PharmaSUG 2019 - Paper AP-191

Creating In-line Style Macro Functions

Arthur Li, City of Hope National Medical Center, Duarte, CA

ABSTRACT

Macro functions that are used in our program are defined by the macro facility. By providing values for the function parameters, the macro function generates a result. We can insert the result directly into a macro statement in our program. Programmers seldom know that we can create a user-defined macro function as well. This paper will focus on the methods of creating an in-line style macro function via various examples.

INTRODUCTION

In programing languages, a function is used to process one or more arguments and generate a result. Similarly, a macro function also returns a result based on provided arguments. SAS supplies many built-in macro functions, such character and quoting functions, that can be used in both macro definitions and open code. These built-in functions, in conjunction with other macro language elements, provide us flexibility for generating SAS codes in our program.

One can create a SAS macro function as well. Creating a SAS macro function is like creating a macro, except that we need to know how to make the macro behaves like a function. The critical component to make a macro like a function is knowing how to transfer a value outside the macro. Once the macro function is created, we can then call this macro function with argument(s) and expect this macro to return the calculated result for us.

A SIMPLE EXAMPLE

Suppose we would like to create a macro function, %CAL_CIRCLE, that is used to calculate area, circumference or volume of a circle. This function takes two arguments. The first argument of the macro is METHOD, which takes values of AC, CC, or VS in either upper, lower, or mixed cases. The second argument of the macro is R, which is the radius of a circle. For example, if we want to calculate the area of a circle with radius equaling to 5, we would call %CAL_CIRCLE (AC, 5). To calculate the circumference of a circle with radius equaling to 10, we would call %CAL_CIRCLE (CC, 10), etc. The macro in Program 1 accomplishes this task and this macro returns the calculated result in the %PUT statement. The way that this macro returns the calculated result is discussed in the following section.

Program 1:

```
%macro cal_circle1(method, r);
    %local result;
    %let method = %upcase(&method);
    %if &method = AC %then %let result = %sysevalf(3.14159*&r*&r);
    %else %if &method = CC %then %let result = %sysevalf(2*3.14159*&r);
    %else %if &method = VS %then %let result = %sysevalf(4*3.14159*(&r**3)/3);
    &result
%mend cal_circle1;

%put The area of circle with radius equaling to 5 is %cal_circle1(AC, 5);
```

SAS Log from Program 1:

```
10   %put The area of circle with radius equaling to 5 is &result;
The area of circle with radius equaling to 5 is 78.53975
```

TRANSFERRING A VALUE OUTSIDE THE MACRO

Macro programs are used to generate SAS code for us and make our code-writing efficient. The generated SAS code often contains DATA or PROC steps that are used for creating SAS data sets, displaying statistical output, generating tables, or figures. Passing value outside a macro has seldom be the main concern of writing a macro program.

One way to pass a value outside a macro is to store the value in a macro variable in the global symbol table. For example, Program 2 is similar to Program 1 except that the calculated result is stored in the global macro variable &RESULT. The value from &RESULT is extracted in the %PUT statement by using direct referencing.

Program 2:

```
%macro cal_circle2(method, r);
    %global result;
    %let method = %upcase(&method);
    %if &method = AC %then %let result = %sysevalf(3.14159*&r*&r);
    %else %if &method = CC %then %let result = %sysevalf(2*3.14159*&r);
    %else %if &method = VS %then %let result = %sysevalf(4*3.14159*(&r**3)/3);
%mend cal_circle2;

%cal_circle2(AC, 5)
%put The area of circle with radius equaling to 5 is &result;
```

SAS Log from Program 2:

```
17   %put The area of circle with radius equaling to 5 is &result;
The area of circle with radius equaling to 5 is 78.53975
```

The main disadvantage of passing value via a global macro variable is that it might conflict an existing macro variable in the global symbol table. Furthermore, using this approach breaks the flow of the program since we have to call the macro first, then reference the macro variable in the %PUT statement.

In order to let a macro behave like a function, the macro needs to return a value. In Program 1, the returned value is stored in the local macro variable &RESULT by using the %LET statement. Using a local macro variable to store the returned result is to ensure that there will be no conflicts between local macro variables the one that might have already existed in the global symbol table. After &RESULT is resolved, the resolved value becomes a SAS language element. The SAS language element is pushed to the input stack and left in the %PUT statement after the macro call.

Program 3 also accomplishes the same task. In this program, the returned value is simply placed in the %THEN clause within the %IF-%THEN statement. The evaluated text from the %SYSEVALF function is no longer a macro language element and cannot be executed by the macro processor. Therefore, the value that is calculated from the %SYSEVALF function is pushed to the input stack.

Program 3:

```
%macro cal_circle3(method, r);
    %let method = %upcase(&method);
    %if &method = AC %then %sysevalf(3.14159*&r*&r);
    %else %if &method = CC %then %sysevalf(2*3.14159*&r);
    %else %if &method = VS %then %sysevalf(4*3.14159*(&r**3)/3);
%mend cal_circle3;

%put The area of circle with radius equaling to 5 is %cal_circle3(AC, 5);
```

SAS Log from Program 3:

```
25   %put The area of circle with radius equaling to 5 is %cal_circle3(AC, 5);
The area of circle with radius equaling to 5 is 78.53975
```

RENAMING VARIABLES

We often need to rename a list of variables by placing a prefix in front of the variable names. Renaming variables can be done by either using the RENAME statement or the RENAME data set option. Regardless whether using the RENAME statement or the data step option, we need to construct a code fragment like OLD_VAR1 = NEW_VAR1 OLD_VAR2 = NEW_VAR2 OLD_VAR3 = NEW_VAR3 …

The macro program RENAMES in Program 4 constructs the code segment by taking a user-specified prefix and a list of variable names. The returned results from the macro can be either inserted in the RENAME statement or the RENAME data set step. For example, calling **`%RENAMES(new_, a b c)`** returns **`a = new_a b = new_b c = new_c`**.

More specifically, when calling %RENAME(new_, a b c), two local variables, &PRE and &NAMELIST are created with corresponding values new_ and a b c. Next, the %LOCAL statement ensures that both &N and &I are local variables in the macro. Then the %LET statement stores the number of variables in &N variable by using a DATA

step function COUNTW along with the %SYSFUNC function. Creating the macro variable &N is important because we need to use it as the stopping point for the iterative %DO loop in the following step.

Next, the returned value is constructed piece by piece via an iterative %DO loop. For example, during the first iteration, %SCAN function extracts the first element from &NAMELIST, which is a, and `a = new_a` is constructed. Since `a = new_a` is no longer a macro statement, this part of the code is then placed in the input stack. After the macro call, all the constructed texts that are left in the input stack will be directed to the DATA step compiler and become part of RENAME statement or RENAME option.

Program 4:

```
%macro renames (pre, namelist);
   %local N i;
   %let N = %sysfunc(countw(&namelist));
   %do i = 1 %to &N;
        %scan(&namelist, &i) = &pre%scan(&namelist, &i)
   %end;
%mend renames;

data mydata;
    input a b c $ d e $;
datalines;
1 3 C 3 6
;
data mydata2;
    set mydata;
    rename %renames(new_, a b c);
run;

data mydata3;
    set mydata (rename=(%renames(new_, a b c)));
run;
```

NESTED FUNCTION CALLS

In Program 4, we specify the names of the variables that we need to rename in the %RENAMES macro function. Sometimes we might need to rename all the variables in a data set. Instead of listing them individually, we can write a macro function to return all the variable names.

The %ALL_NAMES macro function in Program 5 is used to return all the variable names from the input data. When we need to rename all the variables of a data set, we can call %ALL_NAMES function within the %RENAMES function like the example in Program 5.

Program 5:

```
%macro all_names(datname);
   %let dsid=%sysfunc(open(&datname, i));
   %do i=1 %to %sysfunc(attrn(&dsid, nvars));
       %sysfunc(varname(&dsid, &i))
   %end;
%mend all_names;

data mydata4;
    set mydata;
    rename %renames(new_, %all_names(mydata));
run;

data mydata5;
    set mydata (rename=(%renames(new_, %all_names(mydata))));
run;
```

There are many ways to list the names of all variables of a data set. For example, we can output the descriptor portion of a data set and store all the variable names in a macro variable. This process can be wrapped within a

macro. Using this approach needs to utilize a DATA step and a CONTENTS procedure within the macro. That is to say, this macro must contain non-macro language elements. This type of macro cannot be modified into a macro function because calling this macro in a DATA step, like in previous programs, will insert a DATA/PROC step within a DATA step, which will lead to a programming error. Therefore, when we create a macro function, we need to ensure that all the statements within the macro needs to be macro statements.

CONCLUSION

Writing a macro to imitate a function will make our program more convenient, flexible, or even more readable. To grasp this technique, we need to know how to transfer a value outside the macro. To avoid conflicting with global macro variables, the macro definition should not contain any global macro variables. Lastly, we also need to make sure that all statements in the macro must be macro statements.

REFERENCES

Carpenter, Art L., 2006, Carpenter's Complete Guide to the SAS® Macro Language, 3nd Edition, Cary, NC: SAS Institute Inc.
SAS Institute Inc. (2016). SAS® 9.4 Functions and CALL Routines, Fifth Edition. Cary, NC

CONTACT INFORMATION

Arthur X. Li
City of Hope National Medical Center
Division of Information Science
1500 East Duarte Road
Duarte, CA 91010 - 3000
Work Phone: (626) 256-4673 ext. 65121
Fax: (626) 471-7106
E-mail: arthurli@coh.org