INTRODUCTION.
The SAS macro facility is a very dynamic tool in SAS that enables users to write programs that are general, flexible, and reusable. The facility's capabilities include:

- passing variables throughout code in order to perform conditional activities or to create conditional SAS statements,
- storing a set of SAS statements that are identified and invoked by a name,
- generating multiple DATA and PROC steps using iterative loops in order to condense the amount of code that would have to be written otherwise, and
- creating interactive systems via simple windows.

The development of these capabilities involves the utilization of three main macro components: SAS macro variables, SAS macros and macro windows.

SAS MACRO VARIABLES.
SAS macro variables are independent of SAS data sets and variables. They can be either system-defined, user-defined or data-dependent and can be created and used anywhere in the SAS code, except in data lines. Once a macro variable is defined, its value does not change unless it is initialized or redefined by the user or programmer (see EX. 1).

SAS MACROS.
SAS macros are sets of stored SAS statements that are identified by a name. When their respective names are invoked in the program the stored commands are executed (see EX. 2).

SAS MACRO WINDOWS.
SAS macro windows allow you to create a simple interactive system with users. These windows can create displays with text that users can read and input fields into which they can enter values that define macro variables (see EX. 3).
%function(argument);

where argument is any macro expression. These functions can be divided into three categories:

1. Character functions change or provide information about the string that is their argument.
2. The evaluation function treats macro variable integers as numeric values and performs simple arithmetic calculations only (see EX. 5).
3. Finally, quoting functions perform the activity equivalent to enclosing a portion of a SAS statement in single or double quotes; i.e., they allow the symbols ;, %, and & to be treated as text.

A list of several of these functions can be found in SAS Guide to Macro Processing, pp. 117-118.

DATA STEP INTERFACES.
There are two macro routines which can be used in the DATA step to process macro variables.

The call symput routine assigns a value produced in the DATA step to a macro variable. It enables you to create data-dependent macro variables and to transfer information from SAS output to macro variables. This routine has the general form

call symput(macro-variable,data-step-information);

which creates macro-variable with the value identified in data-step-information. Either argument can be character literals or DATA step character values/strings.

In EX. 6, target is the macro variable whose value is identified by the data variable control.

The symget function does the opposite, returning a character value (whose maximum length is 200) of a macro variable to the DATA step during DATA step execution. This function has the form

symget(argument);

where argument identifies a macro variable.
SAS MACROS

INTRODUCTION.
SAS macros are stored SAS statements that are referred to by a name. This section will discuss how to define, use, and pass parameters to SAS macros. It will also talk about how to manipulate SAS statements within the macro with do-loops and if-then/else statements.

DEFINING A MACRO.

General.
All macros begin with the %macro statement, followed by the body of the macro, and ending with the %mend statement:

```
%macro name <options>;
  macro expressions and/or SAS statements;
%mend name;
```

(name is the name of the macro and must be a valid SAS name)

With Parameters.
One of the options in the %macro statement allows the user to pass n parameters to a SAS macro via the statement:

```
%macro name(var1 var2 ... varn);
  macro expressions and/or SAS statements
%mend name;
```

where var1, var2, ... varn are the local macro variables created specifically for use within the macro name (see Local Macro Variables later in this section for more details).

This statement decreases the amount of code needed to create a macro variable within a SAS macro by eliminating the %let and %local statements.

EX. 7 shows how to define a macro called lotname using the general macro statement and the macro statement with a parameter. This macro creates titles based on the macro variable lot.

USING A MACRO.

General.
After the macro is defined, it is invoked with the

```
EX. 9: Iterative %do %to loop

This program generates two sub-data sets, one for each lot, from the data set devices. It also creates the macro variable, target, for each data set.

%macro subdev;
  %do i=1 %to 2;
    data device&i;
    set devices;
    if lot=&i;
    call symput ('target', target);
  run;
  %end;
%mend subdev;

Code generated:
```

```text
data device1;
set devices;
if lot=1;
call symput ('target', target);
run;
data device2;
set devices;
if lot=2;
call symput ('target', target);
run;
```

EX. 10: An infinite %do loop

This example creates an infinite %do %until loop.

```
%macro dontdo;
  %do <10 %until(&value>20);
    proc print;
    run;
  %let value=&value+1;
  %end;
%mend dontdo;
```

EX. 11: Using an %if-%then / %else statement

This program creates specific graphs for two different controls using an open %if-%then-%else statement.

```
%macro graphs;
  %let data1=devicel;
  %let data2=device2;
  %if &target=20 %then
    %do;
    proc gplot data=&data1;
    plot resp*week;
    run;
    %end;
  %else %if &target=150 %then
    %do;
    proc gplot data=&data2;
    plot resp*week;
    run;
    %end;
  title3 "Target Response - &target units";
%mend graphs;
```
Macros with parameters are invoked by the statement

\%name(valuel value2 \ldots valuen)

where valuel value2 \ldots valuen are the values assigned to the macro variables varl, var2, \ldots varn, respectively.

