Paper 028-2010

When Best to Use the %LET Statement, the SYMPUT Routine, or the INTO Clause to Create Macro Variables

Arthur X. Li, City of Hope Comprehensive Cancer Center, Duarte, CA

ABSTRACT

Macro variables are the most essential part of the SAS® macro facility. One can create macro variables by using the %LET statement, the SYMPUT routine, or the INTO clause in the SELECT statement from the SQL procedure. Sometimes a SAS programmer is often unsure when best to use which method due to a lack in understanding each step of macro language processing. This lack in understanding includes how SAS statements are transferred from the input stack to the macro processor and DATA step compiler, what role the macro processor plays during this process, and when best to utilize the interface to interact with the macro facility during the DATA or SQL execution. Once one grasps the intricacies of macro language processing, one will best know how to accurately create a macro variable.

INTRODUCTION

One of the most important components of Base SAS is the SAS macro facility, which includes macro variables and macro programs. One can use SAS macro variables and macro programming statements in a SAS macro program to generate SAS codes. In order to utilize the SAS macro facility, one needs to learn SAS macro language. Although the convention of the macro language is similar to SAS language, macro language is indeed a separate language that is used in the SAS macro program. The prerequisites of learning macro programs begins with macro variables. In this paper, we will only focus on macro variables.

To truly understand macro variables, one must grasp the mechanisms of macro processing. One excellent source for an introduction to macro processing is from SAS® Macro Programming Made Easy by Michele M. Burlew. In this book, Burlew cleverly uses figures to illustrate the mechanics of macro processing. The figures in this paper are adapted from the figures in her book.

CREATING AND REFERENCING MACRO VARIABLES

Macro variables are either automatic, which is provided by SAS, or user-defined, which is created by SAS users. We will only cover the user-defined macro variables in this paper.

CREATING USER-DEFINED VARIABLES USING THE %LET STATEMENT

One way to create a macro variable is to use the %LET statement, which has the following form.

%LET *macro-variable* = *value*;

The *value* is stored as character strings that can be ranged from 0 to 65,534 characters. Mathematical expressions that are stored as *value* are not evaluated. The case of the *value* is also preserved. If the *value* contains quotation marks that include literals, the quotation marks will be part of the *value*. Before assignment is made, any of the leading and trailing blanks will be removed from the *value*.

If *macro- variable* and/or *value* contain references to another macro variable, the reference will be evaluated first before the assignment. Also, if the *macro-variable* has already been defined previously in the program, *value* will replace the most current value. Here are some examples of defining macro variables by using the %LET statement.

```
%let var1 = 4 + 3;
%let var2 = hello;
%let var3 =    leading blank;
%let var4 = " quotations ";
```

The values of the macro variables just defined are summarized in the following table:

Macro variable name	Value	Notes
var1	4 + 3	Mathematical expressions are not evaluated
var2	hello	
var3	leading blank	Leading blanks are removed
var4	“ quotations “	Quotation marks are part of the values

REFERENCING MACRO VARIABLES

Once a macro variable is defined, the value of the macro variable is stored in the global symbol table[1]. In order to substitute a macro variable in the SAS program, you must reference the macro variable by preceding the macro variable name with an ampersand (&). The reference causes the macro processor to search the macro variable in the symbol table. Once the macro variable is found, the value that corresponds to the macro variable will be substituted into the SAS program. If the reference of a macro variable is within quotations, the double quotation marks must be used.

Consider the following dataset, *Height.sas7bdat*, which contains information for SEX and HEIGHT of each student. Suppose we would like to create an indicator variable, TALL, which is defined as either 1 for students who are taller than 63 inches or otherwise 0. The number 63 could possibly be used multiple times in the program. Furthermore, the definition of TALL might change in the future; for instance, instead of using 63 inches, we might use 65. Thus, we can define a macro variable that contains the threshold value and reference this macro variable in the program. Since we use a macro variable to store this threshold value, if we are going to make changes for this threshold, we will only need to change it once in the macro variable definition. Program 1 below contains the partial code of the entire program:

Height.sas7bdat

	name	sex	height
1	John	m	65
2	Tom	m	60
3	Mary	f	62
4	Helen	f	64

Program 1:
```
%let ht = 63;
data ex1;
    set height;
    tall = height > &ht;
run;
```

In Program 1 above, to reference the macro variable HT, an ampersand (&) is placed in front of the macro variable HT. After the substitution, the code will be as follows:

```
data ex1;
    set height;
    tall = height > 63;
run;
```

SAS AND MACRO PROCESSING

SAS PROCESSING

In order to understand how macro variables are processed and stored, one needs to understand how SAS processing works. Once a sequence of SAS code is submitted, it is processed in two-phase sequences: the compilation and execution phases. The compilation phase is performed by the **compiler**.

Before the SAS code is transferred to the compiler, the codes are placed in a memory area, which is called the **input stack**. Next, the **word scanner** takes the SAS code from the input stack and breaks that code into words or symbols, which are called **tokens**, and directs the tokens to the correct destination, such as the compiler or the macro processor. When the compiler receives the semicolon following the RUN statement, it stops accepting tokens from the word scanner. The compiler then compiles the received tokens, checking for syntax errors. If there are no syntax errors, the execution phase begins (see Figures 1a-c for illustrations). The types of tokens that the compiler recognizes are illustrated in following table:

[1] In addition to the global symbol table, there are also local symbol tables. The local symbol table is used within the macro definition. Since we are not covering macro programs, we will only use global symbol table in this paper. For simplicity, the symbol table in this paper will be referred to as the global symbol table.

Types of Token	Contains…	Examples
Literal	Characters enclosed in quotation marks	“John” ‘John’
Number	Numerals including decimals, E-notation, date, time, and datetime constants, and hexadecimal constants	555 ‘01mar2010’d 30e4 2.5
Name	Characters that begin with a letter or underscore and that continues with underscores, letters, or numbers. A period can sometimes be part of a name	_n_ means dollar9.2 descending
Special character	Characters other than a letter, number, or underscore that have a special meaning to the SAS system	* / + % & . ;

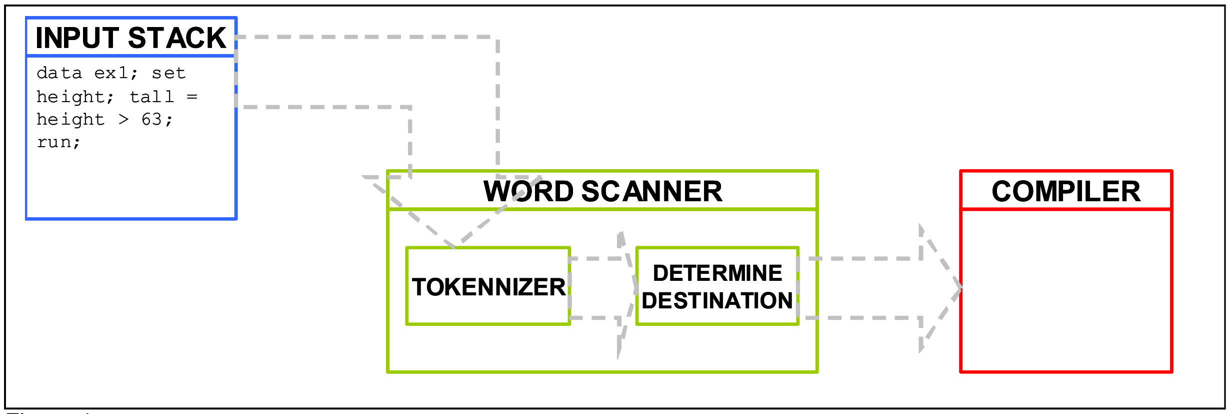

Figure 1a. The input stack receives submitted SAS code.

