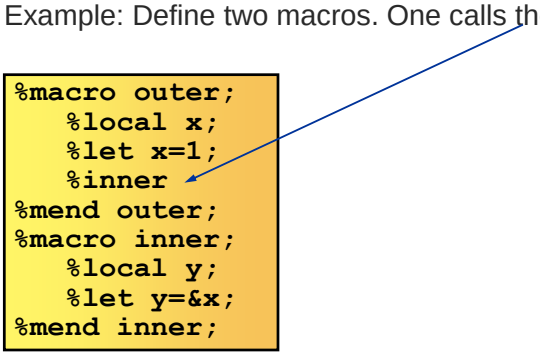
```
%macro a(value=);
   %b
%mend a;

%macro b;
   %put The value to write is: &value.;
   %put _user_;
%mend b;

%a(value=Today is Monday)
```

83

Multiple Local Tables

Multiple local tables can exist concurrently during macro execution.

Example: Define two macros. One calls the other.

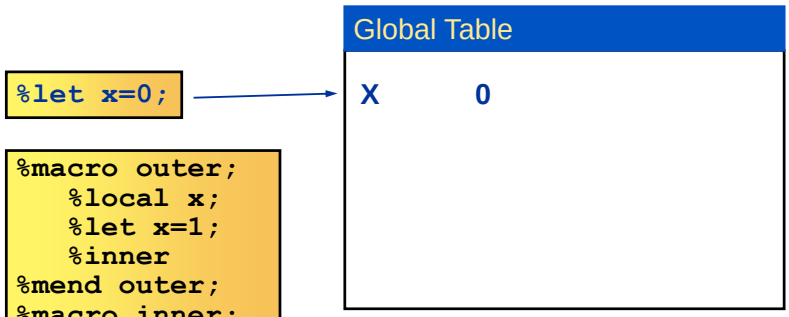


```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```

85

Multiple Local Tables

Create global macro variable **X**.

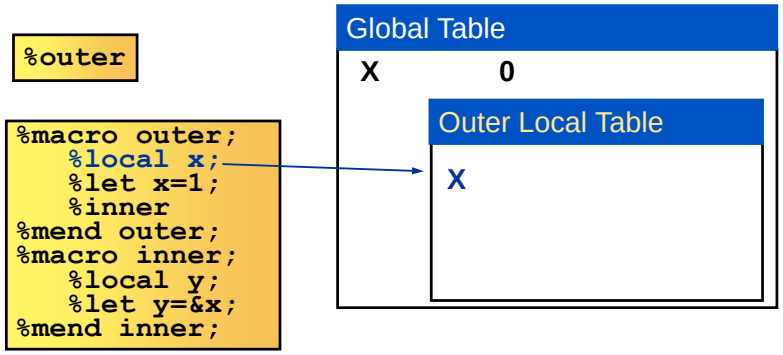



```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```

86

Multiple Local Tables

Call the **Outer** macro. When the %LOCAL statement executes, a local table is created.

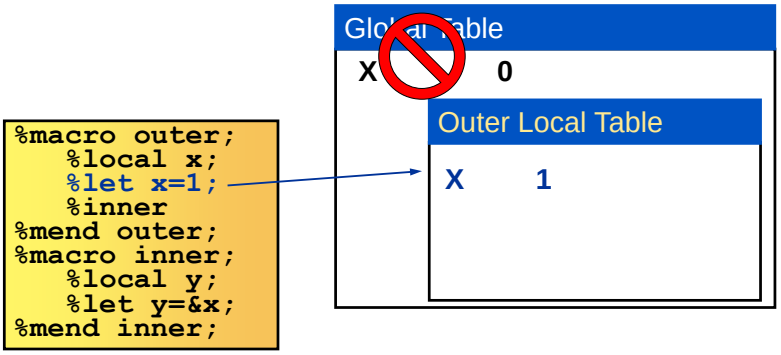



```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```

87

Multiple Local Tables

The %LET statement updates local macro variable **X**.
Access to global macro variable **X** is blocked.

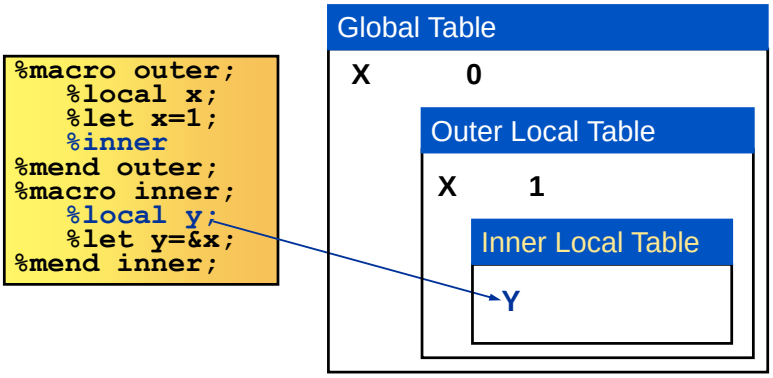


What happens if the %LOCAL statement in the **Outer** macro is omitted?

88

Multiple Local Tables

A nested macro call can create its own local symbol table in addition to any other tables that might currently exist.



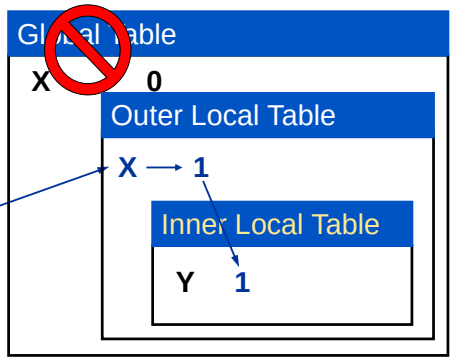
89

Multiple Local Tables

The macro processor resolves a macro variable reference by searching symbol tables in the reverse order in which they were created, as follows:

1. current local table
2. previously created local tables
3. global table

