INTRODUCTION

SAS® macro language provides the user with many capabilities including conditional execution of the SAS DATA and PROC steps, generation of data-dependent SAS code, and generation of repetitive SAS code. The functions available in SAS macro language further facilitate the user's ability to evaluate expressions, manipulate character strings, and handle special characters (quoting functions). Some of these functions will be described in detail along with examples of how they can be used. This paper focuses on those functions used to evaluate expressions and manipulate character strings, with some discussion of quoting functions.

MACRO VARIABLES

Macro variables fall into two classes, automatic which are defined by the SAS supervisor; and user-defined. User-defined variables can be assigned in several ways. Most examples in this paper show macro variables created with the %LET statement. The general form of the %LET statement is

%LET varname = value;

where varname follows the rules for SAS names and value is any string or macro expression. Thus, value can be constant text, a null value (zero characters), macro variable references, macro functions, or macro calls. Leading and trailing blanks are stripped away.

The macro processor handles value as a string regardless of whether it is made up of characters, digits, or other keyboard symbols. Quotes are not required when assigning value and are included as part of value if they are used.

Values of macro variables can be displayed using the %PUT statement. The form of this statement is

%PUT text;

where any macro variable references in text will be resolved before the text is written.

SAMPLE PROGRAM (MACFUN)

To demonstrate the various MACro FUNCTIONs, macro MACFUN will be developed. Hypothetically, MACFUN calls another macro which accesses a large database and generates reports for sites specified by the user. Among the variables included in the database are CITY and STATE. The user specifies the sites by interactively entering the names of the cities and states for which reports are to be generated. If desired, the user can request a report for the whole state by not specifying a city. This paper is not concerned with either the final output or the processes used to generate the output, but with the statements in MACFUN.

The steps in MACFUN are:

1) Request that the site be input in a specific manner.
2) Accept the responses entered interactively.
3) Create a macro variable (&COLLECT) to collect all the sites which have been input.
4) Count the number of sites entered.

The following statements handle these steps:

%PUT A report will be produced for each site you enter.;
%PUT ENTER SITE - either City, State or State (2 letter abbr. for state);
%LET COUNT = 0;
%DO %UNTIL(%SYSBFRR EQ )
%INPUT;
%LET COUNT = %EVAL(%COUNT + 1);
%IF &COUNT = 1 THEN %LET COLLECT = &SYSBFRR;
%ELSE %LET COLLECT = &COLLECT/&SYSBFRR;
%END;

As the entries may contain blanks, the value entered is assigned to the automatic macro variable &SYSBFRR. The examples in this paper are based on these four entries:
NEW YORK, NY
philadelphia, PA
Boston, ma

The variable &COLLECT after these entries are made resolves to
NEW YORK, NY/philadelphia, PAlBoston, MA/lo/

Step 4 will be addressed in detail in the next section. The sections which follow provide discussions of the additional MACFUN statements which perform these steps:

5) Extract each site from &COLLECT.
6) Determine the length of the value for each site so that it can be centered in the title when the final report is generated via a DATA step.
7) Determine if each site includes a city or only a state.
8) Create two distinct variables, the values of which are the city and state portions of each site.
9) Select the appropriate observations from the database.

One may question the wisdom of combining into a single variable (&COLLECT) data that was entered individually only to extract the identical individual elements (Step 5). This was done so that the user would not have to wait for what could be a rather lengthy program to finish processing one site before the next site could be entered. While this technique furnishes a good example for demonstrating the usefulness of macro functions, other programming techniques could have also been used.

Appendix A shows MACFUN in its entirety.

EVALUATING EXPRESSIONS

These statements show the value of DATA step variable C after values are assigned values in a routine manner:

DATA EVAL1;
A = 4;
B = 7;
C = A + B;
PUT 'C = ' C; => C = 11

Because SAS macro language processes character strings, macro variable assignments cannot be made in this manner. Compare the value of macro variable &C if statements similar to those used for DATA step variables are used:

%LET A = 4;
%LET B = 7;
%LET C = &A + &B;
%PUT ' &C=' &C; => '&C=' 4 + 7

To evaluate arithmetic and logical expressions in the macro language, use the %EVAL function. The general form of this function is

%EVAL(expression).

