Paper 158-29

Fun with Fancy Arrays

Mary McDonald, UBS Financial Services, Weehawken, New Jersey

ABSTRACT

Many novice SAS® programmers avoid complicated arrays because they think that they are difficult. The truth is that they make work easier. This paper is intended to be a simple introduction to working with multidimensional and nested arrays. Topics covered are: common applications, loading constants, referencing, defining and processing.

INTRODUCTION

Multidimensional arrays are useful when you want to group your data into a 'table like' arrangement for processing. You will know you need one when you start thinking that you could process your data in EXCEL a lot easier than in SAS, except that you would need a separate spread sheet for each record in your file. Arrays reduce the amount of code that you have to write, make your program look tidier and make it easier to think about what you are doing. However, no array type will improve the computer's processing efficiency.

Referencing and manipulation is not much different than for one-dimensional arrays. Actually the simple array is a special case of the multi-dimensional array. Both can be either implicit or explicit, only the later is discussed in this paper. References to implicit arrays do not appear in the documentation for Version 7 and beyond. The code will still work but it's probably a bad idea to rely on it.

Let's start with the simple numeric array and build on that to understand the more complex code. Its basic elements are as follows:

```
ARRAY array-name (size) array-elements;
```

If you put one record in a spreadsheet the data would look like this:

	COL1	COL2	COL3	COL4
ROW1	240	1000	72	95

Or in SAS with spreadsheet type names:

```
ARRAY SIMPLE (4) R1C1 R1C2 R1C3 R1C4;
```

Adding another dimension just adds additional rows to your table:

	COL1	COL2	COL3	COL4
ROW1	240	1000	72	95
ROW2	341	998	84	83
ROW3	549	800	102	77

```
ARRAY DOUBLE(3,4) R1C1 R1C2 R1C3 R1C4
                  R2C1 R2C2 R2C3 R2C4
                  R3C1 R3C2 R3C3 R3C4
                  R3C1 R3C2 R3C3 R3C4;
```

Try to think of the first dimension as rows, the second as columns. For higher dimensions, just add another comma and the size of the next dimension. This goes in front of the row subscript, which is sort of odd at first. Its logical if you remember that it will be the outer most of the nested loops. SAS 'fills' the array starting with the first dimension. More advanced, 3-d and up arrays are rare, I have never needed one or seen one in 'real world' code.

LEVEL1:

	COL1	COL2	COL3	COL4
ROW1	240	1000	72	95
ROW2	341	998	84	83
ROW3	549	800	102	77

LEVEL2:

	COL1	COL2	COL3	COL4
ROW1	761	1022	78	16
ROW2	341	968	584	583
ROW3	549	780	109	76

```
ARRAY TRIPLE(2,3,4)
          L1R1C1 L1R1C2 L1R1C3 L1R1C4
          L1R2C1 L1R2C2 L1R2C3 L1R2C4
          L1R3C1 L1R3C2 L1R3C3 L1R3C4
          L1R3C1 L1R3C2 L1R3C3 L1R3C4
          L2R1C1 L2R1C2 L2R1C3 L2R1C4
          L2R2C1 L2R2C2 L2R2C3 L2R2C4
          L2R3C1 L2R3C2 L2R3C3 L2R3C4
          L2R3C1 L2R3C2 L2R3C3 L2R3C4;
```

Processing is usually done in nested loops, the first dimension is the outer most and 'columns' are the inner most.

```
DO LEVEL = 1 TO 2;
     [Code to process the triple array data]
     DO ROW =1 TO 3;
        [Code to process the triple array data]
         DO COLUMN 1 TO 4;
          [Code to process the triple array data]
         END;
     END;
END;
```

You do not have to go through the entire array. SAS has code for conditional execution as follows: DO UNTIL (evaluate the condition at the bottom of the loop), DO WHILE (evaluate at the top) and BY (to increment the index by a specified amount). This works exactly the same way on all dimensions of arrays.

Processing with out specifying the size is slightly different. DIM returns the first dimension by default. For 2-d plus specify the number desired. In our example, DIM(TRIPLE) would get the number of levels, DIM(TRIPLE,2) or DIM2(TRIPLE) would get the number of rows, DIM(TRIPLE,3) or DIM3(TRIPLE) the number of columns. Boundary specifications can be added to the subscript. LBOUND AND HBOUND will work just like DIM.

Optional array specification code <$> and <length> for arrays of character variables works exactly the same. <*> Doesn't work because SAS cannot determine the array subscripts by counting the number of elements in multidimensional or _TEMPORARY_ arrays.