```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```



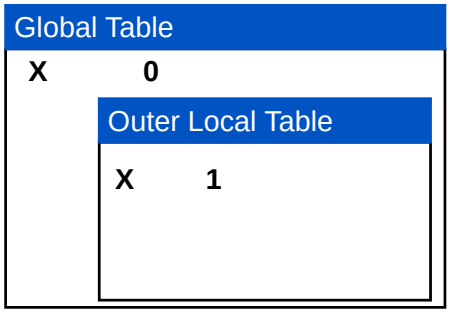
90

The global variable **X** is ***not*** available to the **Inner** macro.

Multiple Local Tables

When the **Inner** macro finishes execution, its local table is deleted. Control returns to the **Outer** macro.

```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```



91

Multiple Local Tables

When the **Outer** macro finishes execution, its local table is deleted. Only the global table remains.

```
%macro outer;
   %local x;
   %let x=1;
   %inner
%mend outer;
%macro inner;
   %local y;
   %let y=&x;
%mend inner;
```

Global Table

X	0

92

Setup for the Poll

The **Numobs** macro creates a macro variable with the number of observations in a SAS data set.

```
%macro numobs(dsn);
   options nonotes;
   data _null_;
      call symputx('num', number);
      stop;
      set &dsn nobs=number;
   run;
   options notes;
%mend numobs;
```

93

m105d14

5.07 Short Answer Poll

What is wrong with this attempt?

```
%macro numobs(dsn);
   options nonotes;
   data _null_;
      call symputx('num', number);
      stop;
      set &dsn nobs=number;
   run;
   options notes;
%mend numobs;

%numobs(orion.order_fact)
%put --->  &num observations;
```

94 m105d14

Multiple Local Tables

The SYMPUTX routine accepts an optional *scope* argument, which specifies where to store the macro variable.

```
%macro numobs(dsn);
   options nonotes;
   data _null_;
      call symputx('num', number,'G');
      stop;
      set &dsn
   run;
   options notes;
%mend numobs;
```

CALL SYMPUTX(*macro-variable*, *text* <,*scope*>**);**

G	The global symbol table.
L	The current macro's local symbol table. If no local symbol table exists for the current macro, a local symbol table is created.

SAS Log

```
1831  %numobs(orion.orders)
1832  %put --->  &num observations;
--->  490 observations
```

96 m105d14

The *scope* argument is recommended any time that the SYMPUTX routine is used within a macro definition.

The STOP statement stops DATA step execution. The special variable **NUMBER** is populated during compile time.

Multiple Local Tables

Example: The **Printds** macro calls the Numobs macro.

```
%macro printds(dsn);
   %let dsn=%upcase(&dsn);
   %numobs(&dsn)
   %if &num > 10 %then %do;
      title "First 10 of &num observations from &dsn";
   %end;
   %else %do;
      title "All &num observations from &dsn";
   %end;
   proc print data=&dsn(obs=10);
   run;
%mend printds;

