Paper 040-31

Tight Looping With Macro Arrays

Ted Clay, Clay Software & Statistics, Ashland, Oregon

ABSTRACT

You can apply the power of macro %DO-looping in the tight space of a single statement by using macro arrays. A macro array is a list of macro variables that share the same prefix and a numeric suffix, for example, AA1, AA2, AA3, etc., plus an additional macro variable with a suffix of "N" that contains the length of the array. Two macros, %ARRAY and %DO_OVER, make it simple to create and use macro arrays. %ARRAY stores text in a macro array from an explicit list of values or from variables in a data set. Then, wherever you need those text values in your program, %DO_OVER loops over the macro array and substitutes the text values wherever you put a "?" in your code phrase.
%DO_OVER(AA,PHRASE="?") generates a list with double-quotation marks around each text string.
SET %DO_OVER(AA,PHRASE=mylib.?); concatenates a series of data sets.
RENAME %DO_OVER(AA,PHRASE=?=pre_?); renames a list of variables by adding a prefix.
%DO_OVER(AA,MACRO=mymacro) repeatedly executes a macro with a series of values.
The code phrase can consist of many statements, and multiple arrays can be defined and looped-over in parallel.
The %ARRAY and %DO_OVER macros turn many time-consuming programming tasks into quick work.

INTRODUCTION

This paper explains what a macro array is, how to put text into it, and how to loop (iterate) over the text values stored in it. Two macros, %ARRAY and %DO_OVER, make the use of macro arrays painless and fun. Macro %DO-looping is common practice in many programs using the macro language. Unfortunately, the usual solutions involve somewhat awkward macro syntax, with double ampersands and macro definitions which can break up the continuity of programs. These two macros allow you to hide the repetitive machinery, resulting in programs that are shorter and more readable.

The %ARRAY and %DO_OVER macros are analogous to the ARRAY and DO OVER statements in the SAS® data step language, which define and loop over implicitly subscripted arrays. Because these macros are self-contained and use global macro variables, you can use them freely in "open code". The macros use regular characters as much as possible, freeing you from the need for macro quoting functions.

This paper covers "The Basics", followed by "Full Features" and "Syntax", wrapping up with "How it Works".

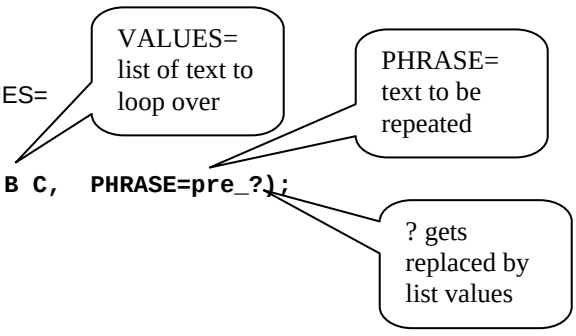
PART 1 -- THE BASICS

In the data step language, you must always explicitly define an array before you can use DO OVER. But the %DO_OVER macro uses a VALUES= parameter which allows you to skip the %ARRAY statement. We start here.

%DO_OVER WITH VALUES=

The power of the macro array concept is exploited by the %DO_OVER macro. The author of this article has found that more than half the time, he only uses the %DO_OVER macro, without needing the %ARRAY macro, and without needing to think about the macro array at work behind the scenes. The array itself is hidden inside the %DO_OVER macro. The following examples show the power of %DO_OVER alone.