The precedence and name of each operator that can be used in expression are listed below. Either the mnemonic or the operator for each may be used. Any operations within parentheses are executed prior to any of the these operations.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operation (Mnemonic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>exponentiation</td>
</tr>
<tr>
<td>2</td>
<td>positive and negative prefixes</td>
</tr>
<tr>
<td>3</td>
<td>logical not (NOT)</td>
</tr>
<tr>
<td>4</td>
<td>multiplication, division</td>
</tr>
<tr>
<td>5</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>6</td>
<td>comparison(LT, LE, EQ, NE, GT, GE)</td>
</tr>
<tr>
<td>7</td>
<td>logical and (AND)</td>
</tr>
<tr>
<td>8</td>
<td>logical or (OR)</td>
</tr>
</tbody>
</table>

The arithmetic operations performed with %EVAL are limited to integer arithmetic. If the result of a division of two integers is noninteger, the fractional part is discarded. Noninteger values, such as decimal points in expression with arithmetic operators will cause an error message.

Examples of the results generated using the %EVAL function with the values assigned to &A and &B as above are:

%LET C = %EVAL(&A + &B);
%PUT ' &C=' &C; => '&C=' 11

%LET D = %EVAL(&B/&A);
%PUT ' &D=' &D; => '&D=' 1

The %EVAL function is often used to increment a counter. This is exactly what Step 4 of MACFUN
MACFUN Step 4 - Count the number of sites entered.

The statements which follow show that the variable &COUNT is incremented each time a new site is entered by the user. Because this variable will increment one additional time when no site is actually entered, after the %DO loop the %EVAL function is again used to decrease the counter by 1 and assign this value to &N. Since four sites were entered, &COUNT and &N resolve to 5 and 4, respectively.

%LET COUNT = 0;
%DO %UNTIL(&SYSBUFFR EQ );
  %INPUT;
  %LET COUNT = %EVAL(&COUNT + 1);
  %IF &COUNT = 1 %THEN %LET COLLECT = &SYSBUFFR;
  %ELSE %LET COLLECT = COLLECT/&SYSBUFFR;
%END;
%PUT '&COUNT' = &COUNT,
=> &COUNT' = 5
%LET N = %EVAL(&COUNT -1);
%PUT '&N' = &N; => '&N' = 4

Discussion of the %EVAL function thus far has focused on explicit use of %EVAL. This function is also used implicitly in macro language %DO and %IF statements as well as in other macro functions. For instance, in the preceding example the EQ or '=' comparator was used in the %DO %UNTIL and %IF statements. Each of these statements contains an implied %EVAL function that evaluates the comparison and returns a value of 1 (true) or 0 (false).

HANDLING SPECIAL CHARACTERS

To assign a string to a DATA step character variable the string is enclosed in quotes as in:

DATA _NULL_;
  LENGTH A $ 12.;
  A = 'A , / % & , ;';
  PUT 'A = ' A;
  => A = 'A , / % & , ;'

Variable A contains several characters that usually have syntactical meaning. But, as part of the character string assigned to variable A, these characters have no special connotation.

In the macro language, quotes (single and double) are treated as text. To accomplish the operation equivalent to enclosing a value in quotes, it is necessary to use one of the quoting functions. Several quoting functions are available. The discussion here will concentrate on the %STR and %QUOTE functions. Both functions remove the syntactical meanings of these special characters:

; , + - */ ** < > | - =

In addition, the meanings of the mnemonics (LT LE EQ NE GT NE AND OR) used in an %EVAL function (either implicit or explicit) are ignored. Leading and trailing blanks which would otherwise be stripped away are preserved when these functions are used. The general form of each function is

%STR(argument)

or

%QUOTE(argument)

where the argument to %STR can be any string and the argument to %QUOTE can be any macro expression. The %STR function quotes the constant text in argument at macro compilation. The %QUOTE function quotes the result of resolving argument during macro execution.

