CC016

A Few Macro Techniques to Shorten Your Programs

Yunchao (Susan) Tian, Social & Scientific Systems, Inc., Silver Spring, MD

ABSTRACT AND INTRODUCTION

There are many ways to write SAS programs to accomplish the same task. Macro language provides a great deal of efficiencies and flexibility even in simple situations. This brief paper presents a few macro techniques that can be used in many common situations to shorten SAS codes. Included topics are: 1. A simple macro that shortens the repetitive SAS statements that are declarative and array can't be used. 2. A macro shortcut for a statement that consists of many repetitive expressions. 3. How to use the %DO and %GOTO statements to write more efficient macros. 4. How to use the SYMPUT routine and the %EVAL function to convert a numbered series of names with a common prefix to an alphabetically ordered series of names with the same prefix. Each of the topics will be described along with examples.

A MACRO TO GENERATE A LIST OF LABEL STATEMENTS

Sometimes you may need to label a numbered series of variables with a common prefix. When working with healthcare data, we often need to convert data from multiple records per person to single record per person, e. g. convert condition-level data to person-level data. Each person may have as many as 20 conditions and therefore need 20 condition IDs to identify each condition. We need to label each of the condition IDs. We can't use array and DO loop to do this since the LABEL statement is a declarative statement. But we can use %DO loop in a macro like the following to accomplish this:

```
%MACRO LABEL(N);
  %DO I = 1 %TO &N;
    LABEL CONDID&I="condition ID&I";
  %END;
%MEND LABEL;
```

The call

```
%LABEL(20)
```

generates these statements:

```
LABEL CONDID1="condition ID1";
LABEL CONDID2="condition ID2";
……
LABEL CONDID20="condition ID20";
```

A MACRO TO GENERATE REPETITIVE EXPRESSIONS

Quite often we need to create a flag variable from a series of variables. For example, we want to create a flag variable based on the values of all the diagnosis codes of a person. It is tedious to list all the diagnosis codes in a statement, especially when a program contains a number of such statements. The following macro provides an efficient shortcut for doing this:

```
%MACRO MDX(NDX, CODE);
  %DO I = 1 %TO &NDX-1;
    DX&I = "&CODE" OR
  %END;
  DX&NDX = "&CODE"
%MEND MDX;
```

Invoking the macro MDX in the IF-THEN statement:

```
IF %MDX(15, E8710) THEN ERROR1 = 1;
```

produces the following complete IF-THEN statement:

```
IF DX1 = "E8710" OR DX2 = "E8710" OR DX3
= "E8710" OR DX4 = "E8710" OR DX5 =
"E8710" OR DX6 = "E8710" OR DX7 =
"E8710" OR DX8 = "E8710" OR DX9 =
"E8710" OR DX10 = "E8710" OR DX11 =
"E8710" OR DX12 = "E8710" OR DX13 =
"E8710" OR DX14 = "E8710" OR DX15 =
"E8710" THEN ERROR1 = 1;
```

Different data may have different number of diagnosis codes and we usually need to create several flag variables based on different values of these diagnosis codes in one program. So the advantage of the macro MDX is obvious. Depending on your need and how general you want your macro to be, you can choose to use no parameter, one parameter, or two parameters.

A MACRO WITH THE %GOTO STATEMENT AND STATEMENT LABEL

We can write a macro and invoke the macro with different parameter values. But if the values are a numbered series of names, it will be more efficient to use the iterative %DO loop. If some of the members in the series are not available, you can use the %GOTO statement and statement label to cause execution to jump to the next member, as shown below:

```
%MACRO READ;
  %DO I = 1 %TO 4;
  %IF &I = 3 %THEN %GOTO OUT;

  DATA CTY_&I;
   INFILE "D:\NESUG\CTY_&I";
   INPUT  ID  F96  F97  F98;
  RUN;

  %OUT: %END;
%MEND READ;
```

The call

```
%READ
```

generates these statements:

```
DATA CTY_1;
 INFILE "D:\NESUG\CTY_1";
 INPUT  ID  F96  F97  F98;
RUN;

DATA CTY_2;
 INFILE "D:\NESUG\CTY_2";
 INPUT  ID  F96  F97  F98;
RUN;

DATA CTY_4;
 INFILE "D:\NESUG\CTY_4";
 INPUT  ID  F96  F97  F98;
RUN;
```

This technique is useful when reading state health statistics data that consist of data from all counties with the same layout. By using the %GOTO statement and statement label, we can start to process the data before we receive data from all counties.

THE SYMPUT ROUTINE WITH THE %EVAL AND BYTE FUNCTIONS

For some reason, we want all the SAS data sets created in the previous section to be saved in an alphabetically ordered series of names instead of numbered series of names. In another word, the SAS data set created from the ASCII file CTY_1 will be called CTY_A, and the data set from CTY_2 will be called CTY_B. You can do this by using the following SYMPUT routine:

```
CALL SYMPUT('N', BYTE(%EVAL(&I+64)));
```

The %EVAL function returns an integer value which is the sum of 64 and the value of the index variable I. The BYTE function returns a value of a letter. For example, it returns a value of A if the value of I is 1 since A is the 65th character on the ASCII system. Then the SYMPUT routine assigns the value produced by the BYTE function to the macro variable N. If we apply the above statement to the data step in the previous section and retrieve the macro variable N's value in another data step, we can create a series of SAS data sets with alphabetically ordered names. So invoking the following macro will create 3 permanent SAS data sets named CTY_A, CTY_B, and CTY_D.

```
%MACRO READ;
  %DO I = 1 %TO 4;
  %IF &I = 3 %THEN %GOTO OUT;

  DATA CTY_&I;
   INFILE "D:\NESUG\CTY_&I";
   INPUT  ID  F96  F97  F98;
   CALL SYMPUT('N', BYTE(%EVAL(&I+64)));
  RUN;

  DATA OUT.CTY_&N;
   SET CTY_&I;
  RUN;

  %OUT: %END;
%MEND READ;
```

CONCLUSIONS

Macros are very useful tools for repeating tasks. The above examples illustrate how macros can reduce the amount of text you must enter, and therefore save time and increase productivity. Even simple macro facility features will help you extend your SAS programming. You don't have to be a macro expert to use macros. I hope that the simple macros presented in this paper would inspire the beginner macro users to use the macro language to take control of circumstances that may seem a bit unruly at first.

ACKNOWLEDGMENTS

I would like to thank our project director Lita Manuel and my supervisor Devi Katikineni for their continual support and encouragement in my career growth. I would also like to thank all my clients in AHRQ for the wonderful time working together. Special thanks go to my two long-term clients Barbara Schone and Chunliu Zhan. I want to thank my colleague Raymond Hu for sharing with me his tips and tricks and many helpful discussions as I learn how to be a good SAS programmer. Finally, I thank my colleague Paul Gorrell for providing me all NESUG information as I prepare this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Yunchao (Susan) Tian
Social & Scientific Systems, Inc.
8757 Georgia Avenue, 12th Floor
Silver Spring, MD 20910
Work Phone: (301) 628-3285
Fax: (301) 628-3201
Email: stian@s-3.com

****