Basic Example:
%DO_OVER with VALUES=




```
%DO_OVER(VALUES=A B C,  PHRASE=pre_?);
```

generates:

```
pre_A pre_B pre_C
```

Table 1: Applications of %DO_OVER with different PHRASE parameters

Application	Code Example	Generates
1. Bulk renaming	`rename %DO_OVER(VALUES=A B C,` `  PHRASE=?=pre_?) ;`	`rename A=pre_A B=pre_B C=pre_C;`

2. A quoted list	`if letter in (`
	` %DO_OVER(VALUES=A B C,`
	`  PHRASE="?"));`
		`if letter in ("A" "B" "C");`

3. Tracking merged data	`Merge %DO_OVER(VALUES=A B C,`
	`  PHRASE=?(in=in?));`
		`merge A(in=inA)`
		`  B(in=inB)`
		`  C(in=inC);`

4. Complete statements	`Proc freq;`
	`%DO_OVER(VALUES=A B C,`
	`PHRASE=table ? / out=freqs?;)`
		`Proc freq;`
		`table A / out=freqsA;`
		`table B / out=freqsB;`
		`table C / out=freqsC;`

5. Multiple statements	`%DO_OVER(VALUES=A B,`
	`PHRASE=`
	`title "Printout of ?";`
	`proc print data = ?;)`
		`Title "Printout of A";`
		`Proc print data = A;`
		`Title "Printout of B";`
		`Proc print data = B;`

6. Macro language uses	`%LET OLD=A B C;`
	`%LET NEW=%DO_OVER(VALUES=&OLD,`
	`  PHRASE=pct_?);`
		`Pct_A pct_B pct_C`
		`  (assigned to variable NEW)`

7. The default phrase (single question-mark)	`%DO_OVER(VALUES=A B C, PHRASE=?)` `%DO_OVER(VALUES=A B C);`	`A B C` `A B C  (the same result)`

DISCUSSION

Close parentheses, and correct placement of semi-colons, make a big difference. The %DO_OVER macro begins with an open parenthesis, and needs a close parenthesis. In parsing the PHRASE= parameter, the macro processor continues until it finds a comma or an **unbalanced** close parenthesis. Items 2 and 3 above both end with the same three characters: "**));**" but they have very different meanings. In Item 2, the %DO_OVER macro was inside a pair of parentheses, so the first final close-parenthesis marks the end of the PHRASE parameter, and the second close-parenthesis and semicolon are outside the macro. In Item 3, the PHRASE contains an expression in parenthesis: "(in=in?)". It is immediately followed by %DO_OVER's closing parenthesis and the semi-colon ending the MERGE statement. In Item 4, the semi-colon is inside the macro close parenthesis. The PHRASE contains a complete SAS statement which gets repeated. In Item 5, two complete statements get repeated. Inside the DO_OVER is machinery to parse the VALUES= string into discrete items of a list, store them in a internal hidden macro array, and substitute those values in the phrase. Next we will examine macro arrays.

THE MACRO ARRAY STRUCTURE

A ***macro array*** is a list of macro variables sharing the same prefix and a numerical suffix. The suffix numbers run from 1 up to a highest number. The value of this highest number, or the ***length*** of the array, is stored in a macro variable with the same prefix, plus the letter "N". The prefix is also referred to as the ***name*** of the macro array.

Example: The macro array "DAYS" containing the first three days of the work-week.

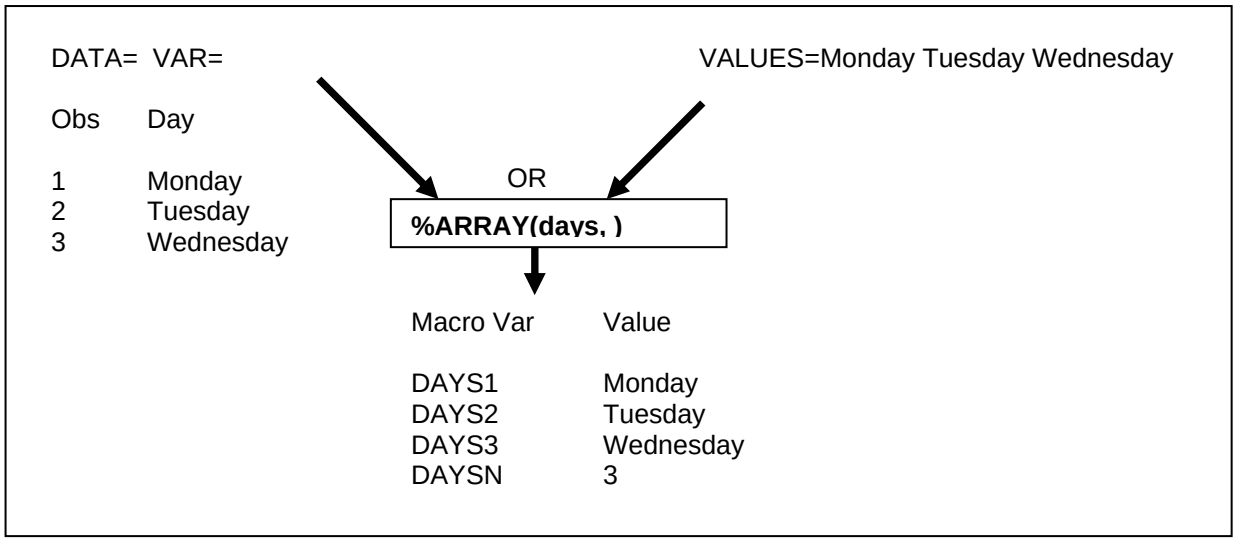
Macro Variable Name	Contents
DAYS1	Monday
DAYS2	Tuesday
DAYS3	Wednesday
DAYSN	3

The secret to the power of this structure is in the last row By also storing the length of the array, using a macro variable name with the same prefix and a standard, predictable suffix, you only need to give the prefix or name of the array, in this case "DAYS." Note that it does not use a macro variable called "DAYS" -- a macro variable and macro array of the same name can coexist.

CREATING AND USING A MACRO ARRAY

The purpose of the %ARRAY macro is to create a macro array and load data into it from either a SAS data set or an explicit list of values.

Figure 1: The %ARRAY macro accepts two possible sources of data for creating a macro array:



Why create a macro array? (1) You want to use a list in many places, (2) You like the "sound" of the abbreviated syntax. (3) It is a quick way to get variable values from a data set into the macro environment.

%ARRAY WITH VALUES=

We use the familiar VALUES= parameter, but this time to create a macro array. The macro array named "days" is used in the %DO_OVER macro. In both %ARRAY and %DO_OVER the name of the macro array is the first parameter, and it is a positional parameter.

Name of macro array being created

VALUES= used here, not with %DO OVER

```
%ARRAY(days,    VALUES=Monday Tuesday Wednesday)

Set %DO_OVER(days, PHRASE=sales.?);
```

Name of macro array instead of VALUES=

generates:

```
Set sales.Monday
    sales.Tuesday
    sales.Wednesday;
```

%ARRAY WITH DATA= AND VAR=

Often you need to create a macro array using data already stored in a data set. Suppose we have a data set "week" with a variable "day" as in the above graphic. The following code shows how you would (1) assign values from the data set into a macro array, and (2) use the macro array with %DO_OVER.

DATA and VAR describe source of values

```
%ARRAY(days,    DATA=week, VAR=day)
 if today in ( %DO_OVER(days, PHRASE="?"));
```

Name of macro array being used

generates:

```
if today in
("Monday" "Tuesday" "Wednesday")
```

PART 2 – FULL FEATURES

Table 2: Significant Additional Features.
In these examples assume that we have already created a macro array named "ABC" with values "A", "B" and "C". We would do this by **`%ARRAY(abc,VALUES=A B C);`**

Feature	Code Example	Generates
1. VALUES= a numeric list	`%ARRAY(yrs,VALUES=1983-1986);` `%PUT %DO_OVER(yrs);`	`1983 1984 1985 1986`

	`%DO_OVER(VALUES=time4-time7)`
	`%DO_OVER(VALUES=4-7,`
	`PHRASE=time?)`
		`time4 time5 time6 time7`
		
		`time4 time5 time6 time7`

2. Handling imbedded blanks	`%DO_OVER(VALUES=`
	`Alameda/San Mateo/Santa Clara,`
	`DELIM=/,PHRASE="?")`
		`"Alameda"`
		`"San Mateo"`
		`"Santa Clara"`

3. Inserting something between values	`%DO_OVER(abc,`
	`PHRASE=if letter="?" then ?=1;,`
	`BETWEEEN=else)`
		`if letter="A" then A=1;`
		`else if letter="B" then B=1;`
		`else if letter="C" then C=1;`

3b. The COMMA keyword	`maxabc = max(of %DO_OVER(abc,` `BETWEEN=COMMA));`	`Maxabc = max(of A,B,C);`

4. Inserting the array index	`Merge %DO_OVER(abc,`
	`PHRASE=?(in=in?_i_));`
		`Merge A (in=in1)`
		`B (in=in2)`
		`C (in=in3);`

5. Selecting subset of data for an array	`%ARRAY(males,`
	`DATA=subjects(where=(sex='M')),`
	`VAR=casenum)`
		`* Puts only the male subjects'`
		`casenums into a macro array;`

6. Creating multiple macro arrays from data	`%ARRAY(Xs Ys,` `DATA=temp,vars=X Y)`	`* Variable X into array Xs;` `* Variable Y into array Ys;`

DISCUSSION

Feature 1: VALUES= accepts a numbered list. This works with either %ARRAY or %DO_OVER. The syntax of a list is exactly like a numbered variable list in SAS. Note that the second example of this feature shows no PHRASE= parameter and therefore uses the default phrase of a single question-mark, which generates only the array values. The alternative is to put the stem of the numbered list into the PHRASE= parameter, as shown in the last line. Both variations generate the same code.

Feature 2: The DELIM= parameter shows how to break up a list. It is used with VALUES=, which is a parameter of either %ARRAY or %DO_OVER. The default delimiter is one or more spaces.

Feature 3: BETWEEN= with %DO_OVER generates additional code after every iteration except the last one. Because the comma is often needed between values, and a comma is a special character, the macro recognizes the special keyword COMMA**.** **`BETWEEN=COMMA`** is equivalent to **`BETWEEN=%str(,)`**. Note that in this example we used the default PHRASE= parameter, which generates just the macro array values.

Feature 4: %DO_OVER replaces the character string "?_i_" in the PHRASE= parameter with the numbers 1, 2, 3, etc.. This is analogous to the index variable _i_ of an implicitly subscripted array in the data step language.

Feature 5: You can use a WHERE= data set option. This is a regular feature of SAS, not a macro feature. This is handy when the macro array you want to create is from a subset of a data set.

Feature 6: %ARRAY with VAR= listing more than one variable will create more than one macro array. The names of the macro arrays must appear as positional parameters in the same order as the variable names. All macro arrays created from the same data set would have the same length. Variables can be either character or numeric. The values of numeric variables are converted into character strings for storage in the macro variables that make up the macro array. That conversion produces the formatted numeric values.

Table 2: Significant Additional Features. (continued)

Feature	Code Example	Generates

7. Looping over multiple macro arrays	`%ARRAY(abc,VALUES=A B C);`
	`%ARRAY(def,VALUES=D E F);`
	`Rename %DO_OVER(abc def,`
	`PHRASE=?abc=?def);`
		`Rename A=D B=E C=F;`

8. Passing array values to a macro	`%MACRO doit(xxx);`
	`<code involving &XXX>`
	`%MEND;`
	`%DO_OVER(abc,MACRO=doit)`
		`%doit(A)`
		`%doit(B)`
		`%doit(C)`

9. Passing multiple array values to a macro	`%MACRO doit(xxx,yyy);`
	`<code involving &XXX and &YYY>`
	`%MEND;`
	`%DO_OVER(abc def,MACRO=doit)`
		`%doit(A,D)`
		`%doit(B,E)`
		`%doit(C,F)`

Feature 7: %DO_OVER can loop over multiple arrays. You put the names of the multiple arrays in the same positional parameter as with a single array name. In the PHRASE= parameter, the single question mark would no longer work because there is not a single macro array. To make your intentions clear, put a question mark followed by the name of the macro array you want to insert at that place. The most common situation in programming is when the multiple macro arrays were created from the same data set.

Feature 8: You can pass macro array values to an externally-defined macro. You need this feature when the amount of SAS code in your PHRASE= parameter becomes too large. The MACRO= parameter replaces the PHRASE= parameter. The externally defined macro must be defined before you use %DO_OVER. The macro can have any name, but it must have only positional, not keyword, parameters. Then %DO_OVER will generate a series of macro invocations, each one with the values of the macro array. Although we used parameter "xxx" to receive the values of macro variable "abc", we recommend giving the same name to the parameter and the macro array.

Feature 9: Passing multiple array values to a macro is a slight variation of the prior example. Multiple array names are listed as a positional parameter. The order of the array names must correspond to the order of the positional parameters in the array definition.

PART 3 – SYNTAX

SYNTAX FOR %ARRAY

```
%MACRO ARRAY(ARRAYPOS, ARRAY=, DATA=, VAR=, VALUES=, DELIM=%str( ));
```

Parameter	Definition
ARRAYPOS	Name(s) for the macro array(s) to be defined. (Required)
ARRAY	Keyword alternative to the ARRAYPOS positional parameter..
DATA	Dataset or view source of text data. Data set options are allowed. If this parameter is used, the ARRAY macro must be called between other steps of the program.
VAR	One or more character or numeric variable(s) containing values to put in the array(s). Required with DATA=. If multiple arrays are defined, the VAR= list must correspond one-for-one with the macro names given in the ARRAY or ARRAYPOS parameter.
VALUES	An explicit list of character strings to put in the array. It can only be used when defining a single array. The VALUES= can be a numbered list. This can be number-hyphen-number, or with an alpha prefix like a SAS numbered variable list such as x3-x10.
DELIM	A single-character separator for parts of the VALUES parameter. Default=space.

The %ARRAY macro starts with the ARRAY or ARRAYPOS parameter, which gives the name of the macro array being defined, or names of the multiple arrays. %ARRAY(ARRAY=x, etc.) is absolutely equivalent to %ARRAY(x,etc.). Having both a positional and keyword version of the same parameter gives you the choice of how you want to use this macro. Most prefer the positional parameter for its brevity, and because it "sounds like" a regular implicit array statement. In this paper we consistently use only the positional parameter.

SYNTAX OF %DO_OVER

```
%MACRO DO_OVER(ARRAYPOS, ARRAY=, PHRASE=?, ESCAPE=?, BETWEEN=, MACRO=,
                                VALUES=, DELIM=%str( ));
```

Parameter	Definition
ARRAYPOS	Name(s) for the macro array(s) to iterate over. If multiple macro arrays are given, they must have the same length. The only positional parameter. Required if no VALUES=.
ARRAY	Keyword alternative to the ARRAYPOS positional parameter
PHRASE	SAS code into which to substitute the macro variable values. The PHRASE parameter may contain semicolons and extend to multiple lines. The default value of PHRASE is a single <?>. (See Example 5 for the default case.) **Single Array Case :** Put a "?" (the ESCAPE character) wherever you want the macro values to be placed. **Multiple Array Case:** Put a "?" immediately followed by the name of the macro array. The macro will replace the "?" and name with the value of that particular macro array. References to macro names are case sensitive and must agree with the ARRAY parameter values. **Obtaining the Loop Index:** Put "?_I_" wherever you want the value of the looping index to appear. It is replaced by "1", "2", "3", etc. up to the length of the array.
ESCAPE	A single character to be replaced by macro array values. Default is "?". If followed by a macro array name, it plus the array name are replaced by the array values.
BETWEEN	Code to generate between iterations of the main phrase or macro. Because there is often a need to put a comma between elements of an array, the special parameter value COMMA is recognized for programming convenience. BETWEEN=COMMA is equivalent to BETWEEN=%STR(,).
MACRO	The name of an externally-defined macro to execute on each value of the array. It overrides the PHRASE parameter. The macro must have positional parameters defined, in the same order and meaning as the macro arrays specified in the ARRAY or ARRAYPOS parameter.
VALUES	An explicit list of character strings to iterate over. This parameter is an alternative to giving an array name in the ARRAYPOS or ARRAY= parameter. A single hidden internal array is created and used. When VALUES= are given, the PHRASE parameter may only use single. If a MACRO= is supplied, it must be a single-parameter macro. The VALUES= can be a numbered list. This can be number-hyphen-number, or with an alpha prefix like a SAS numbered variable list such as x3-x10.
DELIM	A single-character separator for parts of the VALUES parameter. Default=space.

SCOPE AND PLACEMENT CONSIDERATIONS

Macro array variables are declared global, so they are available throughout a program after they are created. The %ARRAY macro with DATA= must be outside any SAS data or procedure step. No other restrictions apply to the placement of %ARRAY or %DO_OVER. So they can be located in the tight space of a single statement in open code. %DO_OVER cannot be nested, so if you use the MACRO= parameter of %DO_OVER, the external macro cannot use macro arrays. See the "FOR THE %DO-IT-YOURSELFER" section below for an easy alternative.

PART 4 – HOW IT WORKS

Recall the example of a macro array "DAYS"

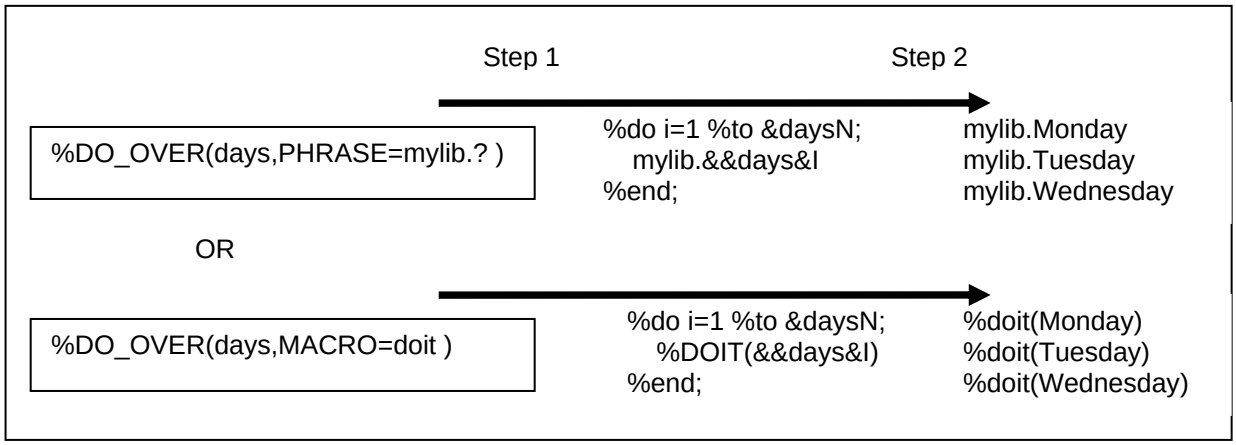
Macro Variable Name	Contents
DAYS1	Monday
DAYS2	Tuesday
DAYS3	Wednesday
DAYSN	3

The %DO_OVER macro does two things:

Step 1. Scans the PHRASE for question marks and substitutes the appropriate macro code for it. In the case of looping over the DAYS macro array, the generated macro code is of the form &&DAYS&I.

Step 2. Runs a macro %DO-loop which iterates I= 1 to the length of the macro array. When I=2, &&DAYS&I resolves to &DAYS2, which then resolves to "Tuesday".