%STR Examples:

The following statements would generate an error message as the semicolon following SORT would end the %LET statement:

%LET A = PROC SORT; BY STATE;
%PUT '&A' = &A; => '&A' = PROC SORT

Use of the %STR function produces the desired string:

%LET A = %STR(PROC SORT; BY STATE);
%PUT '&A' = &A;
  => '&A' = PROC SORT; BY STATE;

The following statements show the effect of the %STR function on leading and trailing blanks:

82
The macro function %NRSTR is similar to the %STR function. %NRSTR also quotes the special characters % and &.

%QUOTE Example:

Suppose MACFUN invoked another macro if the state entered indicated Hawaii (HI). The following statement would then be included in MACFUN:

%IF &STATE = HI %THEN %HAWAII;

If macro variable &STATE resolved to either OR (Oregon) or NE (Nebraska), an error message would result. To avoid this, the %QUOTE function should be used:

%IF %QUOTE(&STATE) = HI %THEN %HAWAII;

There are 3 other macro functions which also quote the result of argument during macro execution. In addition to quoting those characters quoted by %QUOTE, %NRBQUOTE quotes the special characters % and &. %BQUOTE quotes unanticipated special characters, and %NRBQUOTE quotes unanticipated characters as well as the special characters % and &.

The two other quoting functions are %SUPERQ which prevents any resolution of the value of the macro variable named in its argument and %UNQUOTE which undoes quoting.

A value quoted using any of the quoting functions will remain quoted until unquoted via the %UNQUOTE function. If the value is part of generated SAS statements, the value remains quoted until the SAS compiler receives the text for the construction of a DATA or PROC step or %GLOBAL statement.

MANIPULATING CHARACTER STRINGS

There are five DATA step functions used for manipulating strings that have counterparts in the macro language. These macro functions are %INDEX, %LENGTH, %SCAN, %SUBSTR, and %UPCASE. The other macro functions which manipulate character strings are %QSCAN, %QSUBSTR, and %QUPCASE. Values returned by %SCAN, %SUBSTR, and %UPCASE are unquoted, while %QSCAN, %QSUBSTR, and %QUPCASE, respectively, perform the same tasks but return quoted results.

The usual argument(s) (including those referred to as argument, n, delimiter, or position) to each of these macro functions can be constant text, a macro variable reference, another macro function, or a macro call.

The %SCAN and %QSCAN Functions

Both the %SCAN and the %QSCAN functions scan for words. The general form of the %SCAN function is

%SCAN(argument,n,delimiter)

The value returned is the \textit{ith} word of \textit{argument}. Words are considered to be character strings separated by at least one delimiter. In addition to the usual types of arguments, \textit{n} can be an expression, the result of which will be truncated if noninteger. The default delimiters are \texttt{.,-",'\$"*};\texttt{-.,-",".'}. Examples:

%LET TEXT = SHE WAS BORN ON 3/26/70;

If the number of words in argument is less than \textit{n} a null value will be returned.
%LET BDATE = 3/26/70;
%LET BDAY = %SCAN(&BDATE, 4, /);
%PUT &BDAY = &BDAY;
=> &BDAY =

MACFUN Step 5 - Extract each site from &COLLECT.

Recall that the current values of MACFUN macro variables &N and &COLLECT are, respectively, 4 and

NEW YORK, NY/philadelphia, PA/Boston, ma/lo/

Each site entered except the last one (lo) contains a comma, the character used to separate function arguments. The %QUOTE function is used to remove the syntactical meaning of the comma:

%DO I = 1 %TO &N;
  %LET QCOLLECT = %QUOTE(&COLLECT);
  %LET SITE = %SCAN(&QCOLLECT, &I, /);
  %PUT &I &SITE;
%END;

The two %LET statements could have been combined into the one statement

%LET SITE =
  %SCAN(%QUOTE(&COLLECT), &I, /);

in which argument is a macro function. The %PUT statement produces:

1  NEW YORK, NY
2  philadelphia, PA
3  Boston, ma
4  lo

Alternatively, it may be preferable to have a quoted result. This can be accomplished by using the %QSCAN function:

%LET QSITE =
  %QSCAN(%QUOTE(&COLLECT), &I, /);

The %LENGTH Function

The %LENGTH function ascertains the length of a character string. The general form of the %LENGTH function is

%LENGTH(argument);

Unlike the DATA step LENGTH function which will always return a value ≥ 1, this function will return a value of 0 (zero) if argument has a null value.

Examples:

%LET EX1 = ABDEFGH;
%LET LENEX1 %LENGTH(&EX1);
%PUT &LENEX1 = &LENEX1;
=> &LENEX1 = 7

%LENGTH ignores leading and trailing blanks:

%LET EX2 = ABDEFGH;
%LET LENEX2 = %LENGTH(&EX2);
%PUT &LENEX2 = &LENEX2;
=> &LENEX2 = 7

If argument is a macro call, the %LENGTH function returns the length of the result of the macro call. The following statements return the length of the string THEY ARE EQUAL:

%MACRO TEST;
  %IF &LENEX1 = &LENEX2
    %THEN THEY ARE EQUAL;
    %ELSE;
  %END TEST;

%LET COMPARE = %LENGTH(%TEST);
%LET COMPARE = &COMPARE;
=> &COMPARE = 14

MACFUN Step 6 - Determine the length of the value for each site so that it can be centered in the title when the final report is generated via a DATA step.

%LET LENSITE = %LENGTH(&SITE);

The accompanying statement

%PUT &SITE = &SITE
  &LENSITE = &LENSITE;

displays the following for the sites entered:

'&SITE' = NEW YORK, NY  &LENSITE' = 12
'&SITE' = philadelphia, PA
  &LENSITE' = 16
'&SITE' = Boston, ma  &LENSITE' = 10
'&SITE' = lo  &LENSITE' = 2

The %INDEX Function

The %INDEX function finds the first occurrence of a
character string. The general form of the %INDEX function is

    %INDEX(argument1,argument2);

The returned value is the position in argument1 of the first occurrence of argument2. If argument2 is not found in argument1, a value of 0 (zero) is returned.

Examples:

%LET CONF = NESUG94;

In this example both argument1 and argument2 are macro variable references:

%LET YR = 94;
%LET POS = %INDEX(&CONF,&YR);
%PUT '&POS' = &POS; => '&POS' = 6

Argument2 is constant text in this example:

%LET POS = %INDEX(&CONF,SUGI);
%PUT '&POS' = &POS; => '&POS' = 0

Consider a program which allows the user to specify the descriptive statistics to be output from PROC MEANS. The option specified is assigned to macro variable &OPT. The %INDEX function can be used to determine if the option specified is valid. If the returned value for &VALID is 0 (zero) an invalid option was requested:

%LET VALID = %INDEX(N MEAN STD MIN MAX T PRT, &OPT);

MACFUN Step 7 - Determine if each site includes a city or only a state.

MACFUN Step 8 - Create two distinct variables, the values of which are the city and state portions of each site.

Since commas separate the city from the state, the position of the comma in the string should be identified. Argument2, which is the comma, must be quoted and the %STR function is used.

If argument1 is %SITE, the result of using the %QSCAN function in Step 5, there is no need to quote argument1:

%LET COMMAI = %INDEX(%QSCAN(%STR(,)),);
%PUT '&SITE' = &SITE
    '&COMMAI' = &COMMAI;

The values returned for &COMMAI are the same in each case. The output provided by the %PUT statement is:

'&SITE' = NEW YORK, NY 'COMMAI' = 9
'&SITE' = philadelphia, PA 'COMMAI' = 13
'&SITE' = Boston, ma 'COMMAI' = 7
'&SITE' = Io 'COMMAI' = 0

When &SITE (or &QSITE) = Io, no comma was found and the returned value is zero, indicating that no city was entered.

The %SUBSTR and %QSUBSTR Functions

The %SUBSTR function produces a substring from a character string. The general form of the %SUBSTR function is

    %SUBSTR(argument,position,[n]);

In addition to the usual types of arguments, position and n can be the result of a macro expression. In any case, they must either be numbers or resolve to numbers.

The character string in argument is read from position for a length of n characters. It is important to note that n is the length to be read, and not the last character position to be read.

%SUBSTR is similar to the DATA step SUBSTR function when the DATA step function is used on the right side of the assignment statement. The DATA step SUBSTR function can also be used on the left side of the assignment statement to replace character value contents. %SUBSTR cannot be used in this manner.

Examples:

%LET TEXT = SHE WAS BORN ON 3/26/70;
%LET DAY = %SUBSTR(&TEXT,19,2);
%PUT ' &DAY' = &DAY;  => ' &DAY' = 26

If n is not supplied, the function reads from position to the end of argument:

%LET BDATE = %SUBSTR(&TEXT, 17);  
%PUT ' &BDATE' = &BDATE;
=> ' &BDATE' = 3/26/70

An n value that is larger than the length of the argument will result in a returned value from position to the end of argument, along with a warning:

%LET BDATE = %SUBSTR(&TEXT, 20, 5);  
%PUT ' &BDATE' = &BDATE;
=> ' &BDATE' = 6/70

MACFUN Step 8 (continued) - Create two distinct variables, the values of which are the city and state portions of each site.

The %INDEX function in the last section located the commas in each of the sites and stored those positions in the macro variable &COMMAI. Having this information, the %SUBSTR function can now be used to provide values for &CITY and &STATE from &QSITE, the quoted variable containing the site.

If a city name was included in the site it should begin in the first character of the string and end one position before the comma appears in the string. The ending position is determined as:

%LET POS = %EVAL(&COMMAI - 1);

Because a state may have been entered without a city, assign a null value to &CITY if no comma was found in the character string. The %QUOTE function is not required since &QSITE is the argument.

%IF &COMMAI NE 0 %THEN %LET QCITY = %SUBSTR(&QSITE, 1, &POS);  
%ELSE %LET QCITY = ;
%PUT ' &QSITE' = &QSITE ' &CITY' = &CITY;

This %PUT statement output shows the values assigned to &CITY for each site:

' &QSITE' = NEW YORK, NY  
' &CITY' = NEW YORK
' &QSITE' = philadelphia, PA  
' &CITY' = philadelphia
' &QSITE' = Boston, ma ' &CITY' = Boston
' &QSITE' = Io ' &CITY' = Io

Similarly, the value to be assigned to macro variable &STATE starts in the position following the comma and completes the string. Instead of using the %LET statement to assign the value of position as was done to assign the value of &CITY, the value for position is the result of a macro expression:

%IF &COMMAI NE 0 %THEN %LET STATE = %SUBSTR(&QSITE, %EVAL(&COMMAI + 1));  
%ELSE %LET STATE = &SITE;
%PUT ' &QSITE' = &QSITE ' &STATE' = &STATE;

The display produced by the %PUT statement for each entry is:

' &QSITE' = NEW YORK, NY ' &STATE' = NY  
' &QSITE' = philadelphia, PA  
' &STATE' = PA
' &QSITE' = Boston, MA ' &STATE' = ma  
' &QSITE' = Io ' &STATE' = Io

%QSUBSTR can be used in the same way as %SUBSTR if the returned value is to be quoted. Any value returned by the %SUBSTR function is unquoted. Therefore, any blanks between the comma and the state, or following the state are stripped away. If %QSUBSTR had been used instead of %SUBSTR, these blanks would remain in values assigned to &STATE.

%UPCASE and QUPCASE

The %UPCASE function, like the DATA step UPCASE function, translates lowercase characters to uppercase. The general form of the %UPCASE function is

%UPCASE(argument)

where argument can be any string.

This function is useful if argument will be used in comparisons, because the lowercase characters are not automatically translated to uppercase characters before a comparison is made.

Example:

%LET RESPONSE = yes;
%MACRO YESNO;
  %IF &RESPONSE = YES %THEN %PUT AFFIRMATIVE;
The result of this macro call will be NEGATIVE because yes = YES does not satisfy the %IF condition.

Replacing

&RESPONSE with %UPCASE(&RESPONSE)

in the %IF statement would satisfy the %IF condition for all of these values of &RESPONSE:

YES
yes
Yes

MACFUN Step 9 - Select the appropriate observations from the database.

Now that the values for &CITY and &STATE have been assigned in Step 8, the database observations corresponding to these sites can be selected. A sample of the values for the variables CITY and STATE in this database (SAS DATA set NESUG94.SAMPLE) is provided in Appendix B. Appendix B shows that the values for these two variables are not consistent with respect to case. That is, some values are entirely uppercase, some are entirely lowercase, and others have mixed cases. In addition, recall that the values input to MACFUN also vary with respect to case.

