STAT 541

Chapter 9: Creating and Using Macro Programs

Definition

MACRO (noun)

Etymology: short for macroinstruction

Date: 1959

: a single computer instruction that stands for a sequence of operations

Retrieved from www.m-w.com

Macro Programs (a.k.a. Macros)

- Macros enable *text substitution* into programs.
- Unlike macro variables, macros can use conditional logic.
- Programs can become more dynamic and reusable, shorter, and easier to follow.
- It is useful for automation. Repetitive tasks can be performed quickly and efficiently. Macros can generalize the same or similar code. Parameters can be passed to the macro so that the macro itself does not have to be changed prior to execution.
- It can help towards modular programming. Referencing a macro is similar to calling a subroutine. The main program can become more readable.

What is the SAS Macro Facility?

- It is a tool for text substitution that is meant to assist you in constructing your programs. The facility is part of base SAS. It has its own language that is different from that used in base SAS, but the two languages have similar conventions and syntax.

What are the potential disadvantages of using the SAS Macro Facility?

- Used a certain way, macros can make your programs harder to read.
- Sharing macro programs with users who do not use macros has its challenges.

What are the SAS Macro Facility's two components?

- The macro processor compiles macros and integrates it with the SAS job.
- The macro language is the interface between the programmer and the processor.
- The macro language statements instruct the macro processor how to substitute text for you or how the facility should create the statements for you. After writing a macro, the macro is executed by invoking it instead of manually modifying the statements in a non-macro program prior to execution.

How is the macro processor triggered in a SAS program?

- &varname

This is a reference to the macro variable named *varname*. The current value of the macro variable will replace all references of &varname in the program.

- %macroname

This is a reference to the macro named *macroname*. This generates statements (with or without errors) that are contained in the macro. The contents of the macro are still subject to debugging by the user.

How do I use SAS macros?

- The first step is to create macro statements/code.
- The second step is to invoke the macro statements/code.

Defining a Macro

- The definition begins with %MACRO and ends with %MEND.
- Example:

```
%macro printsubset(gender,titletext);
proc print; where female="&gender";
title "&titletext";
run;
%mend printsubset;
```

General Syntax

%MACRO *macro-name*;

text

%MEND <*macro-name*>;

- *macro name*is any valid SAS name but not a reserved word in the SAS macro facility
- *text*is:
  – constant text, possibly including SAS data set names, SAS variable names, or SAS statements
  – macro variables, macro functions, or macro program statements
  – any combination of the above

Compiling a Macro

- After submitting the macro definition, the macro scanner goes through it and sends it to the macro processor.

The macro processor:

- Checks for macro language syntax errors (non-macro language statements are NOT checked until AFTER the macro is executed)
- Writes error messages to the SAS log and creates a *dummy (non-executable)* macro if there are errors
- Stores the macro for later use if there are no errors. Stores all compiled macro language statements and constant text in a SAS catalog (default is Work.Sasmacr catalog) under an entry called *macro-name.Macro.*

The MCOMPILENOTE Option

This option will write a note in the SAS log when a macro has completed compilation.

OPTIONS MCOMPILENOTE= NONE | NOAUTOCALL | ALL;

- Default NONE (no notes written to the log)
- NOAUTOCALL (notes to log about completed macro compilation except for AUTOCALL macros)
- ALL (notes to log about all completed macro compilation)

Calling a Macro

- After compilation, the macro can be called for the duration of the SAS session without resubmitting it.

Macro calls:

- are specified by placing a percent sign (%) before the name of the macro
- can be made anywhere in a program except within the data lines of a DATALINES statement
- do not require semicolons (Macro calls are not SAS statements)

Example: %printsubset calls the macro named printsubset

Macro Execution

- Word scanner passes macro call to macro processor, which searches the SAS catalog (usually WORK.Sasmacr) for the macro.
- Compiled macro language statements are executed
- Non-macro text is scanned
- Macro execution is halted at end of SAS step, and SAS code is executed
- Macro execution is resumed

Developing and Debugging Macros

- A number of SAS system options are available.
  - MPRINT Option
  - MLOGIC Option
- Comments can be added to the program

The MPRINT Option

This option shows in the SAS log the code that results from the macro.

