

Adventures in Arrays: A Beginning Tutorial

Jane Stroupe, SAS Institute, Chicago, IL

ABSTRACT

Do you need to

- perform repetitive calculations on a group of variables
- create many variables with the same attributes
- restructure your data
- perform a table lookup with one or more factors?

If your answer is "YES" to any of these questions, you should consider using an array in your DATA step.

INTRODUCTION

In SAS®, an array is simply a way to refer to a group of variables in one observation with a single name. These variables must all be the same type.

Think of an array as being an alias for the names of SAS variables. The variables can be

- existing variables
- new variables
- temporary "variables".

This tutorial explores the many uses of an array by using examples involving vacation data, collected from ten fictitious vacation resorts. Let's explore the costs and benefits of a vacation at each of these locations.

Note that these examples are using long names, a Nashville Release enhancement to SAS.

DATA: SASDATA.EXPENSES stores off-season rates from ten resorts for room, food, and various activities.

Resort Name	Resort	Expense1		Expense5	Expense6
Joe's Pretty Good Resort	HOTEL1	\$165.89	...	\$12.00	\$35.00
Larry, Curly, and Motel	HOTEL2	\$215.32	...	\$18.00	\$30.00
Sand And Shores	HOTEL3	\$178.90	...	\$25.00	\$29.99
Array of Sun Hotel	HOTEL4	\$210.78	...	\$17.00	\$24.00
Report Resort	HOTEL5	\$189.87	...	\$38.00	\$42.00
NoTell Hotel	HOTEL6	\$312.15	...	\$14.00	\$19.40
Dew Drop Inn	HOTEL7	\$197.12	...	\$26.00	\$18.95
Greatest View Resort	HOTEL8	\$240.98	...	\$12.00	\$32.00
Holy Cow Hotel	HOTEL9	\$312.10	...	\$15.00	\$26.00
Come On Inn	HOTEL10	\$187.98	...	\$24.00	\$29.65

EXAMPLE 1: Using a one-dimensional array to perform repetitive calculations.

TASK: You want a SAS data set with both the off-season and seasonal rates which are 25% higher than off-season.

SOLUTIONS:

- Use six assignment statements
- Use an array.

PROGRAM:

The first solution requires considerable typing:

```
data work.SeasonRates;  
  set sasdata.expenses(drop=ResortName);  
  seasonal1 = expense1*1.25;  
  seasonal2 = expense2*1.25;  
  seasonal3 = expense3*1.25;  
  seasonal4 = expense4*1.25;  
  seasonal5 = expense5*1.25;  
  seasonal6 = expense6*1.25;  
run;
```

To solve this problem using an array, you must first establish the correspondence between

- an array name and the existing variables, expense1 - expense6
- an array name and the new variables, seasonal1 - seasonal6.

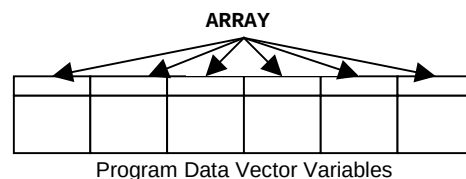
Use array statements to set up the correspondences:

```
array ex{6} expense1 - expense6;  
  
array seasonal{6} seasonal1 - seasonal6;
```

Diagram illustrating the correspondence between array names and variables:

- ARRAY NAME** points to `ex` and `seasonal`.
- DIMENSION** points to `{6}` in both array statements.
- VARIABLES (often called ELEMENTS)** points to `expense1 - expense6` and `seasonal1 - seasonal6`.

The ARRAY statement is a compile time statement that creates the correspondence between the array name and, in this example, the variables in the Program Data Vector.



The general form of a simple ARRAY statement

ARRAY	<i>array-name</i> { <i>n</i> } \$ <i>length</i> <i>elements</i> ;
<i>array-name</i>	any valid SAS name (avoid function names)
<i>n</i>	number of elements to which the array refers
\$	indication that the array refers to character data
<i>length</i>	the length for new character variables. The default is 8.
<i>elements</i>	the variables in the data to which the array refers. If the variables do not exist, SAS will create them.