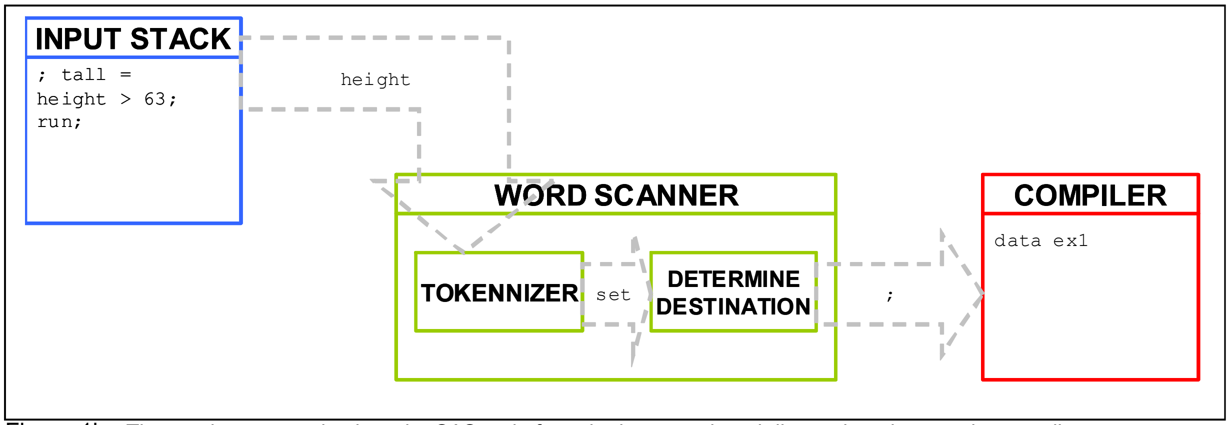

Figure 1b. The word scanner tokenizes the SAS code from the input stack and directs the tokens to the compiler.

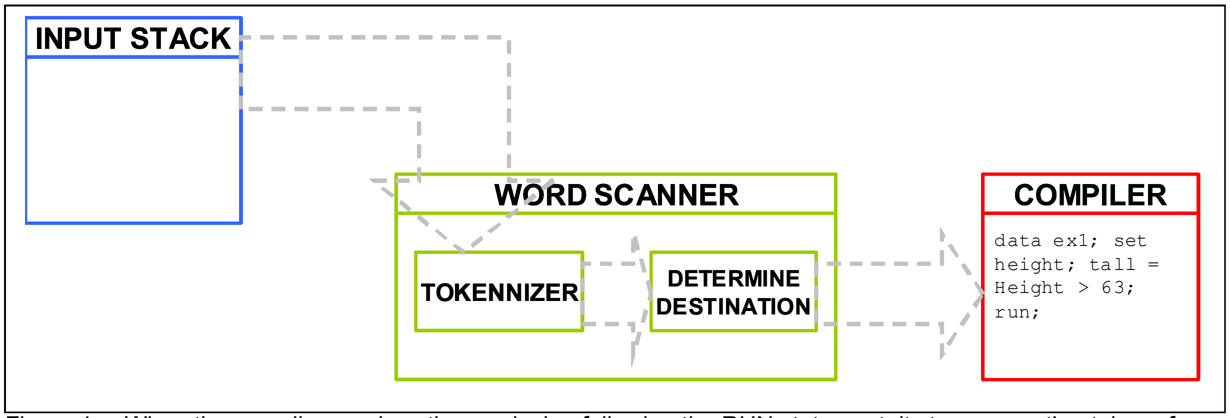

Figure 1c. When the compiler receives the semicolon following the RUN statement, it stops accepting tokens from the word scanner. The compilation begins.

MACRO PROCESSING

The macro processor is responsible for processing all the macro languages, such as the %LET statement and the macro variable references. So that the macro processor can process the macro language, the word scanner has to be able to recognize the macro language and direct the macro language to the macro processor. The tokens that prompt the word scanner that the subsequent codes are macro languages are called **macro triggers**, such as %LET followed by a name token and & followed by a name token. Once the macro triggers are detected by the word scanner, the word scanner passes the tokens to the macro processor. Then the macro processor performs its actions.

For macro variables, the macro processors will either create a new macro variable or modify an existing macro variable in a symbol table. The macro processor also retrieves the value from the existing macro variable and returns it to the input stack where it originally contains the macro reference.

We can use Program 1 as an example to illustrate how macro processing works. First, Program 1 is pushed into the input stack (Figure 2a). The word scanner recognizes that %LET followed by HT (a name token) is a macro trigger; it directs the %LET statement to the macro processor (Figure 2b). The macro processor will keep requesting tokens until it reaches the semicolon. The macro processor creates macro variable HT and assigns the value 63 in the symbol table (Figure 2c). After the macro processor receives the semicolon, the word scanner begins to transfer the subsequent statements to the compiler until it reaches the next macro trigger, which is ampersand (&) followed by a name token, HT. Then word scanner directs &HT to the macro processor (Figure 2d). The macro processor retrieves the value that corresponds to HT in the symbol table, which is 63, and returns it to the input stack (Figure 2e). The word scanner continues scanning. Since there are no more macro triggers, the remaining tokens are passed-on to the compiler. When the compiler receives the semicolon following the RUN statement, it stops accepting tokens from the word scanner. The compilation phase begins.

Remember that the word scanner does not recognize macro triggers that are enclosed in single quotation marks. Thus, you need to use double quotations if you want to include a macro reference inside the quotation.

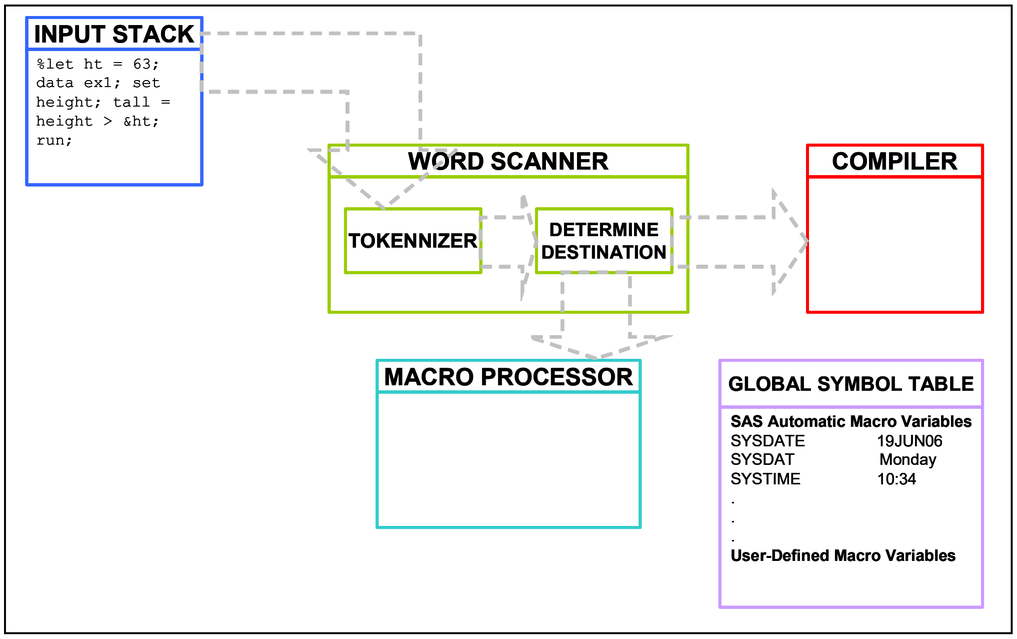

Figure 2a. Program 1 is pushed into the input stack.