For each of the sites entered, the statement

%PUT %UPCASE(&CITY), %UPCASE(&STATE);

would display

NEW YORK, NY
PHILADELPHIA, PA
BOSTON, MA

The statements which follow are part of the DATA step in a macro called by MACFUN, and show how both the DATA step UPCASE and the macro %UPCASE functions are used to ensure that all appropriate observations will be selected from the database, regardless of case.

DATA SELECT;
SET NESUG94.SAMPLE;
%IF &CITY NE %THEN
  IF (UPCASE(CITY) = "%UPCASE(&CITY)") AND;
  %ELSE IF (UPCASE(STATE) = "%UPCASE(&STATE)"") THEN DO;
    PUT &CITY &STATE &CITY;
    OUTPUT;
  END;

When &CITY is not null the statement generated by the %IF condition is

IF (UPCASE(CITY) = "%UPCASE(&CITY)") AND
  (UPCASE(STATE) = "%UPCASE(&STATE)"")
THEN DO;

When &CITY is null the statement generated by the %IF condition is

IF (UPCASE(STATE) = "%UPCASE(&STATE)"")
THEN DO;

In these statements, if &CITY is not null, the values of both CITY and &CITY are translated to uppercase and compared. The same is done for the values of STATE and &STATE, regardless of the value for &CITY. Records are OUTPUT when the IF conditions are true.

The display produced by the PUT statement for the entered value

philadelphia, PA

is

PA       PHILADELPHIA
PA       Philadelphia
pa       philadelphia

If the variable &QSITE were to be translated to uppercase prior to extracting the values for &CITY and &STATE, the returned value would contain special characters and require quoting for additional string manipulations. In that case, the %QUPCASE function could be used to translate the value of QSITE to uppercase and return a quoted value.

MACRO FUNCTIONS OR DATA STEP FUNCTIONS?

Generally, macro functions are used with macro variables. In the DATA step, the DATA step function
can often be used with a macro variable. For instance, in the previous example, the same output would have been produced if

```
UPCASE("&CITY");
```

had been used in place of

```
"%UPCASE(&CITY)";
```

However, when using the DATA step functions with macro variables, bear in mind that the macro processes are performed by the macro facility prior to reaching the SAS compiler. For example, including this statement in a DATA step would generate an error message:

```
Pt1'l' UPCASE ("&CITY") ;
```

while this statement would not:

```
PUT UPCASE ("&CITY") ;
```

CONCLUSION

The macro functions which are available have been discussed along with some examples. The SAS manual *SAS® Guide to Macro Processing* provides a far more in depth discussion of these functions.


---

APPENDIX A

Macro MACFUN

```
%MACRO MACFUN;
%* Request that the site be input in a specific manner;
%PUT A report will be produced for each site you enter.;
%PUT ENTER SITE - either City, State OR State (2 letter abbr. for state); RUN;
%* Initialize counter;
%LET COUNT = 0;
%DO %UNTIL($SYSBUFFER EQ );
%* Accept input;
%INPUT;
%* Increment counter;
%LET COUNT = %EVAL(%COUNT + 1);
%* Collect the sites entered into one variable;
%* ($COLLECT);
```

---

APPENDIX B

Sample of Variables STATE and CITY in NESUG94.SAMPLE

<table>
<thead>
<tr>
<th>CITY</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOS ANGELES</td>
<td>CA</td>
</tr>
<tr>
<td>peoria</td>
<td>TX</td>
</tr>
<tr>
<td>BOSTON</td>
<td>MA</td>
</tr>
<tr>
<td>Boston</td>
<td>MA</td>
</tr>
<tr>
<td>Newark</td>
<td>NJ</td>
</tr>
<tr>
<td>NEW YORK</td>
<td>NY</td>
</tr>
<tr>
<td>NEW YORK</td>
<td>NY</td>
</tr>
<tr>
<td>PHILADELPHIA</td>
<td>PA</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>PA</td>
</tr>
<tr>
<td>wheeling</td>
<td>WV</td>
</tr>
<tr>
<td>Sioux Rapids</td>
<td>IA</td>
</tr>
<tr>
<td>Anita</td>
<td>IA</td>
</tr>
<tr>
<td>Sioux City</td>
<td>IA</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>PA</td>
</tr>
</tbody>
</table>

---