OPTIONS MPRINT | NOMPRINT;

- MPRINT (displays statements generated by macro execution-- statements are useful for debugging macros)
- default NOMPRINT (does not display statements generated by macro execution)

The MLOGIC Option

This option prints messages that indicate macro actions that were taken during macro execution.

OPTIONS MLOGIC | NOMLOGIC;

- MLOGIC (messages about macro actions are printed in the log during macro execution) debugging
- default NOMLOGIC (messages about macro actions are not printed in the log)

The Macro Comment Statement

Macro comments are not part of the code that results after the macro is compiled. Regular SAS comments are.

*%*comment;*

- *comment* is any message
- The statement ends with a semi-colon.

Using Macro Parameters

A parameter list is an optional part of the %MACRO statement that names one or more macro variables whose values are specified upon calling the macro.

Example:

The

```
%macro printsubset(gender,titletext);
proc print; where gender="&gender";
title "&titletext";
%mend printsubset;
```

The macro prints the variables for records with the specified gender and titletext macro variable values.

General Syntax for %MACRO Statement

%MACRO *macro-name* <(*parameter-list*)></ *option-1* <*...option-n*>>;

- *macro-name* names the macro (a SAS name that is not a reserved word in the macro facility)
- *parameter-list* names one or more local macro variables whose values are specified when the macro is invoked (Macro parameters are separated by commas and can be referenced in the macro.)

Syntax for Macros with Positional Parameters

%MACRO *macro-name*

(positional-parameter-1 <. . . *,positional-parameter-n*>)

text

%MEND *macro-name;*

- Parameters can be in any order.
- Macro invocation must have them in the same order as they appear in the %MACRO statement.
- Separate more than one parameter with commas.
- If an invocation does not supply a value for a positional parameter, a null value is assigned to that parameter.

Syntax for Macros with Keyword Parameters

%MACRO *macro-name*

(keyword-parameter=<value><. . . ,keyword-parameter-n=<value>>)

text

%MEND *macro-name;*

- Keyword parameters name one or more macro parameters followed by equal signs. Default values can follow the equal signs. An omitted default value is the null value.
- Override a default value by specifying the macro variable name followed by the equal sign and new value in the invocation.

Syntax for Macros That Include Mixed Parameter Lists

%MACRO *macro-name*

(positional-parameter-1 <. . . *,positional-parameter-n*>,

keyword-parameter=<*value*><. . . *,keyword-parameter-n=*<*value*>>)

text

%MEND *macro-name;*

- Parameter lists can consist of both positional and keyword parameter lists.
- Positional parameters must be listed first before any keyword parameters.

Syntax for Macros That Include the PARMBUFF Option

%MACRO *macro-name /PARMBUFF;*

text

%MEND *macro-name;*

- PARMBUFF option creates an automatic macro variable SYSPBUFF to define a macro that accepts a varying number of parameters each time you call it.
- SYSPBUFF has the value of the list of parameters separated by commas and all enclosed in parentheses.
- *text* contains a reference to the automatic macro variable SYSPBUFF

The Global Symbol Table

- The table is created during the initialization of a SAS session and is deleted at the end of the session. The table contains global macro variables that:
  - are available any time during the session
  - can be created by a user
  - have values that can be changed during the session (except for some automatic variables)
- Create a global macro variable with:
  - a %LET statement (used outside a macro definition)
  - a DATA step that contains a SYMPUT routine
  - a DATA step that contains a SYMPUTX routine
  - a SELECT statement that contains an INTO clause in PROC SQL
  - A %GLOBAL statement

The %GLOBAL Statement

The %GLOBAL statement:

- Creates one or more macro variables in the global symbol table and assigns null values to them
- Can be used either inside or outside a macro definition
- Has no effect on variables that are already in the global symbol table

%GLOBAL *macro-variable-1 <...macro-variable-n>*;

- *macro-variable-1 <...macro-variable-n>*is the name of one or more macro variables or a text expression that generates one or more macro variable names

The Local Symbol Table

- When a macro variable is in a local symbol table, it is available only during execution of the macro in which it is defined.
- The local symbol table contains macro variables that can be:
  - created and initialized at macro parameter invocation
  - created or updated during macro execution
  - referenced anywhere within the macro
- Create a local macro variable *<u>within a macro definition</u>*:
  - parameters in a macro definition
  - a %LET statement within a macro definition
  - a DATA step that contains a SYMPUT routine within a macro definition
  - a DATA step that contains a SYMPUTX routine within a macro definition
  - a SELECT statement that contains an INTO clause in PROC SQL within a macro definition
  - A %LOCAL statement

