MWSUG 2016 - Paper SA09

Array of Sunshine: Casting Light on Basic Array Processing

Nancy Brucken, inVentiv Health, Ann Arbor, MI

ABSTRACT

An array is a powerful construct in DATA step programming, allowing the application of a single process to multiple variables simultaneously, without having to resort to macro variables or repeat code blocks. Arrays are also useful in transposing data sets when PROC TRANSPOSE does not provide a necessary degree of control. However, they can be very confusing to less-experienced programmers. This paper shows how arrays can be used to solve some common programming problems.

INTRODUCTION

Arrays in SAS® are not actual data structures, as in many other programming languages. According to the *SAS 9.3 Language Reference: Concepts*, a SAS array is simply "a temporary grouping of SAS variables that are arranged in a particular order and identified by an array-name." The ARRAY statement defines this grouping, and allows you to refer to the underlying variables as array elements within a DATA step, but the array itself disappears at the close of that step, leaving only the variables themselves in the output data set. If you need to refer to the same array in a subsequent DATA step, you must define it again via a separate ARRAY statement in that step.

DEFINING AN ARRAY

The ARRAY statement allows you to define which variables should be grouped together into an array, specify the length and type of those variables, and designate a starting and ending boundary for the array index. Here's the syntax for the ARRAY statement:

ARRAY array-name { subscript } <$> <length> <array-elements><(initial-value-list)>

Array-name: This is the name you assign to the array, and must be different from the name of any SAS functions.

Subscript: This can be either a number, a range of numbers, or an asterisk, and describes the arrangement of the elements in the array. This paper will only describe one-dimensional arrays, but multidimensional arrays are also allowed.

$: Indicates that all of the array elements are character variables. Arrays may consist of all character or all numeric variables, but cannot be a mixture of the two types.

Length: Indicates the length of the variables in the array. All of the variables in the array must be of the same length.

Array-elements: This is the list of variables that will be grouped into the array, in the order in which they should be processed.

Initial-value-list: This allows you to assign an initial value to each element in the array.

ONE-DIMENSIONAL ARRAYS AND REPEATING CODE BLOCKS

Let's look at an example of a one-dimensional array, and how it can be used to avoid repeating blocks of code. Here's what we'll use as a starting point- this is a sample of code that was used to format columns of a summary table, and replace blank values with zeroes:

```
data table;
  set indata;
  length col1 col2 col3 $14;
  col1 = strip(_1);
  col2 = strip(_2);
  col3 = strip(_3);
  if missing(col1) then col1 = '0';
  if missing(col2) then col2 = '0';
  if missing(col3) then col3 = '0';
run;
```

This is a good example of repeating code blocks that can be replaced by array processing- all of the variables are the same type and length, and the same code applies to all of them. The revised DATA step, with arrays defined for the input and output column variables, looks like this:

```
data table (drop=i);
  set indata;
  array orig(*) _1 _2 _3;
  array cols(*) $14 col1 col2 col3;

  do i=1 to dim(cols);
    cols(i) = strip(orig(i));
    if missing(cols(i)) then cols(i) = '0';
  end;
run;
```

The output dataset is identical to the previous version. We've simply streamlined the code by storing the original and new column variables in arrays, and using a DO-loop to execute the same code block over all of the elements in the array. Note also that, since the type and length of the original column variables has already been defined, we do not need to specify them in the first ARRAY statement. Finally, since we are explicitly listing the variables that are assigned to each array, and we want the array index to start with the default value of 1, we can use the asterisk to specify the array structure.

It may be helpful to view an array as a matrix. For a one-dimensional array, the array index represents the column of the matrix containing that variable. Each record has its own array/matrix definitions, which are reinitialized in the Program Data Vector (PDV) by the implicit DATA step loop in the same way as any other SAS variables. Here's an example of what the PDV might look like after processing the first record in the input data set above:

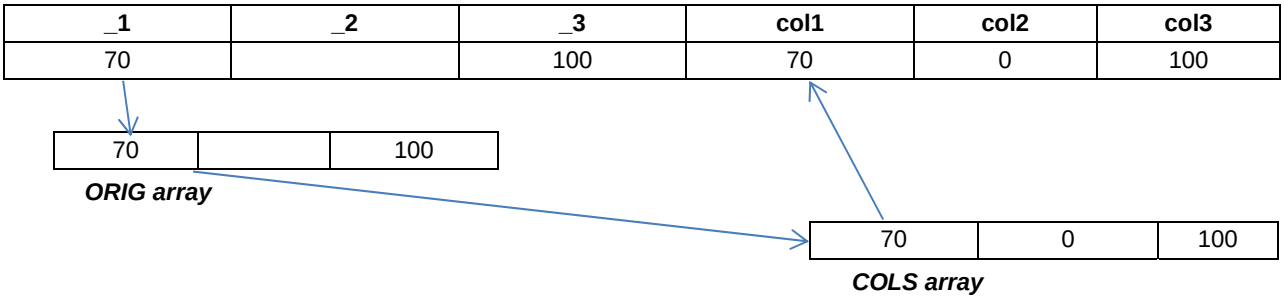

Figure 1. Array Data Flow

TRANSPOSING DATA FROM HORIZONTAL TO VERTICAL

PROC TRANSPOSE can be used for transposing data from a horizontal structure to a vertical structure, but it may not provide the desired degree of control over the attributes or contents of the resulting variables, and may require multiple steps in order to achieve the desired results. However, array processing can make this task easy.

Suppose we have the following horizontal vital signs dataset, and we need to convert it to a vertical dataset, with one record per vital sign value:

Original dataset:

HEIGHT	WEIGHT	SYSBP	DIABP	HR	TEMP
177.8	74.8	110	70	65	37.1

Desired dataset:

PARAM	PARAMCD	PARAMN	AVAL
Height (cm)	HEIGHT	2	177.8
Weight (kg)	WEIGHT	3	74.8
Systolic BP (mm Hg)	SYSBP	4	110
Diastolic BP (mm Hg)	DIABP	5	70
HR (bpm)	HR	6	65
Temperature (C)	TEMP	7	37.1

For this process to work best, we need to make sure the variable labels have been defined in the original dataset so they match the value of PARAM, minus the units, and the variable names in the input dataset match the required values of PARAMCD. In addition, the values of PARAMN run from 2 to 7, instead of starting with 1, so the code will be clearer if we can incorporate those values directly into the array index, instead of adjusting them in another variable. The following DATA step will perform the required transpose:

```
data table (drop=height weight sysbp diabp hr temp);
  set indata;
  length paramcd $ 8;
  array vsigns(2:7) height weight sysbp diabp hr temp;
  array units(6) $7 _temporary_ ('(cm)', '(kg)', '(mm Hg)', '(mm Hg)', '(bpm)',
                                 '(C)');

  do paramn=lbound(vsigns) to hbound(vsigns);
    param = catx(' ', vlabel(vsigns(paramn)), units(paramn));
    paramcd = vname(vsigns(paramn));
    aval = vsigns(paramn);

    if not missing(aval) then output;
  end;
run;
```

Note that the order in which the variables are listed when defining the VSIGNS array is important. This must match the desired values of PARAMN. Also, specifying the first and last values of PARAMN as the lower and upper bounds in the ARRAY statement allows the code to easily handle a string of consecutive PARAMN values that does not begin with 1. The VNAME function is commonly used with array elements to assign the actual name of the underlying variable corresponding to the array element into another variable.

The UNITS array has been designated as a temporary array through the use of the _TEMPORARY_ keyword. Temporary arrays do not contain actual variables- they are simply temporary data elements, and as such, disappear completely at the end of the DATA step. However, they are automatically retained across records, which makes them very useful in this example, where the units for each vital sign remain constant across the entire data set.

TRANSPOSING DATA FROM VERTICAL TO HORIZONTAL

PROC TRANSPOSE can also be used for transposing data from a vertical structure to a horizontal structure, but here, as well, it may not provide the desired output dataset in one step. In that case, arrays can often be used for a more customized transpose.

Suppose we have the following dataset of summary statistics, and we need to transpose them so that the values for each treatment group are in a separate column:

Original dataset:

STATN	STAT	TRTPN	VALUE
1	N	1	100
1	N	2	150
2	Mean	1	27.1
2	Mean	2	28.5
3	SD	1	4.65
3	SD	2	8.64

Desired dataset:

STATN	STAT	COL1	COL2
1	N	100	150
2	Mean	27.1	28.5
3	SD	4.65	8.64

In this case, we can take advantage of arrays, combined with DOW-Loop processing, and transpose the data in a single DATA step.

```
data table (keep=statn stat col1-col2);
  array cols(*) col1-col2;
  do until (last.statn);
    set indata;
    by statn;

    cols(trtpn) = value;
  end;

  output;
run;
```

We can use the SAS shorthand for specifying variable lists in the ARRAY statement, so the individual variables do not need to be listed individually. The DOW-Loop iterates over all records containing the same value of STATN, thus populating all of the array elements before it exits the loop and writes a single record to the output data set.

CONCLUSION

Arrays in SAS are a powerful tool for avoiding repetitive blocks of code, transposing datasets, and performing other tasks which involve processing groups of variables. However, they are implemented differently than in other commonly-used programming languages, and often confuse less-experienced programmers. The key to understanding SAS arrays is to remember that they are only temporary constructs; the arrays themselves disappear at the end of a DATA step, leaving only the variables assigned to those array elements. If the same array is needed in a subsequent DATA step, it must be re-defined in that step.

REFERENCES

Droogendyk, Harry. “Arrays – Data Step Efficiency.” *Proceedings of the SESUG 2011 Conference*. Available at http://analytics.ncsu.edu/sesug/2011/CC17.Droogendyk.pdf.

First, Steve; Schudrowitz, Teresa. “Arrays Made Easy: An Introduction to Arrays and Array Processing.” *Proceedings of the SUGI 30 Conference*. Available at http://www2.sas.com/proceedings/sugi30/242-30.pdf.

SAS Institute. “SAS 9.3 Language Reference: Concepts.” ARRAY Processing. Available at http://support.sas.com/documentation/93/index.html.

SAS Institute. “SAS 9.3 Statements: Reference.” ARRAY Statement. Available at http://support.sas.com/documentation/93/index.html.

Stroupe, Jane. “Adventures in Arrays: A Beginning Tutorial.” *Proceedings of the SCSUG 2007 Conference*. Available at http://www.lexjansen.com/scsug/2007/data/Data-Ravenna.pdf.

ACKNOWLEDGMENTS

Many thanks to my colleagues who reviewed this paper.

CONTACT INFORMATION

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

Nancy Brucken
inVentiv Health
Ann Arbor, MI
(734) 887-0255
Nancy.Brucken@inventivhealth.com