WORD TO THE WISE

If you are going to do exactly the same thing to all the array elements you can and should define the array as a one-dimensional array and process inside a single loop. The old KISS, 'keep it simple stupid', rule applies.

```
ARRAY SIMPLE (24)
          L1R1C1 L1R1C2 L1R1C3 L1R1C4
          L1R2C1 L1R2C2 L1R2C3 L1R2C4
          L1R3C1 L1R3C2 L1R3C3 L1R3C4
          L1R3C1 L1R3C2 L1R3C3 L1R3C4
          L2R1C1 L2R1C2 L2R1C3 L2R1C4
          L2R2C1 L2R2C2 L2R2C3 L2R2C4
          L2R3C1 L2R3C2 L2R3C3 L2R3C4
          L2R3C1 L2R3C2 L2R3C3 L2R3C4;
```

Processing would then be done in one loop:

```
DO J= 1 TO 24;
    [Code to process data]
END;
```

Only use complex arrays when you have to do complex processing and they make it simpler to code. This is easier to understand when you work through coding problems. The following examples are based on real applications, however, the data in this paper is just for demonstration purposes.

PROBLEM 1

This is part of a set of programs that process data on leased office space for budgeting purposes. The goal was to project the current costs for rent, utilities and renovation for each of the properties 5 years into the future. It was

assumed that the rate of cost increase would stay the same for all years: 3 percent increases for utilities, 5 percent for renovation and miscellaneous and 8 for rent.

SOLUTION 1

This can be coded with multiple simple arrays as follows:

```
                                        /* PROPERTY COSTS FOR EACH YEAR */
ARRAY BASE  (5) RENT   ELEC   RENOV   HVAC   MISC ;
ARRAY ONE   (5) RENT1  ELEC1  RENOV1  HVAC1  MISC1 ;
ARRAY TW0   (5) RENT2  ELEC2  RENOV2  HVAC2  MISC2 ;
ARRAY THREE (5) RENT3  ELEC3  RENOV3  HVAC3  MISC3 ;
ARRAY FOUR  (5) RENT4  ELEC4  RENOV4  HVAC4  MISC4 ;
ARRAY FIVE  (5) RENT5  ELEC5  RENOV5  HVAC2  MISC2 ;
                                                    /* ANNUAL TOTALS */
ARRAY TOTAL (5) TOTAL1-TOTAL5;
                                      /* ANNUAL INFLATION CONSTANTS */
ARRAY INCREASE (5) INC1-INC5;

RETAIN INC1 1.08 INC2 INC3 1.03 INC4 INC5 1.05;
                                                     /* FIRST YEAR */
DO J = 1 TO 5;
     ONE(J) = BASE(J)*INCREASE(J);
     TOTAL1 + ONE(J);
END;
                                                    /* SECOND YEAR */
DO J = 1 TO 5;
     TWO(J) = ONE(J)*INCREASE(J);
     TOTAL2 + TWO(J);
END;
                                                     /* THIRD YEAR */
DO J = 1 TO 5;
     THREE(J) = TWO(J)*INCREASE(J);
     TOTAL3 + THREE(J) ;
END;
                                                    /* FOURTH YEAR */
DO J = 1 TO 5;
     FOUR(J) = THREE(J)*INCREASE(J);
     TOTAL4 + FOUR(J);
END;
                                                     /* FIFTH YEAR */
DO J = 1 TO 5;
     FIVE(J) = FOUR(J)*INCREASE(J) ;
     TOTAL5 + FIVE(J);
END;

DROP INC1-INC5 J;
```

That code will work and it is certainly a lot better than writing the computation of every cost in every year. However, it is pretty repetitive and can definitely be improved with fancier arrays..

SOLUTION 2

Doing the same calculation with Multidimensional arrays required less code than the first solution and was, I think, easier to understand.

```
ARRAY COSTS (6,5)
    RENT  ELEC  RENOV  HVAC  MISC
    RENT1 ELEC1 RENOV1 HVAC1 MISC1
    RENT2 ELEC2 RENOV2 HVAC2 MISC2
    RENT3 ELEC3 RENOV3 HVAC3 MISC3
    RENT4 ELEC4 RENOV4 HVAC4 MISC4
    RENT5 ELEC5 RENOV5 HVAC5 MISC5;

                                          /* TOTAL ANNUAL COSTS */
ARRAY TOTAL (5) TOTAL1-TOTAL5;

                                      /* ANNUAL INFLATION CONSTANTS */
ARRAY INCREASE (5) _temporary_ 1.08 1.03 1.03 1.05 1.05);

                                                    /* CALCULATION */
DO R = 2 TO 6;
    P = R - 1;
```

```
    DO C = 1 TO 5;
        COSTS(R,C)=COSTS(P)*INCREASE(C);
        TOTAL(R) + COSTS(R,C);
    END;
END;
```

The 'R' loop starts with '2' for the second 'row' which is the first year for which costs are going to be projected. The first 'row' has the current actual costs. The 'P' index variable is used to reference the prior year's row. The 'C' loop moves us across the array to each type of cost. In this problem, the inflation factors were assumed to be different for each type of cost but the same for each year. If they were the same for the cost but different for each year you would use the 'R' index variable which increments for each year.