Figure 2: The %DO_OVER macro generates a hidden %DO loop, which generates SAS code, as illustrated



THE MACRO CODE

The %DO_OVER macro starts by parsing the list of macro array names into a macro array, PREFIX1, PREFIX2, etc. with length PREFIXN. Usually this list is of length 1. The following is the central code section:

```
  * Step 1;
    %let TP=&PHRASE;
    %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE._I_,%nrstr(&I.)));                    (1)
    %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE._i_,%nrstr(&I.)));
    %do J=1 %to &prefixN;
           %let currprefix=&&prefix&J;
           %LET TP = %qsysfunc(tranwrd(&TP,&ESCAPE&currprefix,                     (2)
                                  %nrstr(&&)&currprefix%nrstr(&I..)));
           %if &prefixN=1 %then %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE,           (3)
                                  %nrstr(&&)&currprefix%nrstr(&I..)));
    %end;
  * Step 2;
    %do I=1 %to &&&prefix1.n;
         %if &I>1 and %length(%str(&between))>0 %then &BETWEEN;
         %unquote(&TP)                                                             (4)
    %end;
```

WALKING THROUGH THE MACRO EXECUTION

Using the example: `Set %DO_OVER(days,PHRASE=mylib.?(in=in?_I_));`

(1) In processing the PHRASE parameter, first any instances of "?_I_" (in upper or lower case) are replaced by the string "&I.". The value of TP (Translated Phrase) becomes `mylib.?(in=in&I.).`

(2) Next, the macro translates any instances of "?" followed by a macro name. This would be necessary when more than one macro array name is being used. In this example, nothing would happen here.

(3) Because it is true that we are using only one macro array, PREFIXN=1 and this statement executes. It translates any instance of "?". The value of TP becomes: `mylib.&&DAYS&I..(in=in&I.)`

(4) Finally, we have the %DO-loop which iterates the macro variable `"I"`. The variable TP, which up until this point has been treated as a plain text string, is now subjected to macro resolution. When `I=2`, `mylib.&&DAYS&I..(in=in&I.)` resolves to `mylib.&DAYS2.(in=in2)` which in turn resolves to `mylib.Tuesday(in=in2).` This is what the SAS statement processor receives from the macro processor.

FOR THE %DO-IT-YOURSELFER

Now that you understand macro arrays, sometimes you may want to create a macro array, and take it from there. For example, if all you need is the number of items in a list, the "length" variable of a macro array contains it. The code to do this would be:

```
%ARRAY(temp,VALUES=&mylist);
%LET mylength = &tempN;
```

A handy application of this is getting the last variable in a list of BY-variables.

```
%ARRAY(by,VALUES=&by);
%LET lastby = &&by&byN;
```

Finally, doing a macro array first can simplify the logic in a larger program. For example:

```
%ARRAY(list,VALUES=&list);
%DO I=1 %to &listN;
    < code with &&list&I >
%END;
```

This code can include `&&list&I` as a parameter value to a macro.

CONCLUSION

The concept of macro arrays closes an important gap in the SAS system. The %ARRAY macro strongly links the world of data to the world of macro variable values. Once defined, you do not have to be concerned with how many elements a macro array contains, only referring to the macro array by its name. Then, in a style similar to implicit arrays and the "DO OVER" statement in the SAS data step language, the macro %DO_OVER allows you to execute SAS code plugging in the values from macro arrays. Feel free contact the author to take advantage of these macros. They extend the power and scope of the SAS programming language.

ACKNOWLEDGEMENTS

David Katz, David Katz Consulting, for the macro array concept and its use with an externally defined macro.
Art Carpenter, California Occidental Consultants, for aid in coding at the 2003 WUSS Code Clinic.

ABOUT THE AUTHOR

Ted Clay, M.S. is a statistical consultant and data analyst. His clients have included pharmaceutical companies, manufacturing companies, and grass-roots organizations, as well as research projects in epidemiology and health policy at the University of California San Francisco.

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

Ted Clay
Clay Software & Statistics
168 Meade St.
Ashland, OR 97520
Work Phone: 541-482-6435
Fax: Same
Email: tclay@ashlandhome.net