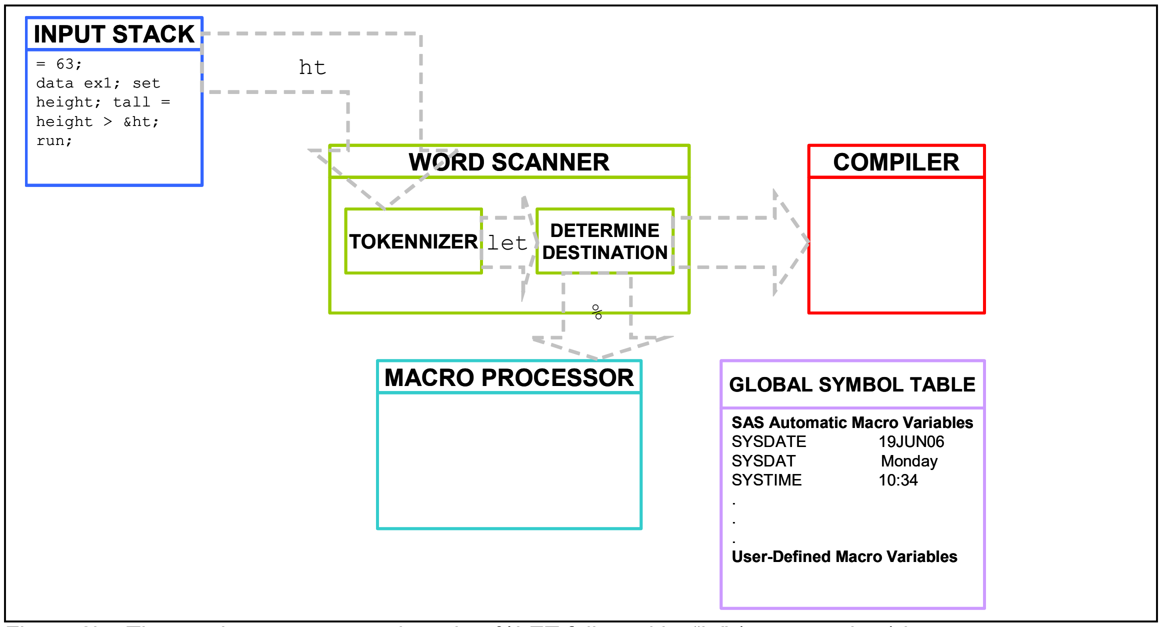

Figure 2b. The word scanner recognizes that %LET followed by “ht” (a name token) is a macro trigger, directing %LET to the macro processor. The macro processor requests additional tokens until it receives a semicolon.

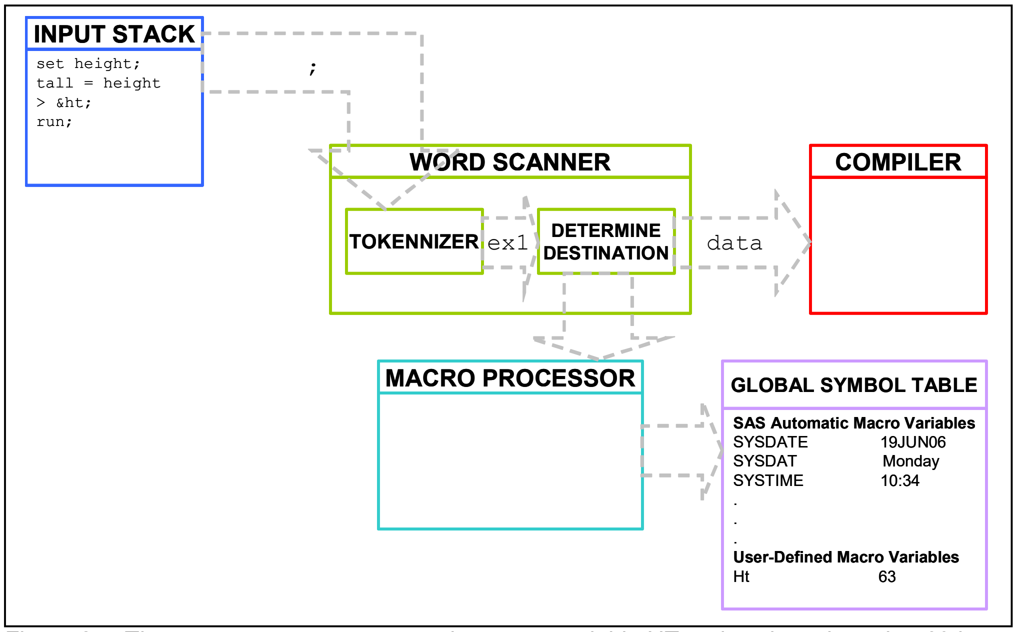

Figure 2c. The macro processor creates the macro variable HT and assigns the value 63 in the global symbol table.

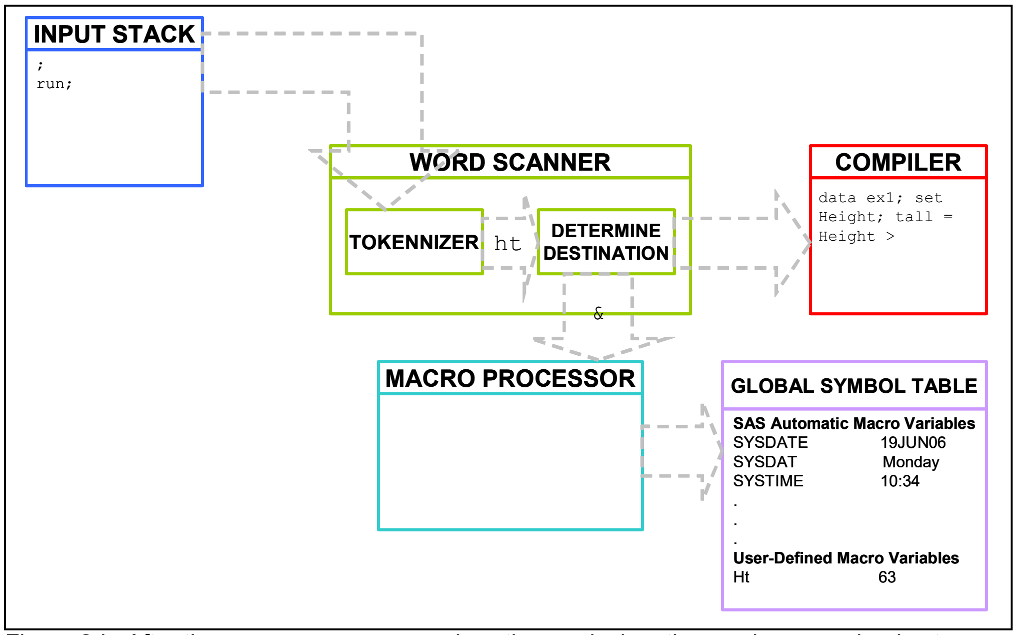

Figure 2d. After the macro processor receives the semicolon, the word scanner begins to transfer the subsequent statements to the compiler until it reaches the next macro trigger, which is ampersand (&) followed by a name token (HT). The word scanner directs &HT to the macro processor.