Once you establish this correspondence, you can use a DO loop to generate the six assignment statements that calculate the weekly total.

```
do i=1 to 6;
    seasonal{i} = ex{i}*1.25;
end;
```

Putting this all together in a second version of the data step:

```
data work.SeasonRates(drop=i);
  ❶ array ex{6} expense1 - expense6;
  ❷ array seasonal{6} seasonal1 -
    seasonal6;
    set sasdata.expenses
      (drop=ResortName);
  ❸ do i=1 to 6;
      seasonal{i} = ex{i}*1.25;
    end;
  ❹ format expense1 - expense6
    seasonal1 - seasonal6
    dollar9.2;
run;
```

- ❶ To use an array, you must first declare it in an ARRAY statement. The ARRAY statement establishes a correspondence between the name of the array, EX, and the variables to which it refers, EXPENSE1 - EXPENSE6.

The association of an array reference with a variable is only for the duration of the DATA step. If you need to refer to the array in another DATA step, you must repeat the array reference in that DATA step.

- ❷ The array, SEASONAL, references variables that do not exist. SAS creates the variables SEASONAL1 - SEASONAL6 from the ARRAY statement.
- ❸ The DO loop processes each element of the array. The assignment statement using the array reference generates six assignment statements that calculate seasonal rates for each of the six expenses.
- ❹ The FORMAT, LABEL, DROP, KEEP, or LENGTH statements must use variable names, not array references.

The resulting data:

expense1		expense6	seasonal1		seasonal6	Resort
\$165.89	...	\$35.00	\$207.36	...	\$43.75	HOTEL1
\$215.32	...	\$30.00	\$269.15	...	\$37.50	HOTEL2
\$178.90	...	\$29.99	\$223.63	...	\$37.49	HOTEL3
\$210.78	...	\$24.00	\$263.48	...	\$30.00	HOTEL4
\$189.87	...	\$42.00	\$237.34	...	\$52.50	HOTEL5
\$312.15	...	\$19.40	\$390.19	...	\$24.25	HOTEL6
\$197.12	...	\$18.95	\$246.40	...	\$23.69	HOTEL7
\$240.98	...	\$32.00	\$301.23	...	\$40.00	HOTEL8
\$312.10	...	\$26.00	\$390.13	...	\$32.50	HOTEL9
\$187.98	...	\$29.65	\$234.98	...	\$37.06	HOTEL10

Valid Syntax for the ARRAY statement:

1. The array name can be different from the variable list root word as long as there is no variable named the root word. You should, however, avoid function names.

For example,

```
array expense{6} expense1 - expense6;
or
array expense{6};
```

are fine because there was not a variable named EXPENSE in the data set.

2. The variable names do not have to be a variable list.

If the data set variables were named RoomRate, DailyFood, SpaVisits, RoundOfGolf HorseBackRiding, and ParkAdmission, you could use:

```
array expense{6} RoomRate DailyFood
    SpaVisits
    RoundOfGolf
    HorseBackRiding
    ParkAdmission;
```

You can use the double dash to refer to this list of variables, if the variables are all contiguous in the SAS data set.

```
array expense{6} RoomRate --
    ParkAdmission;
```

You could refer to the variable list, just not in order.

```
array expense{6} expense6 expense3
    expense2 expense4
    expense5 expense1;
```

Since EXPENSE1 - EXPENSE6 are all numeric variables, you can use the key word `_NUMERIC_` to refer to them. You can use the `_NUMERIC_` keyword only for variables currently defined in the PDV.

For example:

```
array expense{6} _numeric_;
```

For the new variables, you can name the variables with a list that has a different root word.

For example:

```

array seasonalexpense{6} seasonal1 -
                        seasonal6;

or

array seasonal{6} sRoomRate sDailyFood
                  sSpaVisits
                  sRoundOfGolf
                  sHorseBackRiding
                  sParkAdmission;

or

array seasonalexpense{6} seasonal6
                        seasonal4
                        seasonal2
                        seasonal1
                        seasonal5
                        seasonal3;

```

You could not use the double dash to refer to new variable names.

- The ARRAY statement can automatically name new variables.
For example,
array seasonal{6};
would automatically create variables SEASONAL1 - SEASONAL6.
- You do not have to specify the dimensions with a number as we did in the array statements.

However, during compile time, SAS has to know what the dimensions are. You can use a macro variable to specify the dimensions.