%printds(orion.orders)
%printds(orion.country)
```

97 m105d14

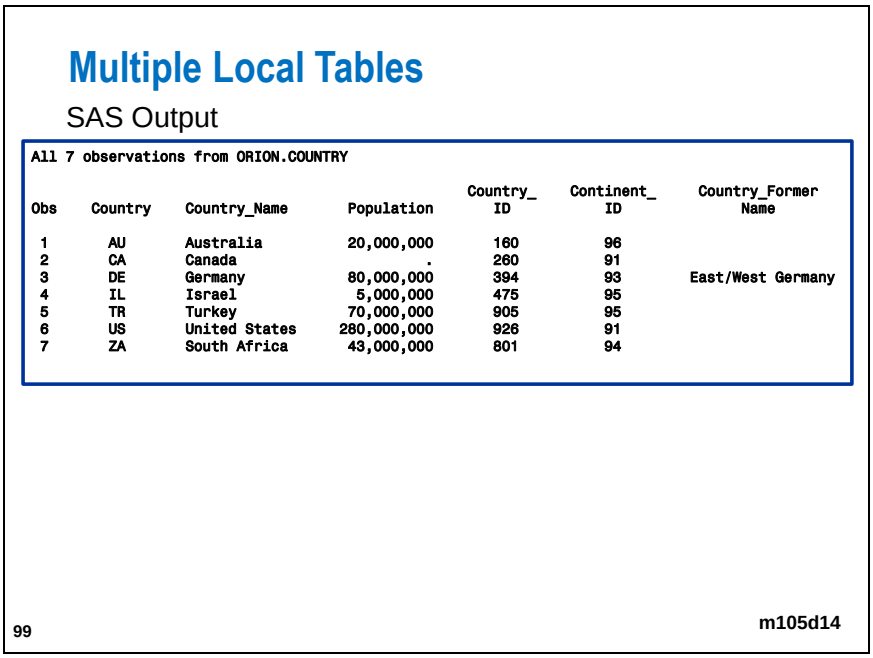
Multiple Local Tables

SAS Output

First 10 of 490 observations from ORION.ORDERS

Obs	Order_ID	Order_ Type	Employee_ID	Customer_ID	Order_ Date	Delivery_ Date
1	1230058123	1	121039	63	11JAN2007	11JAN2007
2	1230080101	2	99999999	5	15JAN2007	19JAN2007
3	1230106883	2	99999999	45	20JAN2007	22JAN2007
4	1230147441	1	120174	41	28JAN2007	28JAN2007
5	1230315085	1	120134	183	27FEB2007	27FEB2007
6	1230333319	2	99999999	79	02MAR2007	03MAR2007
7	1230338566	2	99999999	23	03MAR2007	08MAR2007
8	1230371142	2	99999999	45	09MAR2007	11MAR2007
9	1230404278	1	121059	56	15MAR2007	15MAR2007
10	1230440481	1	120149	183	22MAR2007	22MAR2007

98 m105d14



Exercises

Level 1

10. Understanding Symbol Tables

Without submitting the programs, identify in which symbol table the macro variable **dog** is located.

Assume that each example is submitted in a new SAS session.

a. _____

```
%let dog=Paisley;
%macro whereisit;
   %put My dog is &dog;
%mend whereisit;

%whereisit
```

b. _____

```
%macro whereisit;
   %let dog=Paisley;
   %put My dog is &dog;
%mend whereisit;

%whereisit
```

c. _____

```
%macro whereisit(dog);
   %put My dog is &dog;
%mend whereisit;

%whereisit(Paisley)
```

Level 2

11. Controlling Macro Variable Storage

a. Open the **m105e11** program shown below into the Editor window.

```
%macro varscope;
   data _null_;
      set orion.customer_type end=final;
      call symputx('localtype'||left(_n_), Customer_Type);
      if final then call symputx('localn',_n_);
   run;
   %put _user_;
%mend varscope;

%varscope
```

b. Modify the program so that all macro variables that are created in the DATA step are stored in the ***local*** symbol table.

c. Modify the program by adding the following statement before the DATA step and removing the scope specification in the SYMPUTX routine:

```
%local x;
```

In which symbol table are the macro variables stored?

_____

d. Modify the program so that all macro variables that are created in the DATA step are stored in the ***global*** symbol table.

Challenge

12. Creating Multiple Symbol Tables

a. Open the **cleanup** program and submit the macro to delete all global macro variables.

b. Open the **m105e12** program shown below into the Editor window.