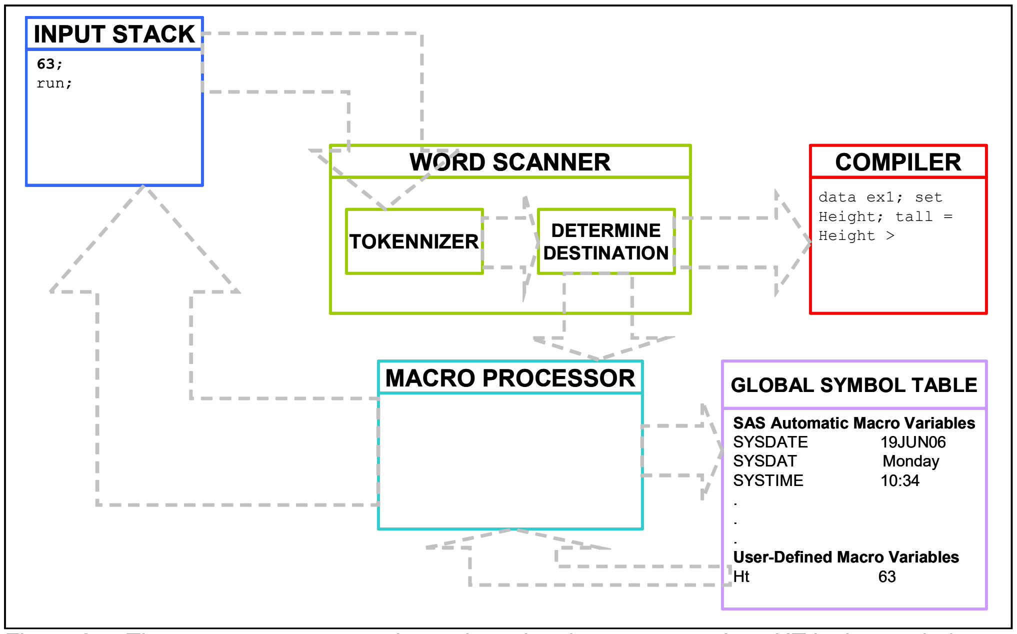

Figure 2e. The macro processor retrieves the value that corresponds to HT in the symbol table, which is 63, returning it to the input stack.

PROCESSING MACRO VARIABLES DURING DATA STEP EXECUTION

In some situations, creating a macro variable during the DATA step execution is necessary. For example, you might want to create a macro variable based on the data value in the SAS dataset, such as a computed value or based on programming logic. Using the %LET statement will not work in this situation since the macro variable created by the %LET statement occurs before the execution starts.

Suppose you would like to create macro variable TITL, which served as a title for printing the dataset *ex1* (that was created in Program 1). If there are some students taller than 63 inches, the TITL macro variable will store the value of “Some students are taller than 63 inches,” and if none of the students are talker than 63 inches, TITL will store “None of the students are taller than 63 inches.” Program 2 uses the %LET statement to create the macro variable.

Program 2:
```
data _null_;
    set ex1 end=last;
    if tall then count_tall +1;
    if last then do;
        if count_tall then do;
            %let titl = "Some students are taller than 63 inches";
        end;
        else do;
            %let titl = "None of the students are taller than 63 inches";
        end;
    end;
run;

proc print data=ex1;
    title &titl;
run;
```

Output:

```
None of the students are taller than 63 inches

Obs    name     sex    height    tall

 1     John      m       65        1
 2     Tom       m       60        0
 3     Mary      f       62        0
 4     Helen     f       64        1
```

Notice that Program 2 did not create the macro variable correctly. There are two people that are taller than 63 inches in the dataset *ex1*, but the title in the output above shows "None of the students are taller than 63 inches."

Let's exam this program in more detail. At first, Program 2 is pushed into the input stack (Figure 3a). The word scanner directs the SAS code to the compiler until it reaches the macro trigger (%LET). The word scanner directs the %LET statement to the macro processor (Figure 3b). The macro processor creates macro variable TITL with the value of "Some students are taller than 63 inches" in the symbol table (Figure 3c). After directing the %LET statement to the macro processor, the word scanner directs subsequent tokens to the compiler until it reads the second %LET statement. The word scanner directs the second %LET statement to the macro processor (Figure 3d). The macro processor reassigns "None of the students are taller than 63 inches" to the macro variable TITL (Figure 3e). The word scanner continues to send the remaining tokens to the compiler. When the compiler receives the semicolon following the RUN statement, it stops accepting tokens from the word scanner. When the compilation begins, there will be no more %LET statements in the DATA step. Here is what the program will look like when it is processed by the compiler:

```
data _null_;
    set ex1 end=last;
    if tall then count_tall +1;
    if last then do;
        if count_tall then do;
        end;
        else do;
        end;
    end;
run;
```

THE SYMPUT ROUTINE

To fix the problem in Program 2, we need to be able to create a macro variable during the DATA step execution, which can be accomplished by using the SYMPUT routine. Since the *macro-variable* is assigned with the *value* during the DATA step execution, you can only reference the *macro-variable* after the DATA step in which it is created. The SYMPUT routine has the following form:

CALL SYMPUT (*Macro-variable, value*);

In the SYMPUT routine, both *macro-variable* and *value* can be specified as literal (text in quotations), a DATA step variable, or a DATA step expression. We will examine each of these cases.

THE SYMPUT ROUTINE: BOTH ARGUMENTS ARE LITERALS

CALL SYMPUT ('*Macro-variable*', '*value*');

When both *macro-variable* and *value* are literal, they are enclosed in quotation marks. The text enclosed in the quotation marks for the first argument is the exact macro variable name. The second argument enclosed in the quotation marks is the exact value that is assigned to the macro variable. In Program 2, the macro variable we attempted to create is based on a calculated value in the DATA step. This is a perfect situation to utilize the SYMPUT routine. Program 3 is a correct modification of Program 2.

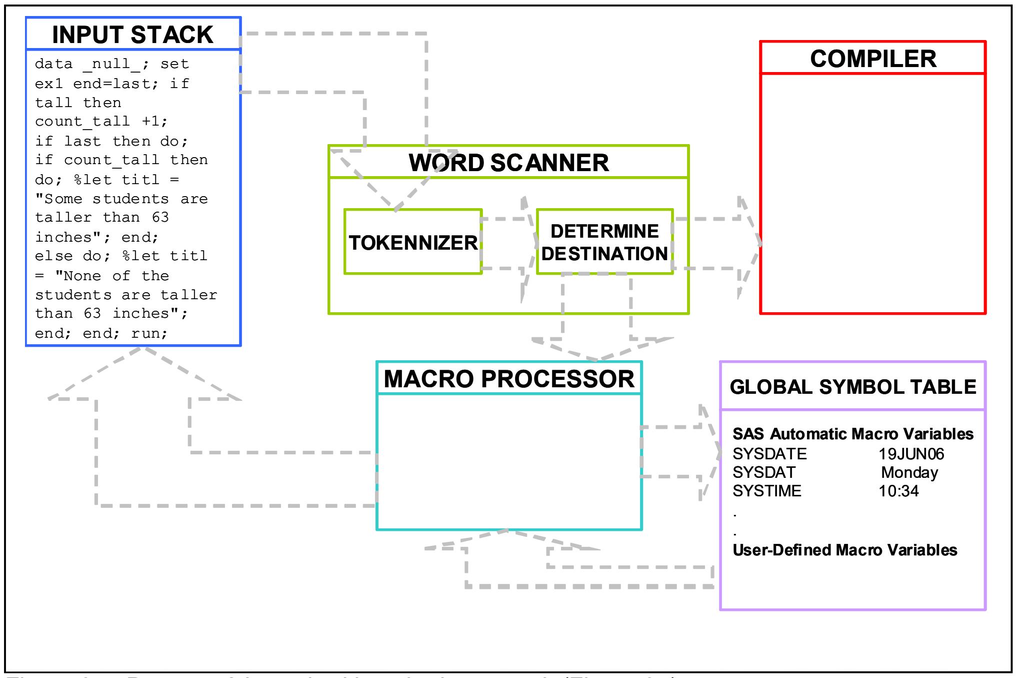
INPUT STACK
data _null_; set ex1 end=last; if tall then count_tall +1; if last then do; if count_tall then do; %let titl = "Some students are taller than 63 inches"; end; else do; %let titl = "None of the students are taller than 63 inches"; end; end; run;
COMPILER
WORD SCANNER
TOKENNIZER
DETERMINE DESTINATION
MACRO PROCESSOR
GLOBAL SYMBOL TABLE
SAS Automatic Macro Variables
SYSDATE 19JUN06
SYSDAT Monday
SYSTIME 10:34
User-Defined Macro Variables