For example:

```

%let number=6;
array ex{&number} expense1 - expense6;

```

If appropriate, you can use a range of values for the dimensions.

For example:

```

array ex{94:99} expense94-
                expense99;

```

and use the do loop:

```

do i=94 to 99;
    SAS statements
end;

```

Sometimes, you can use the * to specify the dimensions and the DIM function in your DO loop.

For example:

```

array ex{*} expense1 - expense6;
and use the do loop:
do i=1 to dim(ex);
    SAS statements
end;

```

If you use the *, you must specify the elements (variables).

EXAMPLE 2: USING A ONE-DIMENSIONAL ARRAY TO PERFORM TABLE LOOKUP.

TASK: You have budgeted the following amounts for each of the daily expenses: \$175 for expense1 (room), \$75 for expense2 (food), \$25 for expense3 (spa treatments), \$35 for expense4 (a round of golf), \$25 for expense5 (horse back riding), and \$30 for expense6 (theme park admission). Use an array to assign these budget amounts as initial values and determine the difference between the budgeted amounts and the actual amounts.

SOLUTIONS:

- Use six assignment statements to calculate the differences.
- Use an array.

PROGRAM

```

data work.diffs;
    drop i;
    set sasdata.expenses
        (drop=ResortName);
    ❶ array budget{6} _temporary_
        (175,75,25,35,25,30);
    array expense{*} expense1 - expense6;
    array diff{6};
    ❷ do i=1 to dim(expense);
        diff{i} = budget{i} - expense{i};
    end;
run;

```

- You can assign initial values for the corresponding elements in the array by putting these initial values within parenthesis and separated by either a comma or a blank. Initial values can be assigned to both variables and temporary data elements.

In the ARRAY statement, the keyword **_TEMPORARY_** creates a list of temporary data elements that can be either numeric or character. The temporary elements must be used in a DATA step and are automatically retained for each iteration of the DATA step. You must refer to these elements by the array name and dimension since they do not have names, and you cannot use the asterisk (*) to refer to all the elements.

Arrays of temporary elements are useful when the only purpose for creating an array is to perform a calculation (often referred to as performing a table lookup). You can improve performance time by using temporary array references.

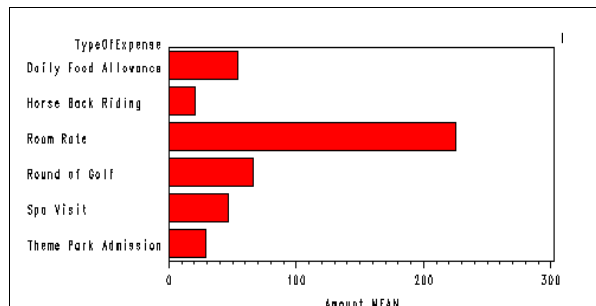
- You can use the DIM function to return the dimensions of the array EXPENSE.

The resulting data:

Resort	expense1	...	expense6	DIFF1	...	DIFF5	DIFF6
HOTEL1	\$165.89	...	\$35.00	9.11	...	13.00	-5.00
HOTEL2	\$215.32	...	\$30.00	-40.32	...	7.00	0.00
HOTEL3	\$178.90	...	\$29.99	-3.90	...	0.00	0.01
HOTEL4	\$210.78	...	\$24.00	-35.78	...	8.00	6.00
HOTEL5	\$189.87	...	\$42.00	-14.87	...	-13.00	-12.00
HOTEL6	\$312.15	...	\$19.40	-137.15	...	11.00	10.60
HOTEL7	\$197.12	...	\$18.95	-22.12	...	-1.00	11.05
HOTEL8	\$240.98	...	\$32.00	-65.98	...	13.00	-2.00
HOTEL9	\$312.10	...	\$26.00	-137.10	...	10.00	4.00
HOTEL10	\$187.98	...	\$29.65	-12.98	...	1.00	0.35

EXAMPLE 3: USING A ONE-DIMENSIONAL ARRAY TO RESTRUCTURE YOUR DATA AND PERFORM A TABLE LOOKUP WITH CHARACTER DATA.