```
%macro createmacvar;
   data _null_;
      set orion.lookup_order_type end=last;
      call symputx('type'||left(start), label,'L');
      if last then call symputx('endloop', _n_,'L');
   run;
%mend createmacvar;
```

```
%macro sumreport;
   %createmacvar
   %do num=1 %to &endloop;
      proc means data=orion.order_fact sum mean maxdec=2;
          where Order_Type = &num;
          var Total_Retail_Price CostPrice_Per_Unit;
          title "Summary Report for &&type&num";
      run;
   %end;
%mend sumreport;

%sumreport
```

Submit the program. Review and describe the results.

_____

_____

_____

c. Correct the program so that the **Sumreport** macro executes correctly and does not create any global macro variables. Verify that the title resolves properly. In addition, add an **s** to the end of the type description in the title.

PROC MEANS Output

Summary Report for Retail Sales

The MEANS Procedure

Variable	Label	Sum	Mean
Total_Retail_Price	Total Retail Price for This Product	44654.56	137.82
CostPrice_Per_Unit	Cost Price Per Unit	11730.73	36.21

Summary Report for Catalog Sales

The MEANS Procedure

Variable	Label	Sum	Mean
Total_Retail_Price	Total Retail Price for This Product	33931.35	199.60
CostPrice_Per_Unit	Cost Price Per Unit	8718.45	51.29

Summary Report for Internet Sales

The MEANS Procedure

Variable	Label	Sum	Mean
Total_Retail_Price	Total Retail Price for This Product	21491.55	174.73
CostPrice_Per_Unit	Cost Price Per Unit	5356.95	43.55

5.5 Solutions

Solutions to Exercises

1. Conditionally Processing Complete Statements

a. Open the program into the Editor window.

b. Modify the macro to test the CUSTTYPE parameter. If the value is null, insert VAR and TITLE statements into the PROC PRINT step. If the value is not null, insert WHERE, VAR, and TITLE statements into the PROC PRINT step.

```
%macro listing(custtype);
    proc print data=orion.customer noobs;
    %if &custtype= %then %do;
       var Customer_ID Customer_Name Customer_Type_ID;
       title "All Customers";
    %end;
    %else %do;
       where Customer_Type_ID=&custtype;
       var Customer_ID Customer_Name;
       title "Customer Type: &custtype";
    %end;
    run;
%mend listing;
```

c. Resubmit the macro definition using a null value and a valid value for CUSTTYPE.

Partial SAS Log

```
36   %listing()
MPRINT(LISTING):   proc print data=orion.customer noobs;
MPRINT(LISTING):   var Customer_ID Customer_Name Customer_Type_ID;
MPRINT(LISTING):   title "All Customers";
MPRINT(LISTING):   run;

NOTE: There were 77 observations read from the data set ORION.CUSTOMER.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.23 seconds
      cpu time            0.01 seconds

37   %listing(2010)
MPRINT(LISTING):   proc print data= orion.customer noobs;
MPRINT(LISTING):   where Customer_Type_ID=2010;
MPRINT(LISTING):   var Customer_ID Customer_Name;
MPRINT(LISTING):   title "Customer Type: 2010";
MPRINT(LISTING):   run;

NOTE: There were 5 observations read from the data set ORION.CUSTOMER.
      WHERE Customer_Type_ID=2010;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           1.13 seconds
      cpu time            0.01 seconds
```

2. Debugging a Macro

a. Open the program into the Editor window.

b. Change SATURDAY to today's value and submit the program.

c. Change today's value to the day that you launched SAS. Enter the day in mixed case, such as **Tuesday**.

3. Debugging a Macro

a. Open the program into the Editor window.

b. The %STR function is required to interpret OR as plain text instead of a logical operator.

```
%macro where(state);
   %if &state=%str(OR)
      %then %put Oregon;
      %else %put Wherever;
%mend where;
```

4. Validating a Macro Parameter

a. Open the program into the Editor window and submit it.

b. Use %IF to validate the TYPE parameter. The MINOPERATOR option in the %MACRO statement activates the IN operator.

```
%macro custtype(type)/minoperator;
   %let type=%upcase(&type);
   %if &type in GOLD INTERNET %then %do;
      proc print data=orion.customer_dim;
         var Customer_Group Customer_Name Customer_Gender
             Customer_Age;
         where upcase(Customer_Group) contains "&type";
         title "&type Customers";
      run;
   %end;
   %else %do;
      %put ERROR: Invalid TYPE: &type..;
      %put ERROR: Valid TYPE values are INTERNET or GOLD.;
   %end;
%mend custtype;
```

c. Resubmit the macro definition and call the macro using valid and invalid parameter values.

Partial SAS Log

```
248  %custtype(internet)

NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE UPCASE(Customer_Group) contains 'INTERNET';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

249  %custtype(silver)
ERROR: Invalid TYPE: SILVER.
```

d. Modify the macro to test whether TYPE is null. If so, do not execute PROC PRINT. Instead, the macro should write messages to the SAS log.

```
%macro custtype(type)/minoperator;
   %if &type= %then %do;
      %put ERROR: Missing TYPE.;
      %put ERROR: Valid TYPE values are INTERNET or GOLD.;
   %end;
   %else %do;
      %let type=%upcase(&type);
      %if &type in GOLD INTERNET %then %do;
         proc print data=orion.customer_dim;
            var Customer_Group Customer_Name Customer_Gender
                Customer_Age;
            where upcase(Customer_Group) contains "&type";
            title "&type Customers";
         run;
      %end;
      %else %do;
         %put ERROR: Invalid TYPE: &type..;
         %put ERROR: Valid TYPE values are INTERNET or GOLD.;
      %end;
   %end;
%mend custtype;
```

e. Resubmit the macro definition with a null parameter value, a valid value, and an invalid value.

Partial SAS Log

```
272  %custtype()
ERROR: Missing TYPE.
ERROR: Valid TYPE values are INTERNET or GOLD.
273  %custtype(internet)

NOTE: There were 8 observations read from the data set ORION.CUSTOMER_DIM.
      WHERE UPCASE(Customer_Group) contains 'INTERNET';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

274  %custtype(silver)
ERROR: Invalid TYPE: SILVER.
ERROR: Valid TYPE values are INTERNET or GOLD.
```

5. Validating a Macro Parameter Using Data-Driven Techniques

a. Open the program into the Editor window and submit it.

b. Modify the macro definition to validate CUSTTYPE against a data-driven list.

1) Use the SQL procedure to create the macro variable IDLIST that contains the unique values of the variable Customer_Type_ID in the orion.customer_type data set.

2) Validate the CUSTTYPE parameter against IDLIST.

3) If CUSTTYPE is in IDLIST, execute PROC PRINT. Otherwise, do not execute PROC PRINT. Instead, the macro should write this message to the SAS log:

Partial SAS Log

```
ERROR: Invalid CUSTTYPE.
       Valid CUSTTYPEs: 1010 1020 1030 1040 2010 2020 2030 3010.
```

```
%macro listing(custtype) / minoperator;
   %if &custtype=  %then %do;
      proc print data= orion.customer noobs;
         var Customer_ID Customer_Name Customer_Type_ID;
         title "All Customers";
      run;
   %end;
   %else %do;
      proc sql noprint;
         select distinct Customer_Type_ID
            into :IDlist separated by ' '
            from orion.customer_type;
      quit;
      %if &custtype in &idlist %then %do;
         proc print data= orion.customer noobs;
            where Customer_Type_ID =&custtype;
            var Customer_ID Customer_Name;
           title "Customer Type: &custtype";
         run;
      %end;
      %else %do;
         %put ERROR: Invalid CUSTTYPE.;
         %put ERROR- Valid CUSTTYPEs: &IDlist..;
      %end;
   %end;
%mend listing;
```

c. Resubmit the macro definition and call the macro with a null parameter value, a valid value, and an invalid value.

Partial SAS Log

```
411  %listing()
MPRINT(LISTING):   proc print data= orion.customer noobs;
MPRINT(LISTING):   var Customer_ID Customer_Name Customer_Type_ID;
MPRINT(LISTING):   title "All Customers";
MPRINT(LISTING):   run;

NOTE: There were 77 observations read from the data set ORION.CUSTOMER.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

412  %listing(1020)
MPRINT(LISTING):   proc sql noprint;
MPRINT(LISTING):   select distinct Customer_Type_ID into :IDlist separated by ' ' from
orion.customer_type;
```

```
MPRINT(LISTING):   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

MPRINT(LISTING):   proc print data= orion.customer noobs;
MPRINT(LISTING):   where Customer_Type_ID =1020;
MPRINT(LISTING):   var Customer_ID Customer_Name;
MPRINT(LISTING):   title "Customer Type: 1020";
MPRINT(LISTING):   run;
NOTE: There were 17 observations read from the data set ORION.CUSTOMER.
      WHERE Customer_Type_ID=1020;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

413  %listing(9999)
MPRINT(LISTING):   proc sql noprint;
MPRINT(LISTING):   select distinct Customer_Type_ID into :IDlist separated by ' ' from
orion.customer_type;
MPRINT(LISTING):   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

ERROR: Invalid CUSTTYPE.
       Valid CUSTTYPEs: 1010 1020 1030 1040 2010 2020 2030 3010.
```

6. Validating a Macro Parameter

a. Open the program into the Editor window and submit it.

b. Modify the macro to validate parameters according to the following requirements:

Macro Variable	Possible Values
decimals	*0* to *4*
order	*INTERNAL*, *FREQ*

c. Create an additional macro variable named **numerrors** that accumulates the number of parameter errors.

1) Execute PROC MEANS only if **numerrors** is zero.

2) The macro should write messages to the SAS log when parameters are invalid.

```
%macro salarystats(decimals=2,order=internal)/minoperator;
   %let numerrors=0;
   %if not (&decimals in 0 1 2 3 4) %then %do;
      %let numerrors=%eval(&numerrors+1);
      %put ERROR: Invalid DECIMALS parameter: &decimals..;
      %put ERROR- Valid DECIMALS values are 0 to 4.;
   %end;
   %let order=%upcase(&order);
   %if not (&order in INTERNAL FREQ) %then %do;
      %let numerrors=%eval(&numerrors+1);
      %put ERROR: Invalid ORDER parameter: &order..;
```

```
      %put ERROR- Valid ORDER values are INTERNAL or FREQ.;
   %end;
   %if &numerrors=0 %then %do;
      options nolabel;
      title 'Salary Stats';
      proc means data=orion.staff maxdec=&decimals order=&order;
          where job_title contains 'Sales';
          var salary;
          class job_title;
       run;
       title;
   %end;
   %else %put ERROR: &numerrors errors. Code not submitted.;
%mend salarystats;

%salarystats()
%salarystats(decimals=5,order=fudge)
```

7. Using Macro Loops and Indirect References

a. Open the program into the Editor window.

b. Define a macro that generates a separate PROC MEANS step for each of the order types in the orion.order_fact data set. The values of Order_Type are *1*, *2*, and *3*.

```
%macro sumreport;
   %do num=1 %to 3;
      proc means data=orion.order_fact sum mean maxdec=2;
          where Order_Type=&num;
          var Total_Retail_Price CostPrice_Per_Unit;
          title "Order Type: &num";
       run;
   %end;
%mend sumreport;

%sumreport
```

c. Insert the provided DATA step to create the series of macro variables.

d. Modify the macro to use the following:

- the **endloop** macro variable as the stop value for the iterative DO loop
- an indirect reference to **type*x*** in the TITLE statement

```
%macro sumreport;
   data _null_;
      set orion.lookup_order_type end=last;
      call symputx('type'||left(_n_), label);
      if last then call symputx('endloop', _n_);
   run;
   %do num=1 %to &endloop;
      proc means data=orion.order_fact sum mean maxdec=2;
          where Order_Type=&num;
          var Total_Retail_Price CostPrice_Per_Unit;
```

```
          title "Order Type: &&type&num";
      run;
   %end;
%mend sumreport;

%sumreport
```

8. Generating Data-Dependent Steps

a. Open the program into the Editor window.

b. Modify the macro to print a listing of the **top*x*** customers from the **orion.customer_dim** data set. Display the variables **Customer_ID**, **Customer_Name**, and **Customer_Type**.
Use a macro loop to dynamically generate values for the WHERE statement based on the macro variables **top1** through top*x*.

```
%macro tops(obs=3);
   proc means data=orion.order_fact sum nway noprint;
      var Total_Retail_Price;
      class Customer_ID;
      output out=customer_freq sum=sum;
   run;

   proc sort data=customer_freq;
      by descending sum;
   run;

   data _null_;
      set customer_freq(obs=&obs);
      call symputx('top'||left(_n_), Customer_ID);
   run;

   proc print data=orion.customer_dim noobs;
      where Customer_ID in (%do num=1 %to &obs; &&top&num %end;);
      var Customer_ID Customer_Name Customer_Type;
      title "Top &obs Customers";
   run;
%mend tops;

%tops()
%tops(obs=5)
```

9. Generating Multiple Macro Calls

a. Review the output generated by the **Listall** macro in program **m105e09**.

```
GLOBAL N 8
GLOBAL TYPE1 Orion Club members inactive
GLOBAL TYPE2 Orion Club members low activity
GLOBAL TYPE3 Orion  Club members medium activity
GLOBAL TYPE4 Orion  Club members high activity
GLOBAL TYPE5 Orion Club Gold members low activity
GLOBAL TYPE6 Orion Club Gold members medium activity
GLOBAL TYPE7 Orion Club Gold members high activity
GLOBAL TYPE8 Internet/Catalog Customers
```

b. Modify the **Listall** macro to call the **Memberlist** macro. The result of the macro call should create a PROC PRINT step for each customer type. Use a macro loop and indirect references to generate the appropriate macro calls.

```
%macro memberlist(custtype);
   proc print data=Orion.Customer_dim;
      var Customer_Name Customer_ID Customer_Age_Group;
      where Customer_Type="&custtype";
      title "A List of &custtype";
   run;
%mend memberlist;

%macro listall;
   data _null_;
      set orion.customer_type end=final;
      call symputx('type'||left(_n_), Customer_Type);
      if final then call symputx('n',_n_);
   run;
   %do num=1 %to &n;
      %memberlist(&&type&num)
   %end;
%mend listall;

%listall
```

10. Understanding Symbol Tables

Without submitting the programs, identify in which symbol table the macro variable **dog** is located.

Assume that each example is submitted in a new SAS session.

a. Because the %LET statement is outside the macro definition, the macro variable **dog** is stored in the global symbol table.

