Paper 53

SAS® Macros Variables and Simple Macro Programs

Steven First, Systems Seminar Consultants, Madison, WI

Abstract

The SAS programming language has a rich "tool-box" of features that can offer a lot of power to the user. The ability to use macro variables and macro code is difficult to learn without some training and examples that are easy to understand. This tutorial will introduce the user to a few of the very powerful techniques possible using macro variables and macro programming.

Objectives:

After this tutorial attendees will:

- understand how the macro system fits with the rest of SAS software
- use system (automatic) macro variables
- create and use user defined macro variables
- define simple macros
- pass data from the data step to the macro system.

SAS Macro Overview

Macros construct input for the SAS compiler.

Functions of the SAS macro processor:

- pass symbolic values between SAS statements and steps
- establish default symbolic values
- conditionally execute SAS steps
- invoke very long, complex code in a quick, short way.

Notes:

The MACRO PROCESSOR is the SAS system module that processes macros.
The MACRO LANGUAGE is how you communicate with the processor.

Traditional SAS Programming

Without macros what are some things you *cannot* easily do?

- substitute text in statements like TITLEs
- communicate across SAS steps
- establish default values
- conditionally execute SAS steps
- hide complex code that can be invoked easily.

Traditional SAS Programming

Without macros, SAS programs are a just series of DATA and PROC steps.

1. The program is scanned one statement at a time looking for the beginning of step (step boundary).
2. When the beginning of step is found, all statements in the step are **compiled**.
3. When the end of step is found (the next step boundary), the previous step **executes.**

Step boundaries are the SAS keywords:

DATA	ENDSAS
PROC	LINES
CARDS	LINES4
CARDS4	PARMCARDS
DATALINES	QUIT
DATALINES4	RUN

Notes:

Each step is compiled and executed independently.
All SAS jobs have a step boundary at the beginning.
Batch and non-interactive jobs have an implied ENDSAS at end of the SYSIN file.
RUN and QUIT have special considerations.

Step boundaries control the compilation and execution of SAS.

```
data saleexps;              <- Step, start compile
 infile rawin;
 input name $1-10 division $12
       years 15-16 sales 19-25
       expense 27-34;     <  Step, exec , start compile
proc print data=saleexps;
                           <  Step, exec, start compile
proc means data=saleexps;
 var sales expense;
                           <  EOF (batch), exec, ENDSAS.
```

Notes:

Interactive jobs have no end-of-file, so the

last step will compile but not execute.

The RUN Statement

RUN acts as an explicit step boundary in most PROCs.

```
data saleexps;                < start compile
 infile rawin;
 input name $1-10
       division $12
       years 15-16 sales 19-25
        expense 27-34;
run;                          < exec previous

proc print data=saleexps; <- start compile
 run;                     <  end, exec previous

proc means data=saleexps;
 var sales expense;
 run;                     <  end, exec
```

Notes:

The use of RUN after each step is highly recommended.

SAS Global Statements

Global statements are executed immediately.

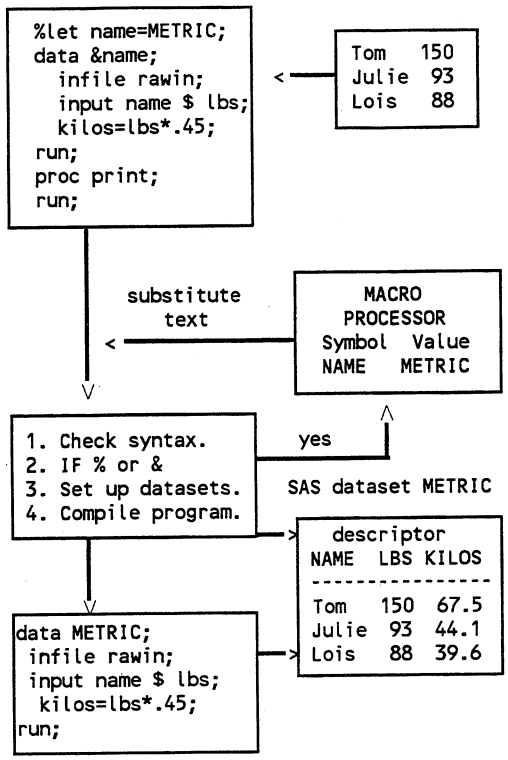
```
filename rawin 'saleexps.dat'; < comp, execute
data saleexps;                 <  Start compile
 infile rawin;
 input name $1-10 division $12
       years 15-16 sales 19-25
       expense 27-34;
 run;                          < end, exec
proc print data=saleexps;      < start compile
options ls=80 nodate;          <  comp, exec
title 'sample title';          < comp, exec
 run;                          <- end, exec
options ls=132 nodate;         <- comp, execute
proc means data=saleexps;      < start compile
 var sales expense;
 run;                          < end, exec
```

Notes:

RUN ensures global statements affect the correct step.

Macro Processor Flow

Macro statements are given to the macro processor BEFORE the compiler.



The SAS Macro Language

A second SAS programming language for string manipulation.

Characteristics:

- strings are sequences of characters
- all input to the macro language is a string
- usually strings are SAS code, but don't need to be
- the macro processor manipulates strings and may send them back for scanning.

Macro Language Components

The macro language has several kinds of components.

Macro variables:

- are used to store and manipulate character strings
- follow SAS naming rules
- are NOT the same as DATA step variables
- are stored in memory in a macro symbol table.

Macro statements:

- begin with a % and a macro keyword and end with semicolon (;)
- assign values, substitute values, and change macro variables
- can branch or generate SAS statements conditionally.

Macro functions:

- help process and evaluate text and macro variables
- have some capabilities of SAS data step functions
- have some unique capabilities.

Macro Language Components (continued)

Macro expressions:

- are sequences of text linked with operators and parentheses
- are needed by some macro functions and programming statements.

Macro constants:

- are treated as character strings
- can contain literals, variable names, numbers, dataset names, SAS statements.

A Macro Problem

You would like to have the Day of Week and current date appear in a title, but SAS titles are text, not variables.

Solution: Use some system macro variables.

```
PROC PRINT DATA=DEPTSALE;
 TITLE  "Department Sales as of &SYSDAY  &SYSDATE";
 TITLE2 "Deliver to Michael O'Malley";
RUN;
```

Department Sales as of Wednesday 04JAN89
Deliver to Michael O'Malley

OBS	DEPT	YEAR	SALARIES	SALES
1	1	87	1000	1000
2	1	88	1000	2000
3	2	87	500	3000
4	2	88	600	2000

Notes:

Macro variables are NOT resolved within double quotes.
Macro variables can be used in almost any SAS statement.

Partial List of Automatic Macro Variables

SYSBUFFR	text entered in response to %INPUT
SYSCMD	last non-SAS command entered
SYSDATE	current date DATE6., DATE7. format
SYSDAY	current day of the week
SYSDEVIC	current graphics device
SYSDSN	last ds Ex. WORK SOFTSALE
SYSENV	SAS environment (FORE or BACK)
SYSERR	return code set by SAS procedures
SYSFILRC	FILENAME rc
SYSINDEX	number of macros started in job
SYSINFO	system information given by some PROCS
SYSJOBID	name of executing job or user
SYSLAST	last SAS dataset built Ex. WORK.SOFTSALE
SYSLIBRC	rc from last LIBNAME statement
SYSLCKRC	whether most recent lock was successful
SYSMENV	macro environment
SYSMSG	message displayed with %DISPLAY
SYSPARM	value passed from SYSPARM in JCL
SYSPROD	indicates whether a SAS product is licensed
SYSPBUFF	all macro parameters passed
SYSRC	return code from macro processor
SYSSCP	operating system where SAS is running
SYSTIME	starting time of job
SYSVER	SAS version

Example:

```
FOOTNOTE "REPORT WAS RUN ON &SYSDAY, &SYSDATE";
```

Resolves to:

```
FOOTNOTE "REPORT WAS RUN ON MONDAY, 17FEB97";
```

Displaying Macro Variables

%PUT displays macro variables to the log at compile time.

Syntax:

%PUT *text macrovariables _all_;*

Example:

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
```

```
%PUT ***** &SYSDATE *****;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Partial SAS Log:

```
***** 13FEB97 *****
```

Notes:

ALL_ support for %PUT was added in release 6.08.

Displaying All Macro Variables

%PUT can display all current macro variables.

%PUT _ALL_;

Partial SAS Log:

```
GLOBAL MBILLYR 97
GLOBAL SSCDEV C
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 18FEB97
AUTOMATIC SYSDAY Tuesday
AUTOMATIC SYSDSN          _NULL_
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 1
AUTOMATIC SYSINFO 0
AUTOMATIC SYSJOBID 0000016959
AUTOMATIC SYSLAST _NULL_
AUTOMATIC SYSMSG
AUTOMATIC SYSPARM
AUTOMATIC SYSRC 0
AUTOMATIC SYSSCP WIN
AUTOMATIC SYSSCPL WIN_32S
AUTOMATIC SYSSITE 0011485002
AUTOMATIC SYSTIME 10:35
AUTOMATIC SYSVER 6.11
AUTOMATIC SYSVLONG 6.11.0040P030596
```

Notes:

User macro variables will be covered later. SASHELP.VMACRO is a SAS dataset containing all known macro variables (use PROC PRINT to view).

Practice Exercises

Exercise 1:

A program that will build several datasets used later in this course is provided on diskette.
Start SAS, INCLUDE `A:SASM0001.SAS' and submit it.
Check the SAS log to insure that the program ran correctly, use the LIBNAME command or submit proc contents data=work._all_;run; to verify that the datasets were built correctly.

Exercise 2:

Determine the values of the following system macro variables using
your current computer system.

&SYSDAY	Current day of the week
&SYSTIME	Current time
&SYSSCP	Operating system being used
&SYSVER	Current SAS version number
&SYSDATE	Current date, in DATE7. Format

Exercise 3:

Using system macro variables run a PROC CONTENTS and a PROC PRINT on the LAST SAS dataset that was created. Include it's name in a title.

A Macro Problem

You reference a SAS datasetname several times in a SAS job.

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

You would like to:

- ☐ be able to change the name quickly in one place only
- ☐ have the datasetname appear in a title.

How can you do the above?

Solution: Use a macro variable.

Macro Variables

You can define macro variables with %LET.
You refer to the variables later with &*variable*.
Macro will substitute *value* for all occurrences of &*variable*.

Syntax:

%LET *variable*=*value*;

```
%LET NAME=PAYROLL;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
```

```
RUN;
```

Resolves to:

```
DATA PAYROLL;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
PROC PRINT DATA=PAYROLL;
 TITLE "PRINT OF DATASET PAYROLL";
RUN;
```

Notes:

Macro variables are not resolved within single quotes.
Leading and trail spaces are discarded.

Assigning a New Value

Use another %LET to assign a different value.

```
%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

Resolves to:

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Assigning SAS Statements to Macro Variables

%STR allows values with ; etc.

```
%LET NAME=NEWPAY;
%LET CHART=%STR(PROC CHART;VBAR EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

Resolves to:

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Nesting of Macro Variables

Macro variables can contain other macro variables.

```
%LET NAME=NEWPAY;
%LET  CHART=%STR(PROC  CHART  DATA=&NAME;VBAR
EMP;RUN;);
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
&CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

Resolves to:

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
PROC CHART DATA=NEWPAY;VBAR EMP;RUN;
PROC PRINT DATA=NEWPAY;
 TITLE "PRINT OF DATASET NEWPAY";
RUN;
```

Practice Exercises

Work out the problems below on paper.
If terminals are available, log on, type in the statements, and use %PUT statements to check you answers.

Exercise 4:

After execution of the following %LET statements

```
%LET A=ANDY;
%LET B=1989;
%LET C=CANES;
%LET D=DECEMBER 31,;
%LET E="TREMENDOUS";
```

What would be the results of these %PUT statements? (Include all spaces).

```
%PUT &C;
```

```
%PUT FISCAL YEAR &B;
```

```
%PUT YEAR ENDED &D &B;
```

```
%PUT &B C&A&C WERE SOLD IN &B;


%PUT &B WAS A &E SALES YEAR!;
```

Practice Exercises

Exercise 5:

After execution of the following %LET statements

```
%LET A=WIND;
%LET B=WATER;
%LET C=SURFING;
%LET &C=FUN;
%LET D=AND;
%LET E=MAN;
%LET &E=%STR( SNOW );
%LET F=%STR(PROC PRINT;);
%LET G=%STR(RUN;);
```

What would be the results of these %PUT statements? (Include all spaces).

```
%PUT &A&C;


%PUT &A &D &B &C = &SURFING;


%PUT H&DY &E;


%PUT H&D.Y &E;


%PUT H&D.Y.&E;


%PUT &&C &&&C;


%PUT &MAN&E;


%PUT &F &G;
```

What is a Macro?

Stored text that can be inserted anywhere in a SAS program and expanded.

Macros can include:

- ☐ constants such as literals, variables, names, statements
- ☐ assignments to macro variables
- ☐ macro programming statements
- ☐ macro language functions
- ☐ invocations of other functions
- ☐ nested macro definitions.

Defining and Using Macros

%MACRO and %MEND define macros.
%*macroname* will invoke it later.

Example: define a macro to run PROC CHART and later invoke

```
%MACRO CHART;
  PROC CHART DATA=&NAME;
    VBAR EMP;
RUN;
%MEND;

%LET NAME=NEWPAY;
DATA &NAME;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
%CHART
PROC PRINT DATA=&NAME;
 TITLE "PRINT OF DATASET &NAME";
RUN;
```

Resolves to:

```
DATA NEWPAY;
 INPUT EMP$ RATE;
 DATALINES;
TOM 10
JIM 10
;
RUN;
PROC CHART DATA=NEWPAY;
VBAR EMP;
 RUN;
 PROC PRINT DATA=NEWPAY;
  TITLE "PRINT OF DATASET NEWPAY";
 RUN;
```

Positional Macro Parameters

Macro parameters are defined in order after the macro name.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
  RUN;
%MEND;

%CHART(PAYROLL,EMP)
```

Resolves to:

```
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Notes:

Keyword parameters are also allowed.
Keyword parameters can give default values.

Nested Macros

Macros can call other macros.

```
%MACRO CHART(NAME,BARVAR);
  PROC CHART DATA=&NAME;
    VBAR &BARVAR;
  RUN;
%MEND;

%MACRO PTCHART(NAME,BARVAR);
%CHART(PAYROLL,EMP)
  PROC PRINT DATA=&NAME;
    TITLE "PRINT OF DATASET &NAME";
  RUN;
%MEND;

%PTCHART(PAYROLL,EMP)
```

Resolves to:

```
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
PROC PRINT DATA=PAYROLL;
  TITLE  "PRINT OF DATASET PAYROLL";
RUN;
```

Conditional Macro Compilation

%IF can conditionally pass code to the compiler.

Example: Run PROC PRINT only if PRTCH = YES.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN PROC PRINT DATA=&NAME;
;
   PROC CHART DATA=&NAME;
    VBAR &BARVAR;
   RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

Resolves to:

```
PROC PRINT DATA=PAYROLL ;
PROC CHART DATA=PAYROLL;
  VBAR EMP;
RUN;
```

Conditional Macro Compilation

%DO allows more than one statement to be conditionally compiled.

Example: Submit as before, but include titles.

```
%MACRO PTCHT(PRTCH,NAME,BARVAR);
  %IF &PRTCH=YES %THEN
      %DO;
          PROC PRINT DATA=&NAME;
          TITLE "PRINT OF DATASET &NAME";
                RUN;
      %END;
  PROC CHART DATA=&NAME;
   VBAR &BARVAR;
  RUN;
%MEND;

%PTCHT(YES,PAYROLL,EMP)
```

Resolves to:

```
PROC PRINT DATA=PAYROLL;
 TITLE  "PRINT OF DATASET PAYROLL";
RUN;
PROC CHART DATA=PAYROLL;
 VBAR EMP;
RUN;
```

Iterative Macro Invocation

%DO can also vary a value.

Example: Run PROC PRINT &PRTNUM times.

```
%MACRO PRTMAC(PRTNUM,NAME);
 %DO I= 1 %TO &PRTNUM;
     PROC PRINT DATA=&NAME&I;
      TITLE "PRINT OF DATASET &NAME&I";
        RUN;
 %END;
%MEND;

%PRTMAC(4,PAYROLL)
```

Resolves to:

```
PROC PRINT DATA=PAYROLL1;
  TITLE "PRINT OF DATASET PAYROLL1";
RUN;
PROC PRINT DATA=PAYROLL2;
  TITLE "PRINT OF DATASET PAYROLL2";
RUN;
PROC PRINT DATA=PAYROLL3;
  TITLE "PRINT OF DATASET PAYROLL3";
RUN;
PROC PRINT DATA=PAYROLL4;
  TITLE "PRINT OF DATASET PAYROLL4";
RUN;
```

Practice Exercises

Exercise 6:

If the following macro was defined to the SAS system

```
%MACRO FREQ;
  PROC FREQ DATA=&DSN;
    TABLES &VAR1*&VAR2 / NOPERCENT;
  RUN;
%MEND FREQ;
```

What code would the SAS compiler see after these statements?

```
%LET DSN=FREQ;
%LET VAR1=DEPT;
%LET VAR2=SALES;

%FREQ
```

Practice Exercises

Exercise 7:

If the following macro was defined to the SAS system

```
%MACRO FREQ(DSN,VAR1,VAR2);
  PROC FREQ DATA=&DSN;
    TABLES &VAR1*&VAR2 / NOPERCENT;
  RUN;
%MEND FREQ;
```

What code would the SAS compiler see after this macro call?

A) `%FREQ(FREQ,SALARIES,SALES)`

SAS would stop processing the code generated by the following calls. Can you determine why?

B) `%FREQ()`

C) `%FREQ(FREQ,SALARIES,)`

SAS DATA Step Interfaces

SYMGET, SYMPUT, and macro variables can transfer values between SAS steps.

Example: Display the number of observations in a dataset in a title.

```
%MACRO OBSCOUNT(NAME);
  DATA _NULL_;
   SET &NAME NOBS=OBSOUT;
   CALL SYMPUT('MOBSOUT',OBSOUT);
   STOP;
  RUN;
  PROC PRINT DATA=&NAME;
   TITLE "DATASET &NAME CONTAINS &MOBSOUT OBSERVATIONS";
  RUN;
%MEND;

%OBSCOUNT(PAYROLL)
```

Resolves to:

```
DATA _NULL_;
  SET PAYROLL NOBS=OBSOUT;
  CALL SYMPUT('MOBSOUT',OBSOUT);
  STOP;
  RUN;
 PROC PRINT DATA=PAYROLL;
 TITLE  "DATASET PAYROLL CONTAINS 50 OBSERVATIONS";
 RUN;
```

Notes:

SYMGET returns macro variable values to the DATA step.

A SAS Macro Application

The following problem needs some help.

