Arrays by example

Diana Suhr, University of Northern Colorado, Greeley, CO

ABSTRACT

Get ready to master array syntax and discover how data step arrays promote efficient programming techniques. Learn how to use implicit/explicit array statements, simple and multidimensional arrays, and arrays for input statements, initializing values, recoding, and calculations. Expand on array syntax with _NUMERIC_, _CHARACTER_, _TEMPORARY_, DO, and %LET. The power of array statements will be illustrated with simple to complex examples.

Array Definitions

According to the Merriam Webster Dictionary Online, an array is a regular and imposing grouping or arrangement. SAS Online Docs, defines an array as a set of variables put into a temporary group for the duration of the DATA step, a set of elements you plan to process as a group (*SAS®9.1.2 OnlineDoc, 2004).*

An array-name
- identifies the array
- is a name you choose
- distinguishes it from other arrays in the same DATA step
- is different from a SAS function name
- is not a variable.

An array statement
- defines a set of variables (all numeric or all character) as elements of an array
- is a convenient way of temporarily identifying a group of variables
- exists for the duration of the DATA step
- is not a data structure.

An array subscript
- describes the number of elements in the array
- describes the arrangement of elements in the array
- specifies a lower and upper bound for array dimensions
- is designated with an asterisk, number, or range of numbers
- is enclosed in braces { }, brackets [ ], or parentheses ( ).

Array elements
- must be all numeric or all character
- allow you to reference variables in the data step
- can be assigned initial values directly in the array statement.

IMPLICIT ARRAY

An implicit array
- Is “quick & dirty”
- does not contain an explicit specification for the number of elements
- references elements by evaluating the current value of an index variable associated with the array.
- is defined with
  - an array name
  - an index variable (one you supply or a default)
  - a list of names (variables or other implicit arrays)
- does not contain an explicit reference to the number of elements in the array
- is often referenced with DO, DO OVER, DO WHILE, or DO UNTIL..

SAS implicit array syntax is
```
   ARRAY array-name <(index variable)> <$> <length> array-elements <(initial values)>;
```

IMPLICIT ARRAY Examples

Recoding Values

To recode missing values, use IF…THEN statements or an implicit array.

```
if q1 = 0 then q1 = .;
if q2 = 0 then q2 = .;
if q3 = 0 then q3 = .;
. . .
if q98 = 0 then q98 = .;
if q99 = 0 then q99 = .;
if q100 = 0 then q100 = .;
```

OR

```
array mis q1-q100;
 do over mis;
  if mis = 0 then mis=.;
 end;
```

DEMO: Recoding Values

You have entered a value of zero (0) for no response or a missing value. Include zero in a frequency table. Exclude zero when calculating an average.

```
data rawsub;
   infile datalines;
   input q1-q10;
datalines;
1 0 3 4 5 0 2 3 4 5
2 3 0 5 1 2 0 4 5 1
3 4 5 0 2 3 4 0 1 2
4 5 1 2 0 4 5 1 0 3
4 0 2 3 4 0 1 2 3 0
;;;;
data rawsub1;
  set rawsub;
array mis q1-q10;
 do over mis;
  if mis=0 then mis=.;
 end;
 proc print data=rawsub;
 proc print data=rawsub1;
 proc freq data=rawsub;
    tables q1-q4;
proc freq data=rawsub1;
   tables q1-q4;
proc means data=rawsub;
   var q1-q10;
proc means data=rawsub1;
   var q1-q10;
run;
```

Reversing Values

To reverse values for variables on a scale of 1 to 5, use the following statements.

```
array rev x1-x20;
  do over rev;
    rev = 6 – rev;
  end;
```

In the above example, if the value of the variable is

1, then rev = 6 – 1 = 5
2, then rev = 6 – 2 = 4
3, then rev = 6 – 3 = 3
4, then rev = 6 – 4 = 2
5, then rev = 6 – 5 = 1

To reverse *selected* variables on a 1-5 scale, try this code

```
 array rev x1 x5 x6 x11 x14 x18;
  do over rev;
    rev = 6 – rev;
  end;
```

To reverse values on a

1-7 scale, use a rev=8-rev statement.
1-4 scale, use a rev=5-rev statement
0-4 scale use a rev= 4-rev statement.
–3, -2, -1, 0, 1, 2, 3 scale, use a rev = -1 * rev statement.

Try substituting the values 0, 1, 2, 3, 4 into the rev=4-rev statement to verify that the values will be reversed on a 0-4 scale.

You may be asking, “why reverse values?”. Questions on a scale may be worded in a positive or negative direction. For example, “Mathematics is difficult for me” and “Solving math problems is easy for me” are worded differently. A respondent could disagree with one statement and agree with the other statement so you want scales in the same direction.

DEMO: Recoding and Reversing Values

```
data rawsub;
  infile datalines;
  input q1-q10;
 array mis q1-q10;
 array rev r1-r10;
  do over mis;
   if mis=0 then mis=.;
   rev=6-mis;
  end;
datalines;
1 0 3 4 5 0 2 3 4 5
2 3 0 5 1 2 0 4 5 1
3 4 5 0 2 3 4 0 1 2
4 5 1 2 0 4 5 1 0 3
4 0 2 3 4 0 1 2 3 0
;;;;
proc print data=rawsub;
run;

data rawsub2;
  infile datalines;
  input q1-q10;
 array qes q1-q10;
 array rev r1-r10;
 do over qes;
   rev=5-qes;
  end;
datalines;
1 0 3 4 5 0 2 3 4 5
2 3 0 5 1 2 0 4 5 1
3 4 5 0 2 3 4 0 1 2
4 5 1 2 0 4 5 1 0 3
4 0 2 3 4 0 1 2 3 0
;;;;
proc print data=rawsub2;
run;
```

EXPLICIT ARRAY

An explicit array

- specifies the number of elements in the array
- could denote bounds on a multidimensional array
- is more powerful and flexible than an implicit array.
- Is defined with
  - an array name
  - a subscript indicating the number of elements in the array
  - a description of the elements.

SAS explicit array syntax is

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

EXPLICIT ARRAY Examples

```
array sales(4) dept month year amount;
```

This array statement tells the SAS system

- to make a group named "sales" for the duration of the DATA step
- the array has 4 elements: dept month year amount
- to assign each variable an additional variable name using the position in the array. The additional variable name includes an array reference.

The array sales contains 4 elements, dept is assigned the array reference sales(1), month is assigned sales(2), year is assigned sales(3), and amount is assigned sales(4). After defining the array in the DATA step, variables can be referenced with their original variable names or array references. In this example, dept and sales(1) are equivalent.

The array subscript could be indicated explicitly as (4) or the SAS System will determine the number of elements in the array. List the variables and use (*) as the subscript.

```
    array sales(4) dept month year amount;
OR  array sales(*) dept month year amount;
```

In the following example, SAS determines the array has 12 elements and finds the sum.

```
array sales(*) amt1-amt12;
annual_sales = sum(of amt (*));
```

Defining Arrays

Variable list names

- reference previously defined variables in the same DATA step
- are _CHARACTER_, _NUMERIC_, or _ALL_
- use _CHARACTER_ for character values
- use _NUMERIC_ for numeric values
- or _ALL_ for all numeric or character values, depending on how the variables were previously defined.

```
data rawsub;
 input (d1-d7) ($3.)  s1-s7;
 array days (*) _CHARACTER_;
 array sales(*) _NUMERIC_,;
```

In the DATA step rawsub, d1-d7 are defined as character variables and s1-s7 as numeric variables. The array days reads previously defined character variables while the array sales reads previously defined numeric variables. The SAS system determines the size of the array when the * is used as the array subscript.

Be aware of data step variables and make sure you are using the variables you intend to process with the array statements. Add a statement to calculate total sales, and the array sales has 8 elements rather than 7.

```
data rawsub;
 input (d1-d7) ($3.)  s1-s7;
 totsales=sum(of s1--s7);
 array days (*) _CHARACTER_;
 array sales(*) _NUMERIC_,;
```

DEMO: Defining Arrays

```
data rawsub;
input (d1-d7) ($)  s1-s7;
array days (*) _CHARACTER_;
array sales(*) _NUMERIC_;
  file print;
  put / 'day' '    ' 'daily' '    ' 'weekly';
   subtot=0;
   do I = 1 to 7;
    subtot= subtot + sales(I);
    put days(I) ' ' sales(I) comma6. '  ' subtot comma6.;
   end;
cards;
mon tue wed thu fri sat sun 1005 1267 943 1110 2930 3098 2056
mon tue wed thu fri sat sun 1126 1314 885 1020 2432 2987 2945
;;;;
proc print data=rawsub;

data rawsub1;
set rawsub;
totsales=sum(of s1--s7);
array days (*) _CHARACTER_;
array sales(*) _NUMERIC_;
   file print;
   do J = 1 to 8;
    put sales(J) ' ' @@;
   end;
   put;
proc print data=rawsub1;
run;
```

DIM function

With an iterative DO statement, the DIM function returns the number of elements in a one-dimensional array. The DIM function could be used to avoid changing the upper bound in a DO loop each time the number of elements in the array changes.

```
data rawsub;
  input s1-s30;
  array sales(*) s1-s30;
   do J = 1 to dim(sales);
     if  sales(J)  > 20000 then output;
   end;
```

DEMO: DIM Function

```
data rawsub;
input (d1-d7) ($)  s1-s7;
array days (*) _CHARACTER_;
array sales(*) _NUMERIC_;
  file print;
  put / 'day' '    ' 'daily' '    ' 'weekly';
   subtot=0;
   do I = 1 to dim(sales);
    subtot= subtot + sales(I);
    put days(I) ' ' sales(I) comma6. '  ' subtot comma6.;
   end;
cards;
mon tue wed thu fri sat sun 1005 1267 943 1110 2930 3098 2056
mon tue wed thu fri sat sun 1126 1314 885 1020 2432 2987 2945
;;;;
proc print data=rawsub;
data rawsub1;
set rawsub;
totsales=sum(of s1--s7);
array days (*) _CHARACTER_;
array sales(*) _NUMERIC_;
   file print;
   do J = 1 to dim(sales);
    put sales(J) ' ' @@;
   end;
   put;
proc print data=rawsub1;
```

Setting Initial Values
Array elements could be Initialized to numeric values. In the array scr initial values are s1=20, s2=40, s3=50, and s4=70.

```
        array scr(4) s1 s2 s3 s4 (20 40 50 70);
```

Array elements could be initialized to character values. Array char initial values are c1='x', c2='y', c3='z'.

```
        array char(3) $ c1 c2 c3 ('x','y','z');
```

Assigning Variable Names
The array test creates an array with 4 elements named test1, test2, test3, test4.

```
        array test(4);
```

DEMO: Initial Values and Assigning Variable Names

```
        data rawsub;
          array scr(4) s1 s2 s3 s4 (20 40 50 70);
          array chr(3) $ c1 c2 c3 ('x','y','z');
          array test(4);
        proc print data=rawsub;
        run;

        data rawsub2;
          array scr(4) s1 s2 s3 s4 (20 40 50 70);
          array chr(3) $ c1 c2 c3 ('x','y','z');
          array test(4)(17, 32, 44, 59);
        proc print data=rawsub2;
        run;
```

Using %Let in Array Processing
To use the same array in more than one DATA step, use a MACRO variable, %LET. The MACRO variable stores the variable names you want to use.

```
        %LET dlist = d1-d7;
        data day1;
          input (d1-d7) ($3.) s1-s7;
          array days(*) &dlist;
        data day2;
          array days(*) &dlist;
```

Counting Responses
Arrays could be used to count responses.

```
        data rawsub;
          input id r1-r10;
          count = 0;
          array resp $ r1-r10;
            do over resp;
              if resp eq 'YES' then count=count+1;
            end;
```

Selecting Responses
Arrays could be used to select observations for data checking.

```
        data rawsub;
          input id d1-d120;
          array resp d1-d120;
          do over resp;
             if  resp lt 0 or resp gt 100 then output;
          end;
        proc sort data=rawsub;
           by id;
        data chkfl;
           set rawsub;
           by id;
           if last.id;
        proc print data=chkfl;
```

DO Processing
Arrays are often processed with iterative DO loops, DO J = …, DO WHILE, or DO UNTIL. Examples are
```
do J = 1 to 10;
do M = 1 to 90 by 3;
do until sales gt 20000;
do K = 1 to 30 while (charge < 500);
```

A step or “by” in the DO loop allows customized processing.
```
DATA ALL;
 INPUT M1-M10;
  ARRAY ALM(10) M1-M10;
  ARRAY ALD(10) D1-D10;
  ARRAY ALT(10) T1-T10;
  **double every other variable;
   DO I = 1 TO 10 BY 2;
     ALD(I) = ALM(I) * 2;
   END;
  **triple every third variable;
   DO J = 1 TO 10 BY 3;
     ALT(J) = ALM(J) * 3;
   END;
```

DEMO: DO Processing and Generating Random Numbers
Arrays can be used to generate 100 random numbers, round the random numbers to integers, and print a table, with the following code.
```
data rawsub;
array ala(100) r1-r100;
do I = 1 to 100;
 ala(I) = round((ranuni(0) * 100), 1);
end;
format r1-r100 z3.;
file print;
do I = 1 to 100 by 10;
 put ala(I)   ' ' ala(I+1) ' '
     ala(I+2) ' ' ala(I+3) ' '
     ala(I+4) ' ' ala(I+5) ' '
     ala(I+6) ' ' ala(I+7) ' '
     ala(I+8) ' ' ala(I+9) ' ';
  end;
run;
```

Temporary Arrays
If elements of an array are only needed during the duration of the DATA step, a temporary array can be used. Temporary array elements
- do not have names
- do not appear in the output data set
- are referred to by the array name and subscript
- are automatically retained
- are not reset to missing at the beginning of each DATA step iteration

Items on a test may be scored using a _TEMPORARY_ array.
```
data rawsub;
input r1-r20;
array key(20) _TEMPORARY_ (1 4 3 5 2 2 4 4 3 5 1 1 3 4 2 2 5 3 4 2);**key;
array resp(20) r1-r20;                              **responses;
array scr(20) s1-s20;                    **score (0=incorrect, 1=correct);
  do J=1 to 20;
     scr(J)=0;                           **initialize, set array values to zero;
   end;
```

```
      do K = 1 to 50;
        if key(K)=resp(K) then scr(K)=1;
      end;
    rawscr=sum(of s1--s50);
  proc print data=rawsub;
```

Multi-Dimensional Arrays
A set of variables can be specified as a one- or two--dimensional array with

```
array yr (60:99)   y60-y99;  OR  array yrs (6:9,0:9) y60-y99;
```

The array resp is a two-dimensional array with 2 rows and 5 columns. The rightmost dimension represents columns. Moving right to left, the next dimension represents rows. Each position farther left represents a higher dimension. In this example, there are 10 elements in the array (2*5) which might represent clinical trials for 2 groups measured at 5 different times.

```
array resp(2,5) r1c1-r1c5 r2c1-r2c5;
```

The elements are arranged in the table as follows

```
r1c1   r1c2   r1c3   r1c4   r1c5
r2c1   r2c2   r2c3   r2c4   r2c5
```

A two-dimensional array statement lists row and column dimensions. SAS assigns variables to array locations by filling rows in order across columns. For example,

```
array yrs(2,5) x1-10;  or  array yrs(1:2, 1:5) x1-x10;
```

assigns x1 to yrs(1,1), x2 to yrs(1,2), x3 to yrs(1,3), x4 to yrs(1,4), and x5 to yrs(1,5) in row 1 and columns 1-5. In row 2, x6-x10 are assigned to columns 1-5. The lower bound of the array dimension is conveniently set to 1 and does not have to be specified.

DEMO: Multidimentional Arrays

```
data rawsub;
array yr(2,5) x1-x10 (1,2,3,4,5,1,2,3,4,5);
array yrs(1:2, 1:5) y1-y10 (11,12,13,14,15,21,22,23,24,25);
file print;
 do I = 1 to 2;
  do J = 1 to 5;
    put 'I=' I 'J=' J 'yr=' yr(I,J) 'yrs=' yrs(I,J);
  end;
put;
  end;
  do J = 1 to 5;
  do I = 1 to 2;
    put 'I=' I 'J=' J 'yr=' yr(I,J) 'yrs=' yrs(I,J);
  end;
put;
  end;
proc print data=rawsub;
run;
```

Input Statements
Array names can be used in the input statement.

```
data names;
  infile rawin delimiter='*';
  length dept1-dept75 sn1-sn75 $10 desc1-desc75 $40;
  array ala(75) $ dept1-dept75;  **deptname;
  array alb(75) $ sn1-sn2;       **short deptname;
  array alc(75)   code1-code75;  **product code;
  array ald(75) $ desc1-desc75;  **product description;
  array ale(75)   amt1-amt75;    **product cost;
  do i = 1 to 75;
    input #i  ala(i) $ alb(i) $ alc(i) ald(i) $ ale(i);
  end;
```

Transforming Data Structure
Arrays can be used to reshape variables into observations.

```
data testfl;
   array pc(5) pc1-pc5;
   set rawfl;
  do i = 1 to 5;
      physchar = pc(i);
      output;
  end;
drop pc1-pc5;
```

Arrays can be used to create a list of means and sort them by magnitude, transforming several variables into one.

```
proc means data = rawsub;
  var r1-r10;
  output out=newmn
        mean=m1-m10 n=n1-n10 std=s1-s10;
data mnfl;
  set newmn;
  array alm(10) m1-m10;
  array als(10) s1-s10;
  array aln(10) n1-n10;
  do i=1 to 10;
    mn=alm(i);
    sd=als(i);
    num=aln(i);
    ques=i;
   output;
  end;
keep mn ques num sd;
proc sort data = mnfl;
  by mn;
proc print data = mnfl;
  var ques mn sd num;
```

CONCLUSION

You have learned, with examples and demos, how DATA step arrays promote efficient programming techniques. You know how to use implicit, explicit, simple, and multidimensional arrays as well as _NUMERIC_, _CHARACTER_, _ALL_, _TEMPORARY_, DO, DIM, and %LET. You have learned techniques to make your programming effortless and efficient. Arrays by example you have experienced the power and flexibility within your control with SAS.

REFERENCES

Merriam-Webster, Incorporated. (2004), *Merriam-Webster Online Dictionary*, Merriam-Webster, Incorporated (http://www.m-w.com/).

SAS Institute, Inc. (1980), *SAS® Applications Guide, 1980 Edition*, Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990), *SAS® Language, Version 6*, Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1999*), SAS® OnlineDoc, Version 8*, Cary, NC: SAS Institute, Inc. (http://v8doc.sas.com/sashtml/).

SAS Institute, Inc. (2004*), SAS®9.1.2 OnlineDoc*, Cary, NC: SAS Institute, Inc. (http://support.sas.com/documentation/onlinedoc/sas9doc.html).

SAS Institute, Inc. (1989), *SAS® Language and Procedures, Version 6, First Edition*, Cary, NC: SAS Institute, Inc.

Virgile, Robert. (1996), *An Array of Challenges: Test Your SAS Skills*, Cary, NC: SAS Institute, Inc.

ABOUT THE AUTHOR

Diana Suhr is a Statistical Analyst in the Office of Institutional Research at the University of Northern Colorado. She holds a Ph.D. in Educational Psychology with an Emphasis in Research, Measurement, and Statistics. She has been a SAS programmer since 1984.

CONTACT INFORMATION

Diana Suhr, Statistical Analyst
Institutional Research
University of Northern Colorado
Greeley, CO 80639
970-351-2193, diana.suhr@unco.edu