```
%let dog=Paisley;
%macro whereisit;
   %put My dog is &dog;
%mend whereisit;
%whereisit
```

b. Because the %LET statement is inside the macro definition, the macro variable **dog** is stored in the local symbol table.

```
%macro whereisit;
   %let dog=Paisley;
   %put My dog is &dog;
%mend whereisit;
%whereisit
```

c. Because DOG is a macro parameter, it is stored in the local symbol table.

```
%macro whereisit(dog);
   %put My dog is &dog;
%mend whereisit;
%whereisit(Paisley)
```

11. Controlling Macro Variable Storage

a. Open the program into the Editor window.

b. Specifying **L** as the third argument of the SYMPUTX routine stores the macro variable in the local symbol table.

```
%macro varscope;
   data _null_;
      set orion.customer_type end=final;
      call symputx('localtype'||left(_n_), Customer_Type,'L');
      if final then call symputx('localn',_n_,'L');
   run;
   %put _user_;
%mend varscope;
%varscope
```

c. By adding the %LOCAL statement and removing the scope specification, the SYMPUTX routine creates local macro variables.

```
2    %macro varscope;
3      %local x;
4       data _null_;
5          set orion.customer_type end=final;
6          call symputx('localtype'||left(_n_), Customer_Type);
7          if final then call symputx('localn',_n_);
8       run;
9       %put _user_;
10   %mend varscope;
11
12   %varscope

NOTE: Numeric values have been converted to character values at the places given by:
      (Line):(Column).
      1:94
NOTE: There were 8 observations read from the data set ORION.CUSTOMER_TYPE.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


VARSCOPE LOCALN 8
VARSCOPE LOCALTYPE1 Orion Club members inactive
VARSCOPE LOCALTYPE2 Orion Club members low activity
```

```
VARSCOPE LOCALTYPE3 Orion  Club members medium activity
VARSCOPE LOCALTYPE4 Orion  Club members high activity
VARSCOPE LOCALTYPE5 Orion Club Gold members low activity
VARSCOPE LOCALTYPE6 Orion Club Gold members medium activity
VARSCOPE LOCALTYPE7 Orion Club Gold members high activity
VARSCOPE LOCALTYPE8 Internet/Catalog Customers
VARSCOPE X
```

d. Specifying **G** as the third argument of the SYMPUTX routine stores the macro variables in the global symbol table.

```
%macro varscope;
   %local x;
   data _null_;
      set orion.customer_type end=final;
      call symputx('localtype'||left(_n_), Customer_Type,'G');
      if final then call symputx('localn',_n_,'G');
   run;
   %put _user_;
%mend varscope;
%varscope
```

12. Creating Multiple Symbol Tables

a. Open the **cleanup** program and submit the macro.

b. Open the **m105e12** program into the Editor window.

c. Correct the program so that the **Sumreport** macro executes correctly and does not create any global macro variables. Verify that the title resolves properly. In addition, add an s to the end of the type description in the title.

The macro variables **endloop** and **type*x*** are stored in the local symbol table for the **Createmacvar** macro and are not available to the **Sumreport** macro. Delete the scope argument from the SYMPUTX routine and add the %LOCAL statement to the **Sumreport** macro to force the macro variables into the local symbol table for the **Sumreport** macro.

```
%macro createmacvar;
   data _null_;
      set orion.lookup_order_type end=last;
      call symputx('type'||left(start), label);
      if last then call symputx('endloop', _n_);
   run;
%mend createmacvar;

%macro sumreport;
   %local type1 type2 type3 endloop num;
   %createmacvar
   %do num=1 %to &endloop;
      proc means data=orion.order_fact sum mean maxdec=2;
          where Order_Type = &num;
          var Total_Retail_Price CostPrice_Per_Unit;
        title "Summary Report for &&type&num..s";
      run;
   %end;
```

```
%mend sumreport;

%sumreport
```

Solutions to Student Activities (Polls/Quizzes)

5.01 Short Answer Poll – Correct Answer

Submit the program **m105a01**. What error do you see in the log?

Partial SAS Log

```
514  %macro reports;
515     %daily
516     %if &sysday=Friday then %weekly;
ERROR: Expected %THEN statement not found.  A dummy macro will be
       compiled.
517  %mend reports;
```

If a macro definition contains macro language syntax errors, error messages are written to the SAS log and a dummy (nonexecutable) macro is created.

13

Idea Exchange

What is the difference between %IF-%THEN and IF-THEN?

	%IF-%THEN	IF-THEN
Valid in	Macro definition	DATA step
Performs	SAS code (text) processing	Data processing
Passed to	Macro processor	DATA step compiler
Purpose	Determine what SAS code to place on the input stack for tokenization, compilation, and eventual execution	Determine what DATA step statement (or statements) to execute during each execution-time iteration of the DATA step

24

5.02 Short Answer Poll – Correct Answer

Instead of using multiple OR operators, which SAS comparison operator compares a value to a list of values?