TASK: In order to create the following graph,



you need to have the data in the form of

Expense	TypeOfExpense	Expenses
1	Room Rate	\$165.29
2	Daily Food Allowance	\$45.50
3	Round of Golf	\$72.00
4	Spa Visit	\$45.00
5	Horse Back Riding	\$12.00
6	Theme Park Admission	\$35.00
7	Room Rate	\$215.32
.	.	.
.	.	.

SOLUTIONS:

- Use complicated logic to rotate the data then use 6 assignment statements to create a variable that is the name of the expense.
- Use PROC TRANSPOSE to rotate the data then a data step with assignment statements to create a variable that is the name of the expense.
- Use an array.

PROGRAM

```

data work.rotate;
  keep resort expense amount
  TypeOfExpense;
  set sasdata.expenses;
  ❶ array ex{*} _numeric_;
  ❷ array name{6} $ 32
    ('Room Rate'
     'Daily Food
      Allowance'
     'Round of Golf'
     'Spa Visit'
     'Horse Back Riding'
     'Theme Park
      Admission');
  ❸ do Expense = lbound(ex) to
    hbound(ex);
  ❹   Amount = ex{Expense};
  ❺   TypeOfExpense = name{Expense};
  ❻   output;
  end;
run;

```

- ❶ You can use the following SAS variables to reference variables that have been previously defined in the same DATA step:

NUMERIC indicates all numeric variables.
CHARACTER indicates all character variables.
ALL indicates all variables. For arrays, all the previously defined variables must be of the same type.

In this case NUMERIC refers to the variables EXPENSE1 – EXPENSE6. The SET statement must come before the ARRAY statement in order to refer to these variables with the NUMERIC keyword.

- ❷ Since the array NAME refers to character values, the ARRAY statement uses the \$ to indicate character data and a length, 32. The length needs to be established in the array statement; otherwise, SAS uses the default length of 8. You can use a length statement prior to the array statement to assign the length.
- ❸ The index variable, EXPENSE, becomes a data set variable that for this example contains the number of the expense, 1 - 6.

You can use the HBOUND function to return the upper bound of a one-dimensional array and the LBOUND function to return the lower bound. By using these functions you can avoid changing the bounds of an iterative DO loop each time you change the bounds of the array. The HBOUND and LBOUND functions are especially useful if you are using a range for the dimensions.

For example:

```

array expense{100:105} expense1 -
                          expense6;
do i = lbound(expense) to
      hbound(expense);

```

- ❹ Assign the array value in EX{expense} to the variable AMOUNT.

- ⑤ Assign the array value in NAME{expense} to the variable TYPEOFEXPENSE.
- ⑥ Use an OUTPUT statement in the DO loop in order to get an observation for each iteration of the loop.

The resulting data:

RESORT	Expense	Amount	TypeOfExpense
HOTEL1	1	165.89	Room Rate
HOTEL1	2	45.50	Daily Food Allowance
HOTEL1	3	78.00	Round of Golf
HOTEL1	4	45.00	Spa Visit
HOTEL1	5	12.00	Horse Back Riding
HOTEL1	6	35.00	Theme Park Admission
HOTEL2	1	215.32	Room Rate
HOTEL2	2	64.00	Daily Food Allowance
HOTEL2	3	54.00	Round of Golf
HOTEL2	4	0.00	Spa Visit
HOTEL2	5	18.00	Horse Back Riding
HOTEL2	6	30.00	Theme Park Admission
HOTEL3	1	178.90	Room Rate
HOTEL3	2	76.00	Daily Food Allowance
HOTEL3	3	32.00	Round of Golf
.	.	.	.
.	.	.	.
.	.	.	.

EXAMPLE 4: Using a two-dimensional array

You can think of a two-dimensional array as having 'rows' and 'columns', but to SAS, a two-dimensional array is no different from a one-dimensional array. It is still an alias for data set variables.

For example, if you used the array statement:

```
array num{2,3} (1,2,3,4,5,6);
```

SAS would create 6 variables, NUM1, NUM2, NUM3, NUM4, NUM5, and NUM6, and would set up the correspondence:

NUM{1,1}	NUM{1,2}	NUM{1,3}	NUM{2,1}	NUM{2,2}	NUM{2,3}
↓	↓	↓	↓	↓	↓
NUM1	NUM2	NUM3	NUM4	NUM5	NUM6
1	2	3	4	5	6

TASK: You know the following information about daily rates for taxes and gratuity charged by the resorts.

Resort	Taxes	Gratuity
1	18.54	14
2	17.84	20
3	12.50	15
4	14.25	18
5	16.33	12
6	19.00	16
:	:	:

You want to combine this information with the data in SASDATA.EXPENSES to calculate the total cost for each resort.

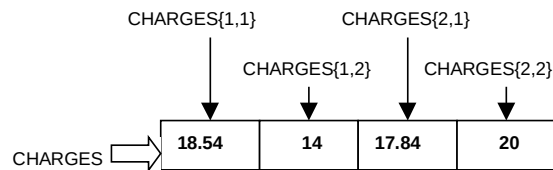
SOLUTION:

- Create a SAS data set containing these constants and merge with SASDATA.EXPENSES.
- Use IF..THEN logic and assignment statements to create the new variables.
- Since this information is in rows and columns, use a temporary two-dimensional array to hold the tax and gratuity information.

PROGRAM

```
data work.total;
  drop num;
  ❶ array charge{10,2} _temporary_
      (18.54      14
       17.84      20
       12.50      15
       14.25      18
       16.33      12
       19.00      16
       12.75      19
       14.98      16
       15.76      20
       13.75      17);
  set sasdata.expenses;
  ❷ num = substr(resort,6);
  ❸ tax = charge{num,1};
  gratuity = charge{num,2};
  total = sum(of expense1-
              expense6,tax,gratuity);
run;
```

- ❶ The array, CHARGES, refers to 10 rows and 2 columns. It is used to hold constants. Even though you can type them into rows and columns, and that helps you visually see what you need to do in the program, SAS is assigning these values to "contiguous slots" in memory. You can refer to these slots by row and column number.



If you had created variables instead of using `_TEMPORARY_`, the variables would have been named

CHARGES1 referring to CHARGES{1,1}
 CHARGES2 referring to CHARGES{1,2}
 CHARGES3 referring to CHARGES{2,1}
 CHARGES4 referring to CHARGES{2,2},
 etc.

- ② The variable RESORT in SASDATA.EXPENSES has values HOTEL1, HOTEL2, etc. You want to use this as an index when you reference the array. But an array index must be numeric. Therefore, you must extract the digit from the end of this value by using the function, `SUBSTR('text',start,end)`.
- ③ Create a variable, TAX and a variable GRATUITY using the appropriate resort number as the row dimension, and the appropriate number as the column number, where column 1 is the value for TAX and column 2 is the value for GRATUITY.

The resulting data:

RESORT	Expense1		Expense6	TAX	GRATUITY	TOTAL
	\$165.89	...	\$35.00	18.54	14.00	413.93
HOTEL2	\$215.32	...	\$30.00	17.84	20.00	419.16
HOTEL3	\$178.90	...	\$29.99	12.50	15.00	418.39
HOTEL4	\$210.78	...	\$24.00	14.25	18.00	388.03
HOTEL5	\$189.87	...	\$42.00	16.33	12.00	477.20
HOTEL6	\$312.15	...	\$19.40	19.00	16.00	611.55
HOTEL7	\$197.12	...	\$18.95	12.75	19.00	445.82
HOTEL8	\$240.98	...	\$32.00	14.98	16.00	469.09
HOTEL9	\$312.10	...	\$26.00	15.76	20.00	475.71
HOTEL10	\$187.98	...	\$29.65	13.75	17.00	372.92

CONCLUSION

If your data processing needs

- repetitive code
 - creation of many variables with the same attributes
 - converting data from a horizontal structure to a vertical structure
 - assigning many constants,
- then think ARRAY. Arrays are powerful tools because they allow you to group similar variables into one umbrella alias for similar processing. Though array references are assigned at compile time, the syntax allows for flexibility when declaring variable names and dimensions.

So enjoy yourself as your adventure in arrays takes you further in your data step programming.

CONTACT INFORMATION

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

Jane Stroupe
 SAS Institute Inc.
 Chicago Regional Office
 Two Prudential Plaza, 52nd Floor
 Chicago, IL 60601
 312-819-6800 ext. 8823
 Fax: 312-240-0342
 Email: Jane.Stroupe@sas.com