Data Set COUNTYDT

Obs	COUNTYNM	READING
1	ASHLAND	125
2	ASHLAND	611
3	BAYFIELD	101
4	BAYFIELD	101
5	BAYFIELD	222
6	WASHINGTON	143

Each day you read a SAS dataset containing data from counties in Wisconsin.
Anywhere between 1 and 72 counties might report that day. Do the following:

1. Create a separate dataset for each reporting county.
2. Produce a separate PROC PRINT for each reporting county.
3. In the TITLE print the county name.
4. Reset the page number to 1 at the beginning of each report.
5. In a footnote print the number of observations processed for each county.

How do you do it?
Solution: Write a SAS Macro.

The Solution

A multi-step macro can communicate between steps.

```
DATA _NULL_;
 SET COUNTYDT END=EOF;        /* READ SAS DATASET      */
 BY COUNTYNM;                 /* SORT SEQ              */
 IF FIRST.COUNTYNM THEN DO;   /* NEW COUNTY ?          */
    NUMCTY+1;                 /* ADD 1 TO NUMCTY       */
    CTYOBS=0;                 /* OBS PER COUNTY TO 0   */
    END;
 CTYOBS+1;                    /* ADD ONE OBSER FOR CTY */
IF LAST.COUNTYNM THEN DO;     /* EOF CTY, MAKE MAC VARS*/
  CALL SYMPUT('MCTY'||LEFT(PUT(NUMCTY,3.)),COUNTYNM);
  CALL SYMPUT('MOBS'||LEFT(PUT(NUMCTY,3.)),
       LEFT(CTYOBS));
  END;
 IF EOF THEN                  /* VERY LAST OBS ?       */
    CALL SYMPUT('MTOTCT',NUMCTY);
                              /* MAC VAR NO DIF CTYS   */
RUN;
%PUT *** MTOTCT=&MTOTCT;      /* DISPLAY NO OF CTYS    */

%MACRO COUNTYMC;              /* MACRO START           */
%DO I=1 %TO &MTOTCT;          /* LOOP THRU ALL CTYS    */
 %PUT *** LOOP &I OF &MTOTCT; /* DISPLAY PROGRESS      */
 PROC PRINT DATA=COUNTYDT;    /* PROC PRINT            */
  WHERE COUNTYNM="&&MCTY&I";  /* GENERATED WHERE       */
  OPTIONS PAGENO=1;           /* RESET PAGENO          */
  TITLE "REPORT FOR COUNTY &&MCTY&I";
                              /* TITLES AND FOOTNOTES  */
  FOOTNOTE "TOTAL OBSERVATION COUNT WAS &&MOBS&I";
 RUN;
%END;                         /* END OF %DO            */
%MEND COUNTYMC;               /* END OF MACRO          */
%COUNTYMC                     /* INVOKE MACRO          */
```

The Generated Code and Output

```
*** MTOTCT=3
*** LOOP 1 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="ASHLAND";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY ASHLAND";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 2";   RUN;
*** LOOP 2 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="BAYFIELD";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY BAYFIELD";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 3";   RUN;
*** LOOP 3 OF 3
   PROC PRINT DATA=COUNTYDT;
   WHERE COUNTYNM="WASHINGTON";   OPTIONS PAGENO=1;
   TITLE "REPORT FOR COUNTY WASHINGTON";
   FOOTNOTE "TOTAL OBSERVATION COUNT WAS 1";  RUN;
```

REPORT FOR COUNTY ASHLAND

OBS	COUNTYNM	READING
1	ASHLAND	125
2	ASHLAND	611

TOTAL OBSERVATION COUNT WAS 2

REPORT FOR COUNTY BAYFIELD

OBS	COUNTYNM	READING
3	BAYFIELD	101
4	BAYFIELD	101
5	BAYFIELD	222

TOTAL OBSERVATION COUNT WAS 3

REPORT FOR COUNTY WASHINGTON

OBS	COUNTYNM	READING
6	WASHINGTON	143

TOTAL OBSERVATION COUNT WAS 1

Exercise 8:

A very common problem is to run a proc against all members of a SAS library.
The following program produces a SAS dataset containing an observation for each variable within each dataset in the library. A partial print is shown below.

```
PROC CONTENTS DATA=WORK._ALL_ NOPRINT OUT=CONTOUT;
 RUN;
PROC PRINT DATA=CONTOUT;
 VAR LIBNAME MEMNAME NAME;
 TITLE 'CONTOUT';
RUN;
```

CONTOUT

OBS	LIBNAME	MEMNAME	NAME
1	WORK	FREQ	DEPT
2	WORK	FREQ	SALARIES
3	WORK	FREQ	SALES
4	WORK	FREQ	YEAR
5	WORK	YEAR1988	EMPLOYDT
6	WORK	YEAR1988	EXPENSES
7	WORK	YEAR1988	NAME
8	WORK	YEAR1988	REGION
9	WORK	YEAR1988	SALES
10	WORK	YEAR1988	STATE

Write a SAS macro that will generate a separate PROC PRINT of all members in the library with an appropriate title. Note you will only want to produce one print per member, not one per variable.

Exercise Solutions

Exercise 1

```
%inc `a:\sasm0001.sas';
proc contents data=work._all_;run;
```

Exercise 2

(The following code can be entered to see the results of these System Macro variables)

```
%PUT **** SYSDAY  = &SYSDAY;
%PUT **** SYSTIME = &SYSTIME;
%PUT **** SYSSCP  = &SYSSCP;
%PUT **** SYSVER  = &SYSVER;
%PUT **** SYSDATE = &SYSDATE;
```

Exercise 3

```
proc contents data=&syslast;
 Title "Contents of &syslast";
run;
```

Exercise 4

A) CANES

B) FISCAL YEAR 1989

C) YEAR ENDED DECEMBER 31, 1989

D) 1989 CANDY CANES WERE SOLD IN 1989

E) 1989 WAS A "TREMENDOUS" SALES YEAR!

Exercise 5

A) WIND SURFING

B) WIND AND WATER SURFING = FUN

C) H&DY MAN WARNING:
Apparent symbolic reference DY not resolved.

D) HANDY MAN

E) HANDY.MAN

```
F)    SURFING FUN

G)    SNOW MAN

H)    PROC PRINT; RUN;
```

Exercise 6

```
   PROC FREQ DATA=FREQ;
     TABLES DEPT*SALES / NOPERCENT;  RUN;
```

Exercise 7

```
  A)   PROC FREQ DATA=FREQ;
         TABLES SALARIES*SALES / NOPERCENT;
       RUN;
```

The following code would be passed to the SAS compiler and would be rejected due to syntax error:

```
  B)   PROC FREQ DATA=;
         TABLES * / NOPERCENT;
       RUN;

  C)   PROC FREQ DATA=FREQ;
         TABLES SALARIES* / NOPERCENT;
       RUN;
```

Exercise 8

```
PROC CONTENTS DATA=WORK._ALL_ NOPRINT OUT=CONTOUT;
 RUN;
 PROC PRINT DATA=CONTOUT;
 VAR LIBNAME MEMNAME NAME;
 TITLE 'CONTOUT';


RUN;
DATA ONEMEM;
 SET CONTOUT END=EOF;
  BY MEMNAME;
  IF LAST.MEMNAME;
  KTR+1;
  CALL SYMPUT ('MMEM'||LEFT(PUT(KTR,5.)),MEMNAME);
  IF EOF;
  CALL SYMPUT ('MTOTOBS',LEFT(PUT(KTR,5.)));
RUN;
%MACRO PRTLOOP;
 %DO I = 1 %TO &MTOTOBS;
     PROC PRINT DATA=&&MMEM&I;
        TITLE "&&MMEM&I";
     RUN;
 %END;
%MEND PRTLOOP;
%PRTLOOP
```

Steven First
Systems Seminar Consultants
2997 Yarmouth Greenway
Madison, WI 53711
Phone: (608) 278-9964
Fax: (608) 278-0065
E-mail: sfirst@sys-seminar.com