```
if Country in ('FR' 'CA' 'DE' 'ZA') then do;
```

The IN operator is a SAS comparison operator.

36

5.03 Short Answer Poll – Correct Answer

What can be done if the value list is extremely long or changes frequently, or both?

```
%macro customers(place) / minoperator;
   %let place=%upcase(&place);
   %if &place in AU CA DE IL TR US ZA %then %do;
      proc print data=orion.customer;
         var customer_name customer_address country;
         where upcase(country)="&place";
         title "Customers from &place";
      run;
   %end;
   %else %put Sorry, no customers from &place..;
%mend customers;

%customers(de)
%customers(aa)
```

Create the list dynamically using a data-driven approach.

m105d05b

39

5.04 Multiple Choice Poll – Correct Answer

Which statement correctly creates an index variable named **i**?

a. %do &i=1 %to 10;
b. %do &i=1 to 10;
c. %do i=1 %to 10;
d. %do i=1 to 10;

51

5.05 Short Answer Poll – Correct Answer

Given the symbol table below, what is the value of &&VAL&COUNT?

```
SPLIT COUNT 7
SPLIT DATA orion.customer
SPLIT VAL1 AU
SPLIT VAL2 CA
SPLIT VAL3 DE
SPLIT VAL4 IL
SPLIT VAL5 TR
SPLIT VAL6 US
SPLIT VAL7 ZA
SPLIT VAR country
```

&&VAL&COUNT ⇨ &VAL7 ⇨ ZA

61

5.06 Short Answer Poll – Correct Answer

How many local symbol tables are created when macro **A** is called and begins to execute?

```
%macro a(value=);
   %b
%mend a;

%macro b;
   %put The value to write is: &value.;
   %put _user_;
%mend b;

%a(value=Today is Monday)
```

One. The parameter list in macro A causes a local symbol table to be created. No local symbol table is created when macro B is called because macro B does not create local variables.

84

5.07 Short Answer Poll – Correct Answer

What is wrong with this attempt?

```
352  %macro numobs(dsn);
353     options nonotes;
354     data _null_;
355         call symputx('num', number);
356         stop;
357         set &dsn nobs=number;
358     run;
359     options notes;
360  %mend numobs;
NOTE: The macro NUMOBS completed compilation without errors.
      9 instructions 324 bytes.
361
362  %numobs(orion.order_fact)
363  %put --->  &num observations;
WARNING: Apparent symbolic reference NUM not resolved.
--->  &num observations
```

The macro variable num was placed in the Numobs macro's local table and deleted when the Numobs macro finished execution.

95 m105d14

Chapter 6 Learning More

6.1 SAS Resources

Objectives

- Identify areas of support that SAS offers.

3

Education

Comprehensive training to deliver greater value to your organization

- More than 200 course offerings
- World-class instructors
- Multiple delivery methods: instructor-led and self-paced
- Training centers around the world

http://support.sas.com/training/

4

SAS Books

Convenient. Practical. Enlightening.
Valuable insight with solid results.

Available in a variety of formats to best meet your needs:

- hard-copy books
- e-books
- PDF

5 **www.sas.com/store/books**

SAS Global Certification Program

SAS offers several globally recognized certifications.

- Computer-based certification exams – typically 60-70 questions and 2-3 hours in length
- Preparation materials and practice exams available
- Worldwide directory of SAS Certified Professionals

http://support.sas.com/certify/

6

Support

SAS provides a variety of self-help and assisted-help resources.

- SAS Knowledge Base
- Downloads and hot fixes
- License assistance
- SAS discussion forums
- SAS Technical Support

http://support.sas.com/techsup/

7

User Groups

SAS supports many local, regional, international, and special-interest SAS user groups.

- SAS Global Forum
- Online SAS Community: http://www.sascommunity.org/wiki/

http://support.sas.com/usergroups/

8

6.2 Beyond This Course

Objectives

- Identify the next course that follows this course.

10

Next Steps

To learn advanced macro techniques, enroll in the following course:

SAS® Macro Language 2: Advanced Techniques

11

Next Steps

In addition, there are prerecorded short technical discussions and demonstrations called e-lectures.

http://support.sas.com/training/

12