EX. 8 shows how macros are used and how they effect SAS statements.

ITERATIVE DO LOOPS.
The iterative %do loop executes a portion of a macro repetitively, depending on the value of the macro variable. Its form is:

\%do macro-variable=start %to stop < %by increment > ;
    macro statements;
\%end;

where macro-variable is a macro variable or a macro expression that generates a macro variable name. The values of start, stop, and increment are integers or macro expressions that yield integers.

EX. 9 demonstrates how to use a %do loop to create a data set specific for each strip lot. Note that when the macro is executed it produces code equivalent to having two separate data steps.

DO UNTIL LOOP.
The %do %until statement executes the statements in a loop repetitively until a condition becomes true. This loop is always executed at least once because the value of the condition is checked at the end of the loop. The form of the statement is:

\%do %until(expression);
    macro statements;
\%end;

where expression is any macro expression. (Note: Be careful that you do not create an infinite loop, see EX. 10)

DO WHILE LOOP.
The %do %while statement executes a group of statements repetitively while a condition remains true. This loop may not be executed at all because the loop tests the condition at the top of the loop. It has the form

\%do %while(expression);
    macro statements;
\%end;

(Note: Be careful that you do not create an infinite loop.)

IF THEN ELSE STATEMENTS.
The %if-%then / %else statement is analogous to the if-then and else statements in the DATA step and has a similar form:

\%if expression %then expression;
< %else expression; >

EX. 11 displays how this statement is used to creates different graphs for target=20 and target=150.

MACRO VARIABLES.
By default macro variables are created in what is called the global referencing environment. This means that the macro variable values are retained across DATA and PROC steps and can be changed inside or outside of any macro at anytime during the SAS session. As a general rule it is not good programming practice to use global macro variables.

If you must use global macro variables you should explicitly define a macro variable as global so that it is easily identified. The SAS statement that creates global macro variables is

\%global variable(s) ;

All system-defined macro variables are global.

LOCAL MACRO VARIABLES.
Local macro variables are defined within a macro with the statement

\%local variable(s) ;

where variable(s) lists the macro variable created in the %let statement.

Macro variables defined in this manner retain their values across DATA and PROC steps only if they are
executed from within the same macro. This ensures that values of macro variables created in previously executed macros are not inadvertently affected by values assigned in the current macro.

SAS MACRO WINDOWS

INTRODUCTION.
SAS macro windows are a quick and easy way to develop a simple interactive system in SAS. Although they are sufficient for many applications, complex interactive systems should be developed in SAS/AF software.

DEFINING A WINDOW.
The `%window` statement defines a simple macro window:

```
%window window-name <options> field ... 
    | group=group-name field ...;
```

where `window-name` names the macro window, `options` specify window characteristics, `field` defines the variable or text string to display in the window, and `group=` defines a group of fields to display together.

WINDOW FIELDS.
An individual item in a window is called a field. Window fields can be macro variables or text strings whose placements, text color and highlighting can be controlled. The general form of a field definition is

```
<location> item-to-display <options>.
```

`Location` identifies the row and column position of the field in the form

```
<row> <column>.
```

The row pointer controls are:

- `#n` which specifies that the field will appear on the n\(^{th}\) row of the window
- `/` which moves the pointer to the next row.

The column pointer controls are:

- `@n` which specifies that the field will begin on the n\(^{th}\) column of the window

- `+n` which moves the pointer n columns to the right.

`Item-to-display` can be

1. text enclosed in single or double quotes,
2. a macro variable name without an ampersand or
3. an expression enclosed in quotes that generates a macro variable name (see EX. 3).

`Options` define field color and attributes such as underlining, highlighting, and protection.

WINDOW GROUPS.
The `group=` option allows you to define several sets of fields in the same window and to display them individually. This option helps decrease the number of windows that would have to be created otherwise and avoids repeating window options that do not change from window to window.

For example, the user may need to select either control A or control B, whose data needs to be analyzed. Instead of creating two windows (one for control A and one for control B), a simple solution would be to create one window with two groups as shown in EX. 12.