Figure 3a. Program 2 is pushed into the input stack (Figure 3a).

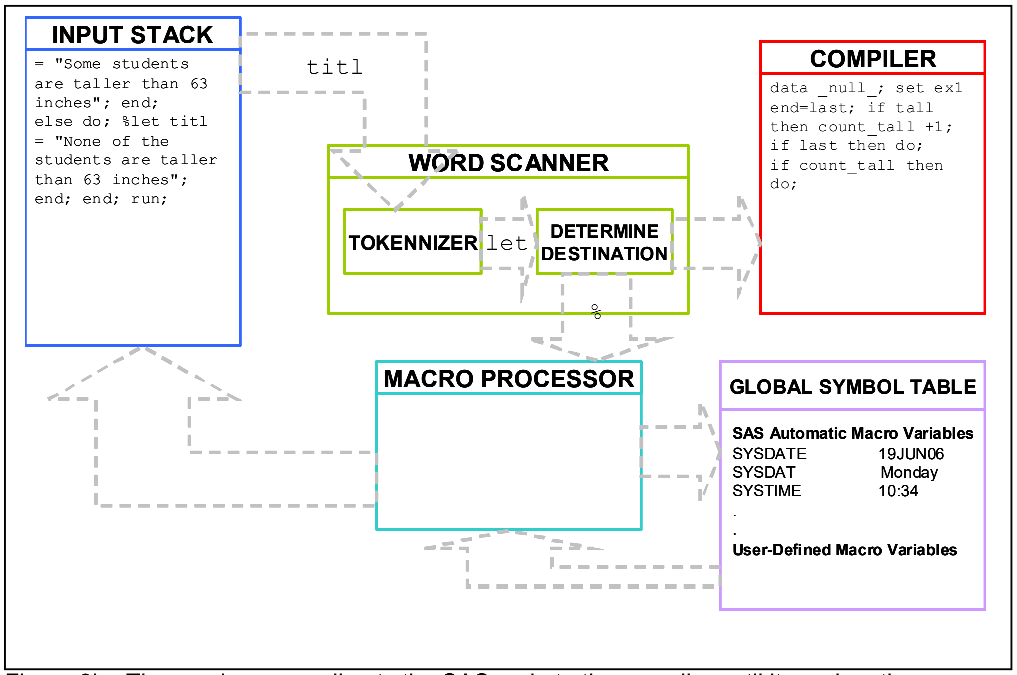
INPUT STACK
= "Some students are taller than 63 inches"; end; else do; %let titl = "None of the students are taller than 63 inches"; end; end; run;
titl
COMPILER
data _null_; set ex1 end=last; if tall then count_tall +1; if last then do; if count_tall then do;
WORD SCANNER
TOKENNIZER
let
DETERMINE DESTINATION
%
MACRO PROCESSOR
GLOBAL SYMBOL TABLE
SAS Automatic Macro Variables
SYSDATE 19JUN06
SYSDAT Monday
SYSTIME 10:34
User-Defined Macro Variables

Figure 3b. The word scanner directs the SAS code to the compiler until it reaches the macro trigger (%LET). The word scanner direct the %LET statement to the macro processor.

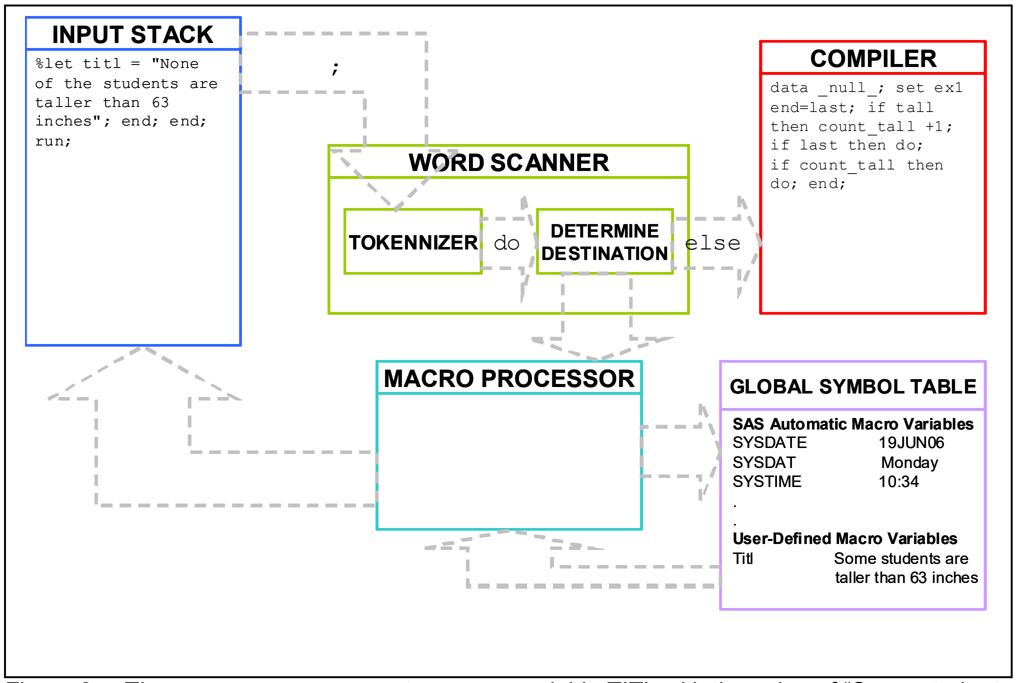
INPUT STACK
%let titl = "None of the students are taller than 63 inches"; end; end; run;
;
COMPILER
data _null_; set ex1 end=last; if tall then count_tall +1; if last then do; if count_tall then do; end;
WORD SCANNER
TOKENNIZER
do
DETERMINE DESTINATION
else
MACRO PROCESSOR
GLOBAL SYMBOL TABLE
SAS Automatic Macro Variables
SYSDATE 19JUN06
SYSDAT Monday
SYSTIME 10:34
User-Defined Macro Variables
Titl Some students are taller than 63 inches

Figure 3c. The macro processor creates macro variable TITL with the value of “Some students are taller than 63 inches” in the symbol table.