The annual inflation constants are defined in a _TEMPORARY_ array. This is a more computer efficient way of doing it than the code in the first solution. Notice that I didn't name the variables, SAS will name them and automatically drop them at the end of the step. This was convenient because they were not needed in the dataset. If you had different factors for the years as well as the type of cost you would use a multidimensional array for the inflation factors. It can still be an array of temporary variable that are constants. For more dynamic modeling macros or an external file could be used.

PROBLEM 2

This code was used to calculate bonus amounts, that are paid based on a combination of points earned for opening new accounts and increases in assets in all client accounts. Each broker has an individual set of eight asset levels, but the seven points categories for new accounts are the same for every one. Those who do not meet both asset and new account goals get nothing. There is no way to compute the amounts to be paid other than looking them up on the table. It could be coded using seven separate arrays one for each level of points but there is a neater solution.

The individual asset goals for each broker are read into an array from an ordinary flat file. Its also necessary is to code a value for the points ranges. PROC FORMAT was used for the points ranges in combination with at PUT statement to create the variable because this is more efficient than IF statements.

```
                                                /* POINTS RANGE FORMATS */
PROC FORMAT;
value ptscd
     LOW-<20.00    = '0'
     20.00-<30.00  = '1'
     30.00-<40.00  = '2'
     40.00-<50.00  = '3'
     50.00-<60.00  = '4'
     60.00-<70.00  = '5'
     70.00-<80.00  = '6'
     80.00-high    = '7' ;
                        /* READ EXTERNAL FILE FOR INDIVIDUAL GOALS */
DATA GOALS;
     INFILE GOALS MISSOVER;
     ARRAY GOAL (8);
     INPUT @1 SSN @;
     N = 10;
     DO J=1 TO 8;
        INPUT @N GOAL(J) 12. @;
        N +12;
     END;
```

Because I did not list the variables in the GOAL ARRAY, SAS will name them GOAL1 to GOAL8 for me. It is usually better to list them so you control what their names are. In this particular case I only need them to do the calculation. That is in the next step so the array is not temporary. They will have to be explicitly dropped after I am done with them.

```
DATA FATOTALS;
     MERGE FATOTALS(IN=F) GOALS(IN=G);
     BY SSN;
     IF F AND G;
                                                    /* BONUS AMOUNTS */
   ARRAY BONUS (0:7,0:8) _temporary_ (0    0    0    0    0    0    0    0    0
                                      0  100  150  200  250  300  350  400  450
                                      0  175  225  275  325  387  450  512  575
                                      0  250  300  350  400  475  550  625  700
                                      0  325  375  425  475  550  625  700  775
                                      0  400  450  500  550  625  700  775  850
                                      0  550  600  650  700  800  900 1000 1100);
```

The amount of the bonus at each achievement level is the same for all brokers and is not going to change, i.e. they are constants. Therefore, the values of the array elements could be specified in the array itself. I defined the bounds of the array subscripts to start at '0' rather than '1', which is actually is the default. In this case, it happens to be conceptually easier for me to code the asset levels. SAS says it works more efficiently if DO loops and subscripts start at '0', but usually programmer's minds don't work that way. I am also going to process the goal array 'backwards' stopping at the highest level met.

```
                                                    /* ASSET GOALS */
ARRAY GOAL (8)   GOAL1-GOAL8;
                                                  /* INITIAL LEVEL */
LEVEL = 0;
                                                  /* COMPUTE LEVEL */
DO J=8 TO 1 BY -1 UNTIL(LEVEL=J);
    IF ASSETS GE GOAL(J) THEN LEVEL=J;
END;
```

Once I know what the goal level is, I used the format to determine the points category. These two numbers become the array subscripts that let me 'look up' the bonus amount in the 'table'.

```
                                                /* FIND POINTS RANGE */
PTS_CAT = PUT(POINTS,PTSCD.)*1;
                                                       /* FIND BONUS */
BONUS = BONUS(PTS_CAT,LEVEL);

DROP GOAL1-GOAL8 J ;
RUN;
```

At the end, don't forget to drop all of the variables you don't need particularly the array index vars.

CONCLUSION

Multidimensional array applications may not occur frequently, but when you need one it really helps to have that tool available in your SAS programmer skill set. It may seem daunting at first, but after you get used to them they are not really different from regular arrays. Some programmers start by working out the code with no arrays, than simple arrays and then, if warranted, more complexity. Bob Virgil once said that "In most applications which use arrays, arrays are 10% of the pie and other tools make up 90%." I would guess that of that 10%, 90% are one-dimensional simple arrays, but that other 10% has all the fun!

REFERENCES

Bob Virgil, "Introduction to Arrays," *Proceedings of NESUG Annual Conference,* 1997, 288-292.

SAS Institute Inc.(2003) *SAS Online Doc Version 8*, Cary, NC:SAS Institute Inc.

CONTACT INFORMATION

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

Mary McDonald
Database Marketing and Analysis
UBS Financial Services Incorporated
1200 Harbor Blvd, 4th floor
Weehawken, NJ 07087-6791
mary.a.mcdonaldl@ubs.com™®