REFERENCE.
The `%display` statement displays one group of fields in a macro window. If a window has several input fields, the user should press `<TAB>` in order to move among the fields. The user should press `<ENTER>` only if he is ready to remove the displayed window and to continue the program. A window without the group option has the form

```
%display window <options>;
```

where `window` names the window to be displayed and `options` define window attributes such as ringing a bell when the window is displayed.

A window with groups has the form

```
%display window.group <options>;
```

where `window.group` names both the window and the specific group of fields to display.
ENHANCEMENTS IN 6.07

The following list of macro language features are several of the enhancements made in SAS 6.07:

AUTOMATIC MACRO VARIABLES.
SYSFILRC indicates whether the last FILENAME statement executed correctly (SYSFILRC=0).

MACRO FUNCTIONS.
%SYSPROD(argument) returns a value indicating whether a SAS Institute software product is licensed where argument is any macro expression that produces a string. One of the following values may be returned:
- 1 valid value licensed as part of the SAS System
- 0 valid value not licensed as part of the SAS System
- -1 not a valid Institute product

DATA STEP INTERFACES.
There are two data-step macro statements that return the resolved value of an argument after it has been processed by the macro facility:

1. call execute(argument);
2. data-step-variable = resolve(argument);

where data-step-variable is the name of the DATA step variable.

In both cases, argument can be a character string that represents a macro expression, the name of a DATA step variable whose value is a macro expression, or a character expression that produces a macro expression.

EX. 13 shows how the call execute routine can be used to print out a data set if it is not empty.

EX. 14 demonstrates several ways that the resolve function can be utilized in a DATA step.

STORED COMPILED MACRO FACILITY.
This facility in SAS allows you to store a compiled macro in a SAS catalog named SASMACR in a permanent SAS data library as shown in EX. 15. (This facility is not useful for macro development purposes and is recommended only for large production-level jobs.)
SAS MACRO TIPS

PROGRAMMING.

For macro variables
• try to use local variables rather than global variables.
• minimize the number of variables used to reduce overhead.

For macros
• avoid resolving expressions inside a loop.
• use the stored compiled macro facility.

For macro windows
• initialize all macro variables at the beginning of the program.
• take advantage of the automatic macro variable &sysmsg to inform the user if he has entered a value incorrectly.
• for better efficiency, create windows outside macros and loops.
• create a window that tells the user that the program is processing
• the submitted statements if there is a time lag between information windows.

AS ALWAYS
• include comment lines in your program.
• use programming structure.
• create a header at the beginning of your program stating the author of the program, the date it was accepted for use, the name of the file, and the procedures that it uses. Other information can be included at your own discretion (see EX. 7).

DEBUGGING.
Use the options statement
options mprint symbolgen mlogic;
at the beginning of your program to help you debug the errors in your program.
• MPRINT shows all SAS statements generated by macro code.
• SYMBOLGEN shows the resolved values of the macro variables in the program.
• MLOGIC shows when each statement within a macro executes.

EX. 13: Call Execute routine
/*******************
This program prints the data set meter if it is not empty.
/*******************
%macro out;
proc print data=meter;
run;
%mend out;
data device;
set device;
if _N_ ne 0 then call execute('%out');
run;

Output: The SAS System 07:14 Tuesday, May 25, 1993 1
OBS D1 D2 D1
1 12 26 39
2 14 46 99
3 38 48 2
4 38 48 24
5 84 74 20

EX. 14: RESOLVE function
/*******************
This program demonstrates what the RESOLVE function is capable of doing in a data step.
/*******************
%let tstand=Blackbox;
%let person=Geoff;

%macro type;
717
%mend type;
data PCS;
length var1-var4 $ 20;
type='%type';
var1=resolve('&tstand');
var2=resolve('%type');
var3=resolve(type);
var4=resolve('&person' leftLN());
run;
proc print data=PCS;
var var1-var4;
run;

Output: The SAS System 07:14 Tuesday, May 25, 1993 1
OBS VAR1 VAR2 VAR3 VAR4
1 Blackbox 717 717 Geoff
When the program is acceptable to use, you may consider removing these options via a comment line in order to save CPU time.

EX. 15: Using the store compiled macro facility

/**************************************************/
This program creates a Stored Compiled Macro in one session.
/**************************************************/
libname PCSlib "Volset_5:[dept_805_work]";
options mstored sasmstore=PCSlib;

%macro lotname / store;
%let lo=504;
title1 "Easy Strip Stability";
title2 "Strip Lot &&lot";
%mend lotname;

/**************************************************/
This program uses the Stored Compiled Macro, lotname, in a later session.
/**************************************************/
libname PCSlib "Volset_5:[dept_805_work]";
options mstored sasmstore=PCSlib;

%lotname;
proc means data=lot3;
run;

REFERENCES