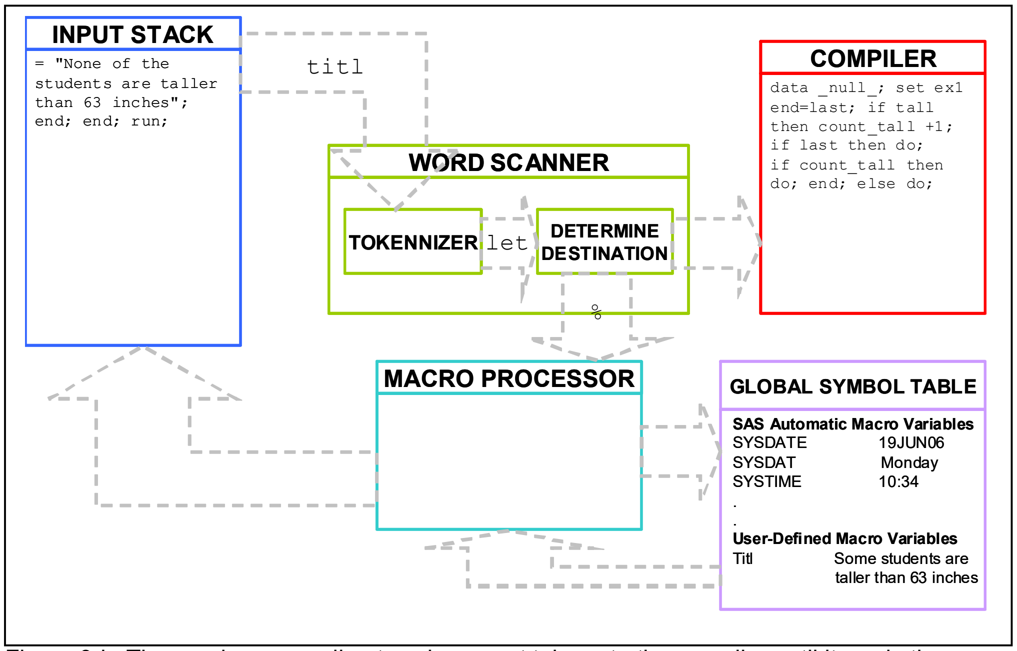
INPUT STACK
= "None of the students are taller than 63 inches"; end; end; run;
titl
COMPILER
data _null_; set ex1 end=last; if tall then count_tall +1; if last then do; if count_tall then do; end; else do;
WORD SCANNER
TOKENNIZER
let
DETERMINE DESTINATION
%
MACRO PROCESSOR
GLOBAL SYMBOL TABLE
SAS Automatic Macro Variables
SYSDATE 19JUN06
SYSDAT Monday
SYSTIME 10:34
User-Defined Macro Variables
Titl Some students are taller than 63 inches

Figure 3d. The word scanner directs subsequent tokens to the compiler until it reads the second %LET statement. The word scanner directs the %LET statement to the macro processor.

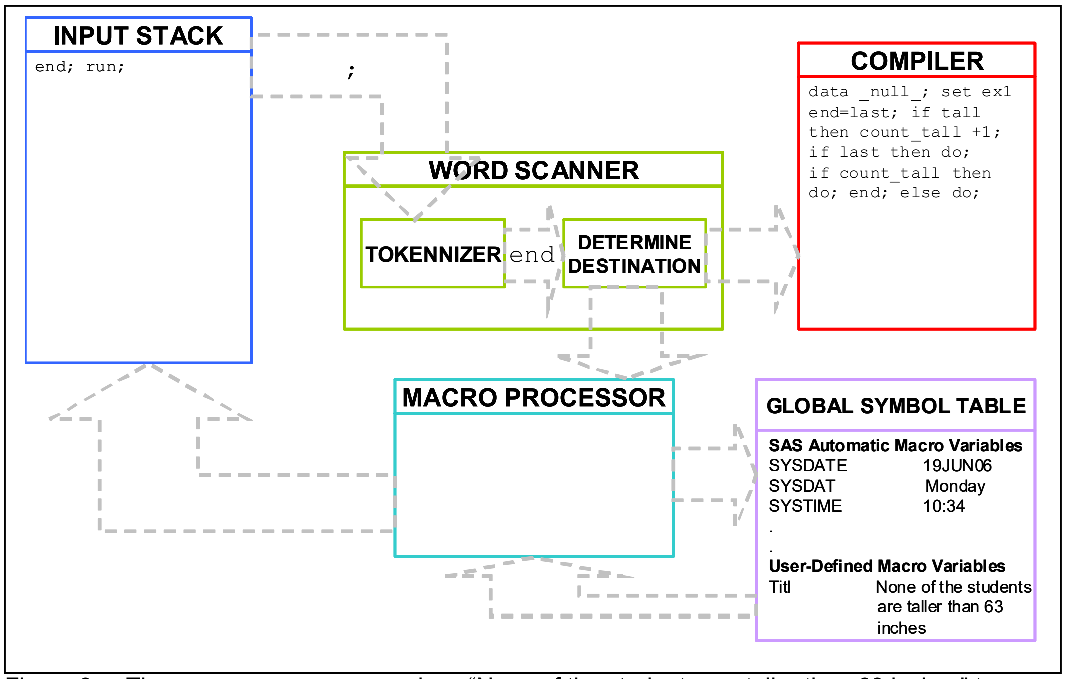
INPUT STACK
end; run;
;
WORD SCANNER
TOKENNIZER
end
DETERMINE DESTINATION
COMPILER
data _null_; set ex1 end=last; if tall then count_tall +1; if last then do; if count_tall then do; end; else do;
MACRO PROCESSOR
GLOBAL SYMBOL TABLE
SAS Automatic Macro Variables
SYSDATE 19JUN06
SYSDAT Monday
SYSTIME 10:34
User-Defined Macro Variables
Titl None of the students are taller than 63 inches

Figure 3e. The macro processor reassigns "None of the students are taller than 63 inches" to macro variable TITL.

Program 3:

```
data _null_;
    set ex1 end=last;
    if tall then count_tall +1;
    if last then do;
        if count_tall then do;
            call symput('titl', 'Some students are taller than 63 inches');
        end;
        else do;
            call symput('titl', 'None of the students are taller than 63 inches');
        end;
    end;
run;

proc print data=ex1;
    title &titl;
run;
```

Output:

```
Some students are taller than 63 inches

Obs    name     sex    height    tall

 1     John      m       65        1
 2     Tom       m       60        0
 3     Mary      f       62        0
 4     Helen     f       64        1
```

In Program 3, both arguments in the SYMPUT routine are closed in quotation marks. The first argument in the SYMPUT routine, TITL, is the name of the macro variable. The second argument (*Some students are taller than 63 inches* or *None of the students are taller than 63 inches*) is the value that is assigned to TITL. Since variable COUNT_TALL is greater than 0, macro variable TITL is assigned with the value "Some students are taller than 63 inches" through the SYMPUT routine. Notice that the output has the correct title this time.

THE SYMPUT ROUTINE: THE SECOND ARGUMENT IS A DATA STEP VARIABLE

When the second argument in the SYMPUT routine is not in quotation marks, you are assigning the value of a DATA step variable to the macro variable. Any leading or trailing blanks that are part of the values of a DATA step variable will be part of the macro variables. If the DATA step variable is a numeric variable, the values will be converted to the character variables automatically by using the BEST12. format.

Suppose that you would like to create four macro variables, JOHN_HT, TOM_HT, MARY_HT, and HELEN_HT, that contain the heights for John, Tom, Mary, and Helen, respectively. To create a macro variable that contains the values of the HEIGHT variable, you need to use HEIGHT as the second argument of the SYMPUT routine without enclosing HEIGHT in quotation marks. You can use the %PUT statement[2] to verify the values of the macro variables.

Program 4:
```
data _null_;
    set height;
    if name = 'John' then call symput ('John_ht', height);
    else if name = 'Tom' then call symput ('Tom_ht', height);
    else if name = 'Mary' then call symput ('Mary_ht', height);
    else if name = 'Helen' then call symput ('Helen_ht', height);
run;

%put macro variable John_ht: &John_ht;
%put macro variable Tom_ht: &Tom_ht;
%put macro variable Mary_ht: &Mary_ht;
%put macro variable Helen_ht: &Helen_ht;
```

SAS Log:
```
359  %put macro variable John_ht: &John_ht;
macro variable John_ht:           65
360  %put macro variable Tom_ht: &Tom_ht;
macro variable Tom_ht:           60
361  %put macro variable Mary_ht: &Mary_ht;
macro variable Mary_ht:           62
362  %put macro variable Helen_ht: &Helen_ht;
macro variable Helen_ht:           64
```

THE SYMPUT ROUTINE: THE FIRST ARGUMENT IS A DATA STEP VARIABLE

When the first argument in the SYMPUT routine is not in quotation marks, you are creating multiple macro variables by using only one SYMPUT routine. The names of the macro variables that you are creating are the values of a DATA step variable. Suppose that you would like to create four macro variables by using the value of the NAME variable in the HEIGHT data set as the macro variable names, namely 'John', 'Tom', 'Mary', and 'Helen'. These four macro variables contain the corresponding heights for these four people. Notice that both arguments of the SYMPUT routine are not enclosed in quotation marks in the following code:

Program 5:
```
data _null_;
    set height;
    call symput (name, height);
run;

%put macro variable John: &John;
%put macro variable Tom: &Tom;
```

[2] Following is the general form of the %PUT statement: ***%PUT text*** , where *text* is any text string. The %PUT statement writes only to the SAS log. If the text is not specified, the %PUT statement writes a blank line. The %PUT statement resolves macro triggers in *text* before *text* is written. For more information, check SAS documentations.

```
%put macro variable Mary: &Mary;
%put macro variable Helen: &Helen;
```

SAS Log:

```
368  %put macro variable John: &John;
macro variable John:           65
369  %put macro variable Tom: &Tom;
macro variable Tom:           60
370  %put macro variable Mary: &Mary;
macro variable Mary:           62
371  %put macro variable Helen: &Helen;
macro variable Helen:           64
```

THE SYMPUT ROUTINE: EITHER ONE OR BOTH ARGUMENTS ARE A DATA STEP EXPRESSION

You can use a DATA step expression in either or both arguments in the SYMPUT routine. Program 5 creates four macro variables by using the values of the DATA step variable NAME. You can add a prefix to the macro variable to make it more meaningful, such as 'height_John', 'height_Tom', etc. In this situation, you can use the concatenating operator "||" to combine "height_" with NAME (the DATA step variable).

Remember that when you use a DATA step numeric variable as the second argument without enclosing it in quotation marks, the numeric values will be converted to character values by using the BEST12. format. The blanks are stored as part of the macro variable. You can use the DATA step function LEFT to remove the leading blanks and TRIM to remove the trailing blanks. Program 6 below uses DATA step expressions in both arguments.

Program 6:

```
data _null_;
    set height;
    call symput ("height_" ||name, trim(left(height)));
run;

%put macro variable height_John: &height_John;
%put macro variable height_Tom: &height_Tom;
%put macro variable height_Mary: &height_Mary;
%put macro variable height_Helen: &height_Helen;
```

SAS Log:

```
411  %put macro variable height_John: &height_John;
macro variable height_John: 65
412  %put macro variable height_Tom: &height_Tom;
macro variable height_Tom: 60
413  %put macro variable height_Mary: &height_Mary;
macro variable height_Mary: 62
414  %put macro variable height_Helen: &height_Helen;
macro variable height_Helen: 64
```

THE SYMPUTX ROUTINE

The SYMPUTX routine is an improved version of the SYMPUT routine and becomes available beginning with SAS®9. Besides creating macro variables like the SYMPUT routine, the SYMPUTX routine can remove leading and trailing blanks from both arguments. Program 6 can be re-written by using the SYMPUTX routine without using the LEFT and TRIM functions.

Program 7:

```
data _null_;
    set height;
    call symputx ("height_" ||name, height);
run;
```

```
%put macro variable height_John: &height_John;
%put macro variable height_Tom: &height_Tom;
%put macro variable height_Mary: &height_Mary;
%put macro variable height_Helen: &height_Helen;
```

SAS Log:

```
429  %put macro variable height_John: &height_John;
macro variable height_John: 65
430  %put macro variable height_Tom: &height_Tom;
macro variable height_Tom: 60
431  %put macro variable height_Mary: &height_Mary;
macro variable height_Mary: 62
432  %put macro variable height_Helen: &height_Helen;
macro variable height_Helen: 64
```

There are also other differences between the SYMPUT routine and the SYMPUTX routine. When the second argument is numeric, the SYMPUT routine converts it to character by using the BEST12.format and writing a note in the log; on the other hand, the SYMPUTX routine will convert it to character by using the BEST32. format without writing a note in the long. Furthermore, the SYMPUTX routine has an optional third argument that enables you to specify the symbol table in which to store the macro variables, but the SYMPUT routine does not. Further details can be found in SAS® documentations.

PROCESSING MACRO VARIABLES DURING THE EXECUTION OF PROC SQL

You can also create and update macro variables during the execution of PROC SQL. The INTO clause in the SELECT statement can create one or more macro variables that performs a similar role to the SYMPUT routine in the DATA step. The INTO clause can assign a calculated result or the value of a data variable to a macro variable. The INTO clause can only be used in the outer query of a SELECT statement and cannot be used when you are creating a table or a view. Here is the general syntax for PROC SQL and the INTO clause:

```
PROC SQL <PRINT|NOPRINT>;
  SELECT column1 <, column2, …>
  INTO :macro-variable1 <, :macro-variable2, …>
   FROM table |view
   <other clauses>;
 QUIT;
```

To select one or more columns from SQL *table* or *view*, you can specify *column1*, *column2*, … after the keyword SELECT. *Macro-variable1*, *macro-variable2* are the names of the macro variables that you are creating after the keyword INTO. You must write colon(s) (:) before each of the macro variables. The INTO clause does not trim the leading and trailing blanks of the macro variables. You can use other SQL clauses to group or subset the data set. To suppress displays output from PROC SQL, you can use the NOPRINT option (with PRINT being the default setting).

CREATING MACRO VARIABLE(S) BASED ON CALCULATED RESULTS

You can easily assign a calculated summary statistic to a macro variable by using the PROC SQL. For example, suppose you would like to create a macro variable MEAN_HT that contains the average heights of students. You can write the following program:

Program 8:

```
proc sql noprint;
    select mean(height) into: mean_ht
    from height;
quit;

%put macro variable mean_ht: &mean_ht;
```

SAS Log:

```
523  %put macro variable mean_ht: &mean_ht;
macro variable mean_ht:    62.75
```

You can create the macro variable MEAN_HT without using the PROC SQL, but you must use multiple steps to accomplish this task. First, you need to calculate the mean heights by using the MEAN procedure and output the mean height to a dataset. Then you can use CALL SYMPUT or CALL SYMPUTX to create the macro variable.

Program 9:

```
proc means data=height noprint;
    var height;
    output out=height_mean mean=ht_mean;
run;

data _null_;
    set height_mean;
    call symputx('mean_ht1', ht_mean);
run;

%put macro variable mean_ht1: &mean_ht1;
```

SAS Log:

```
551  %put macro variable mean_ht1: &mean_ht1;
macro variable mean_ht1: 62.75
```

CREATING MACRO VARIABLES BY USING MULTIPLE INTO CLAUSES

Program 4 above created four macro variables, JOHN_HT, TOM_HT, MARY_HT, and HELEN_HT, that contains the heights for John, Tom, Mary, and Helen. You can accomplish the same tasks by using multiple INTO clauses.

Program 10:

```
proc sql noprint;
    select height into: John_ht1 from height
    where name = 'John';
    select height into: Tom_ht1 from height
    where name = 'Tom';
    select height into: Mary_ht1 from height
    where name = 'Mary';
    select height into: Helen_ht1 from height
    where name = 'Helen';
quit;
%put macro variable John_ht1: &John_ht1;
%put macro variable Tom_ht1: &Tom_ht1;
%put macro variable Mary_ht1: &Mary_ht1;
%put macro variable Helen_ht1: &Helen_ht1;
```

SAS Log:

```
563  %put macro variable John_ht1: &John_ht1;
macro variable John_ht1:       65
564  %put macro variable Tom_ht1: &Tom_ht1;
macro variable Tom_ht1:       60
565  %put macro variable Mary_ht1: &Mary_ht1;
macro variable Mary_ht1:       62
566  %put macro variable Helen_ht1: &Helen_ht1;
macro variable Helen_ht1:       64
```

CREATING A RANGE OF MACRO VARIABLES BY USING ONE INTO CLAUSE

Similar to the SYMPUT routine, you can also create a range of macro variables. Each of the macro variables will contain each row in the result of the SELECT statement. You only need one INTO clause to accomplish this task. Here's the general syntax:

```
PROC SQL <PRINT|NOPRINT>;
  SELECT column1 <, column2, …>
  INTO :macro-variable1_1 -  :macro-variable1_n < NOTRIM>
     <, :macro-variable2_1 -  :macro-variable2_n < NOTRIM>, …>
   FROM table |view
   <other clauses>;
 QUIT;
```

*Macro-variable1_**1*** to *macro-variable1_**n*** are the names of the variables that contain values from *column1*. *Macro-variable2_**1*** to *macro-variable2_**n*** are the names of the variables that contain values from *column2*. By default, the leading and trailing blanks are removed from values before they are stored in macro variables. If you don't want to remove the leading and trailing blanks, you can use the NOTRIM option.

Suppose that you would like to create macro variables NAME1 – NAME4 and HEIGHT1 – HEIGHT4 to store these four students' names and their weights. You can write the following code.

Program 11:
```
proc sql noprint;
    select name, height
    into :name1 - :name4,
         :height1 - :height4
    from height;
quit;

%put macro variable name1: &name1;
%put macro variable name2: &name2;
%put macro variable name3: &name3;
%put macro variable name4: &name4;
%put macro variable height1: &height1;
%put macro variable height2: &height2;
%put macro variable height3: &height3;
%put macro variable height4: &height4;
```

SAS Log:
```
663  %put macro variable name1: &name1;
macro variable name1: John
664  %put macro variable name2: &name2;
macro variable name2: Tom
665  %put macro variable name3: &name3;
macro variable name3: Mary
666  %put macro variable name4: &name4;
macro variable name4: Helen
667  %put macro variable height1: &height1;
macro variable height1: 65
668  %put macro variable height2: &height2;
macro variable height2: 60
669  %put macro variable height3: &height3;
macro variable height3: 62
670  %put macro variable height4: &height4;
macro variable height4: 64
```

When creating a range of macro variables by using the INTO clause, PROC SQL restricts the name convention for the macro variables. The names of the macro variables must end with an integer with a valid range, such as HEIGHT1 – HEIGHT4. You won't be able to use more meaningful names such as JOHN_HT, TOM_HT, MARY_HT, and HELEN_HT.

CREATING A MACRO VARIABLE THAT HOLDS ALL VALUES OF A DATASET VARIABLE

You can also the INTO clause to create a macro variable that holds all the values of a column by concatenating them and separating them by a delimiter. Here is the general syntax:

```
PROC SQL <PRINT|NOPRINT>;
  SELECT column1 <, column2, …>
  INTO :macro-variable1 SEPARATED BY 'delimiter1'
      <, :macro-variable2 SEPARATED BY 'delimiter2', …>
  FROM table |view
  <other clauses>;
 QUIT;
```

Delimiter1 is used to separate all the values in the column and must be enclosed in quotation marks.

Suppose that you would like to create a macro variable, NAMELIST, that holds all the values for the NAME column. Each value of the names are separated by a blank. Supposed that you would also like to create another macro variable, HEIGHTLIST, to hold all the values for the HEIGHT columns, separated by a comma. You can write the following code:

Program 12:
```
proc sql noprint;
    select name, height
    into : namelist separated by ' ',
         : heightlist separated by ','
    from height;
quit;

%put macro variable namelist: &namelist;
%put macro variable heightlist: &heightlist;
```

SAS Log:
```
871  %put macro variable namelist: &namelist;
macro variable namelist: John Tom Mary Helen
872  %put macro variable heightlist: &heightlist;
macro variable heightlist: 65,60,62,64
```

Program 12 can also be re-written to accomplish the same task without using the SQL procedure. Here's the alternative way by utilizing %LET, CALL SYMPUTX, and CALL EXECUTE. Compared to the method below, utilizing the INTO clause from PROC SQL seems to be much simpler.

Program 13:
```
%let namelist1=;
%let heightlist1=;
data _null_;
    set height;
    call symputx('name_temp',name);
    call symputx('height_temp',height);
    call execute('%let namelist1=&namelist1 &name_temp;');
    if _N_ =1 then
        call execute('%let heightlist1=&heightlist1 &height_temp;');
    else
        call execute('%let heightlist1=&heightlist1,&height_temp;');
run;

%put macro variable namemacro1: &namemacro1;
%put macro variable heightlist1: &heightlist1;
```

SAS Log:

```
876  %put macro variable namemacro1: &namemacro1;
macro variable namemacro1: John Tom Mary Helen
877  %put macro variable heightlist1: &heightlist1;
macro variable heightlist1: 65,60,62,64
```

CONCLUSION

Understanding the mechanisms of macro processing is essential for creating macro variables precisely. Creating a macro variable by using the %LET statement occurs before the execution of any other SAS language statements. To create a macro variable during the DATA step execution, you must use either the SYMPUT or SYMPUTX routines. Lastly, the macro variables that are created by the INTO clause occurs during the execution of PROC SQL. In this paper, we didn’t discuss how to write a macro program. But understanding the mechanisms of creating macro variables is an important foundation for learning how best to write macro programs.

REFERENCES

Burlew, Michele M. SAS® Macro Programming Made Easy, 2nd Edition.
SAS Online Doc® 9.1.3. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to thank Kathryn McLawhorn and Scott McElroy, Technical Support Analysts from SAS Technical Support, for their valuable programming suggestions and insight.

CONTACT INFORMATION

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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.