PhUSE 2016

Paper CC07

Add Derived Parameters with Multi-Dimensional Arrays

Siddharth Kumar, GCE Solutions Inc., New Delhi, India

ABSTRACT

Multidimensional arrays are useful when Data needs to be grouped into a 'table like' arrangement for processing. Often we are required to derive and add records in analysis datasets. When there are large numbers of parameters to be added, the use of repeated if-then statements are prone to error and time consuming. Effective use of Multi-Dimensional Arrays or Nested Arrays can increase efficiency of program. It can minimize the code and makes program efficient. This paper demonstrates situation where use of Multi-Dimensional Arrays is practical and appropriate in Clinical Trials.

INTRODUCTION

Creating Laboratory analysis datasets is challenging and daunting task, due to the size of dataset and number of derived parameters required. Creating Lab analysis dataset requires to derive parameters. Common technique used to add records in dataset is the use of if-then statement, or create dummy dataset etc. Use of Nested arrays to derive these parameters limits the chance of error and makes program efficient.

Arrays can be very powerful with its iterative and conditional processing. Array in SAS allows to group a bunch of variables for the same process. The huge block of repetitious statements and redundant calculation codes can be reduced to just a few lines. With arrays you can simplify the coding in many cases and can accomplish tasks which are not easily done.

An array must be defined within the data step prior to being referenced. if an array is referenced without first being defined, an error will occur. As array exist only for the duration of the data step in which they are defined therefore, they can only be used in the same data step where it is referenced.

One Dimensional Arrays

A simple array may be created when the variables grouped together conceptually appear as a single row. This is known as a one-dimensional array.

The array statement to define one-dimensional array will be:

```
array temperature_array {24} temp1 – temp24;
```

The array has 24 elements for the variables TEMP1 through TEMP24. When the array elements are used within the data step the array name and the element number will reference them.

The reference to the ninth element in the temperature array is: temperature_array{9}

The statement used to define an array is the ARRAY statement.

```
array array-name {n}  <$>  <length> array-elements  <(initial-values)>;
```

The Array statements provides the following information about SAS array:

array-name – Any valid SAS name

n – Number of elements within the array

$ - Indicates the elements within the array are character type variables

Length – assigns length for the array elements

Elements – List of SAS variables to be part of the array

Initial values – Provides the initial values for each of the array elements.

In clinical trials arrays are mostly applied in the creation of TLFs. Example of situation where arrays are commonly used are: Creating final format on treatment columns

Multi dimensional Array

Multi dimensional arrays are used when you want to put values in a table format (i.e., rows and columns). Multidimensional arrays can be used to input data or to perform operations on the data set.

Multi dimensional arrays are defined as follows: The number of elements are placed in each dimension after the array name in the form {n, ..}. From right to left, the rightmost dimension represents columns; the next dimension represents rows. Each position farther left represents a higher dimension.

```
array temprg {3, 5} c1t1 c1t2 c1t3 c1t4 c1t5
                    c2t1 c2t2 c2t3 c2t4 c2t5
                    c3t1 c3t2 c3t3 c3t4 c3t5;
```

This is an example of two-dimensional array with three rows and five columns The array contains fifteen variables: five temperature measures (t1 through t5) from three cities (c1, c2 and c3).

SAS places variables into a multidimensional array by filling all rows in order, beginning at the upper left corner of the array (known as row-major order). You can think of the variables as having the following arrangement:

```
c1t1 c1t2 c1t3 c1t4 c1t5
c2t1 c2t2 c2t3 c2t4 c2t5
```

To refer to an element of the array later with an array reference, you can use the array name and subscripts. Always specify the row you want first, and then the column. Separate the two numbers with a comma. The following table lists some of the array references for the previous example:

Array References for array TEMPRG	
Variable	Array Reference
c1t1	temprg{1,1}
c1t2	temprg{1,2}
c2t2	temprg{2,2}
c2t5	temprg{2,5}

In summary, listing the dimensions of arrays works like the following:

One-dimensional array: array x(cols)

Two-dimensional array: array y(rows,cols)

Three-dimensional array: array z(levels, rows, cols)

Using Nested Do Loops

Multidimensional arrays are usually processed inside nested Do loops.

For example, a two-dimensional array can be processed as follows:

```
Do index-variable-1 = 1 to number of rows;
        Do index-variable-2=1 to number of columns;
                more SAS statements …
        End;
End;
```

An array reference can use two or more index variables as the subscript to refer to two or more dimensions of an array. Use the following form:

array-name {*index-variable-1, …*,*index-variable-n*}

The following example creates an array that contains fifteen variables- five temperature measures (t1 through t5) from three cities (c1, c2, c3). The DATA step contains two DO loops.

A do loop is needed for each dimension, thus two do loops are specified below, one for the rows (which is represented by i and set from 1 to 3) and one for the columns (represented by j and set from 1 to 5). Note, if you make i reference the rows (1 to 3), then i should be put in the first position in the array.