The %LOCAL Statement

The %LOCAL statement:

- Can appear only inside a macro definition
- Creates one or more macro variables in the local symbol table and assigns null values to them
- Has no effect on variables that are already in the local symbol table

%LOCAL *macro-variable-1 <...macro-variable-n>*;

- *macro-variable-1 <...macro-variable-n>*is the name of one or more macro variables or a text expression that generates one or more macro variable names

Multiple Local Symbol Tables

- Multiple local symbol tables can exist *concurrently* during macro execution if there are *nested macros.* If a macro program calls another macro program, and if both macros create local symbol tables, then two local symbol tables will exist while the second macro executes.

The MPRINTNEST Option

This option allows the macro nesting information to be written to the SAS log in the MPRINT output. This has no effect on the MPRINT output that is sent to an external file. MPRINT and MPRINTNEST must both be set in order for the output to be written to the log.

```
OPTIONS MPRINTNEST | NOMPRINTNEST;
```

- MPRINTNEST specifies that macro nesting information is written in the MPRINT output in the log
- NOMPRINTNEST specifies that macro nesting information is not written in the MPRINT output in the log

The MLOGICNEST Option

This option allows the macro nesting information to be displayed in the MLOGIC output in the SAS log. The setting does not affect the output of any currently executing macro.

```
OPTIONS MLOGICNEST | NOMLOGICNEST;
```

- MLOGICNEST specifies that macro nesting information is written in the MLOGIC output in the log
- NOMLOGICNEST specifies that macro nesting information is not written in the MLOGIC output in the log

Processing Statements Conditionally

%IF-%THEN/%ELSE Statement

%IF *expression* %THEN *action*;

<%ELSE *action*;>

- *expression*is any macro expression (constant text, a text expression, a macro variable reference, a macro call, or a macro program statement) that resolves to an integer.
- Comparisons are case-sensitive.
- If the expression resolves to an integer other than zero, the expression is true and the %THEN clause is processed.
- If the expression resolves to zero, then the expression is false and the %ELSE statement, if any, is processed.
- If the expression resolves to a null value or a value with nonnumeric characters, the macro processor issues an error message.
- %IF comparisons are case-sensitive.

Processing Statements Conditionally

- Use *%DO-%END with %IF-%THEN and %ELSE statements* in order to conditionally place text that contains multiple statements onto the input stack.

```
%IF expression %THEN %DO;
text and/or macro language statements
%END;
%ELSE %DO;
        text and/or macro language statements
%END;
```

- *text and/or macro language statements* is either constant text, a text expression, and/or a macro statement.

Processing Statements Iteratively

%DO *macro-variable=start* %TO *stop* <%BY *increment*>;

text and macro language statements

%END;

- *macro-variable* is a macro variable or a text expression that generates a macro variable name (functions as index that determines number of loop iterations)—If macro variable does not exist, it is created in the local symbol table.
- *start* and *stop* are integers or macro expressions that generate integers to help control the total number of iterations
- *increment* is a positive integer (default is 1) that is added to the index variable in each loop iteration (*Increment* is evaluated before the first iteration of the loop and can't be changed as the loop iterates.)
- The index variable is evaluated at the beginning of each loop. The loop ends when the index variable first exceeds the *stop* value.

Using Arithmetic and Logical Expressions the %EVAL Function

- The %EVAL function evaluates arithmetic and logical expressions using *integer arithmetic*.

%EVAL (*arithmetic or logical expression*)

- Caution: Error messages are generated in the log when the expression contains non-integer values. The function *does not* convert a value that contains a period to a number.

Using Arithmetic and Logical Expressions the %SYSEVALF Function

- The %SYSEVALF function evaluates arithmetic and logical expressions using *floating-point arithmetic*.

%SYSEVALF (*expression, <conversion-type>);*

- The second parameter *conversion-type* is optional and can be BOOLEAN, CEIL, FLOOR, or INTEGER.
- This is the only macro function that can evaluate expressions with floating point or missing values.