The outer DO loop (DO I=1 TO 3) processes the inner DO loop thrice.

The inner DO loop (DO J=1 TO 5) applies the ROUND function to all the variables in one row.

For each iteration of the DO loops, SAS substitutes the value of the array element corresponding to the current values of I and J.

```
options linesize=80 pagesize=60;

data temps;
  array temprg{3,5} c1t1-c1t5 c2t1-c2t5 c3t1-c3t5;
  do i=1 to 3;
    do j=1 to 5;
      temprg{i,j}=round(temprg{i,j});
    end;
  end;
run;
```

The previous example can also use the DIM function to produce the same result:

```
do
i=1 to dim1(temprg);
  do j=1 to dim2(temprg);
    temprg{i,j}=round(temprg{i,j});
  end;
end;
```

The value of DIM1(TEMPRG) is 3; the value of DIM2(TEMPRG) is 5.

APPLICATION IN CLINICAL TRIAL

An example of application of Two-Dimensional arrays to derive parameters in laboratory analysis dataset

PROGRAMMING SPECIFICATIONS

As per the Programming specifications, 24 parameters are required to add in ADLB (lab) dataset. Table 1, is an example of the parameters required.

Table 1, Programming Specifications

	PARAMCD	PARAM
1	ALT3U	ALT > 3 x ULN
2	ALT5U	ALT > 5 x ULN
3	ALT8U	ALT > 8 x ULN
4	ALT10U	ALT > 10 x ULN
5	ALT20U	ALT > 20 x ULN
6.	…	……
24	LG17	ALP > 5 x ULN & TBL > 2 x ULN

USE OF NESTED ARRAYS

Here is an example of use of nested arrays where the parameters that are required as per the programming specifications are derived.

```
data lab;
   set in_lab;

   array _param (4, 5) $200.
('ALT > 3 x ULN'    'ALT > 5 x ULN' 'ALT > 8 x ULN' 'ALT > 10 x ULN'    'ALT > 20 x ULN'
 'AST > 3 x ULN'    'AST > 5 x ULN' 'AST > 8 x ULN' 'AST > 10 x ULN'   'AST > 20 x ULN'
 'TBL > 1.5 x ULN'  'TBL > 2 x ULN' 'TBL > 3 x ULN' ''                 ''
 'ALP > 2 x ULN'    'ALP > 3 x ULN' 'ALP > 5 x ULN' ''                 '');

   array _paramcd (4,5) $8.
('ALT3U' 'ALT5U' 'ALT8U' 'ALT10U' 'ALT20U'
 'AST3U' 'AST5U' 'AST8U' 'AST10U' 'AST20U'
  'TBL1_5U' 'TBL2U' 'TBL3U' ''      ''
  'ALP2U'   'ALP3U' 'ALP5U' ''      '');

   array _ratio_limit(4,5) (3    5 8 10 20
                            3    5 8 10 20
                            1.5  2 3  .  .
                            2    3 5  .  .);

   array _parcat4 (4) $20. ('ALTSI' 'ASTSI' 'BILISI' 'ALPSI');
   array _crit(5)   CRIT1   CRIT2   CRIT3   CRIT4   CRIT5;
   array _critfl(5) CRIT1FL CRIT2FL CRIT3FL CRIT4FL CRIT5FL;
   array _critfn(5) _CRIT1Fn _CRIT2Fn _CRIT3Fn _CRIT4Fn _CRIT5Fn;
   array _critcd(5) $8. _crit1cd _crit2cd _crit3cd _crit4cd _crit5cd;

     if PARAMCD in ('ALTSI','ASTSI','BILISI','ALPSI') then
       do j=1 to 4;
         if paramcd=_parcat4(j) then
           do;
             do i=1 to 5;
               _crit(i)=_param(j,i);
               _critcd(i)=_paramcd(j,i);
               if R2ANRHI>_ratio_limit(j,i)>. then do; _critfl(i)='Y'; _critfn(i)=1; end;
               else if .<R2ANRHI<=_ratio_limit(j,i) then do; _critfl(i)='N'; _critfn(i)=0; end;
             end;
             output;
           end;
run;
```

Benefits:

Use of Multidimensional arrays reduces lines of code, and is easier to maintain. It is less prone to error, the review of code is quicker and the program is efficient.

CONCLUSION

When there are large number of parameters to derive, SAS two-dimensional arrays can provide a great deal of flexibility and increases the efficiency of program.

REFERENCES (HEADER 1)

http://support.sas.com/documentation/

CONTACT INFORMATION (HEADER 1)

(In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)

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

Siddharth Kumar Mogra
GCE Solutions, Inc
DLF Towers, New Delhi
New Delhi 110001
Work Phone: 9727583226
Fax:
Email: Siddharth.kumar@gcesolutions.com
Web: