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.

<table>
<thead>
<tr>
<th>Resort Name</th>
<th>Resort</th>
<th>Expense1</th>
<th>Expense5</th>
<th>Expense6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe’s Pretty Good</td>
<td>HOTEL1</td>
<td>$165.89</td>
<td>...</td>
<td>$35.00</td>
</tr>
<tr>
<td>Resort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Larry, Curly, and</td>
<td>HOTEL2</td>
<td>$215.32</td>
<td>...</td>
<td>$30.00</td>
</tr>
<tr>
<td>Motel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sand And Shores</td>
<td>HOTEL3</td>
<td>$178.90</td>
<td>...</td>
<td>$29.99</td>
</tr>
<tr>
<td>Array of Sun Hotel</td>
<td>HOTEL4</td>
<td>$210.78</td>
<td>...</td>
<td>$24.00</td>
</tr>
<tr>
<td>Report Resort</td>
<td>HOTEL5</td>
<td>$189.87</td>
<td>...</td>
<td>$22.00</td>
</tr>
<tr>
<td>NoTell Hotel</td>
<td>HOTEL6</td>
<td>$312.15</td>
<td>...</td>
<td>$19.40</td>
</tr>
<tr>
<td>Dew Drop Inn</td>
<td>HOTEL7</td>
<td>$197.12</td>
<td>...</td>
<td>$18.95</td>
</tr>
<tr>
<td>Greatest View Resort</td>
<td>HOTEL8</td>
<td>$240.98</td>
<td>...</td>
<td>$32.00</td>
</tr>
<tr>
<td>Holy Cow Hotel</td>
<td>HOTEL9</td>
<td>$312.10</td>
<td>...</td>
<td>$26.00</td>
</tr>
<tr>
<td>Come On Inn</td>
<td>HOTEL10</td>
<td>$187.98</td>
<td>...</td>
<td>$29.65</td>
</tr>
</tbody>
</table>

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;

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 array-name(n) $ length elements;
array-name any valid SAS name (avoid function names)
n number of elements to which the array refers
$ indication that the array refers to character data
length the length for new character variables. The default is 8.
elements 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);
  ❁ array ex{6} expense1 - expense6;
  ❁ array seasonal{6} seasonal1 - seasonal6;
  set sasdata.expenses (drop=ResortName);
  ❁ do i=1 to 6;
     seasonal{i} = ex{i}*1.25;
  end;
  ❁ format expense1 - expense6
     seasonal1 - seasonal6
     dollar9.2;
run;
```

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.

3. The ARRAY statement can automatically name new variables.
   For example,
   array seasonal{6};
   would automatically create variables SEASONAL1 - SEASONAL6.

4. 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;

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);
  array budget{6} _temporary_ (175,75,25,35,25,30);
  array expense{*} expense1 - expense6;
  array diff{6};
  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:

<table>
<thead>
<tr>
<th>Resort</th>
<th>expense1</th>
<th>expense2</th>
<th>DIFF1</th>
<th>DIFF5</th>
<th>DIFF6</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOTEL1</td>
<td>$165.89</td>
<td>...</td>
<td>9.11</td>
<td>...</td>
<td>5.00</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>$215.32</td>
<td>...</td>
<td>-40.32</td>
<td>7.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HOTEL3</td>
<td>$178.90</td>
<td>...</td>
<td>-3.90</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>HOTEL4</td>
<td>$210.78</td>
<td>...</td>
<td>-35.78</td>
<td>8.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HOTEL5</td>
<td>$189.87</td>
<td>...</td>
<td>-42.00</td>
<td>13.00</td>
<td>12.00</td>
</tr>
<tr>
<td>HOTEL6</td>
<td>$312.15</td>
<td>...</td>
<td>-13.00</td>
<td>11.00</td>
<td>10.60</td>
</tr>
<tr>
<td>HOTEL7</td>
<td>$197.12</td>
<td>...</td>
<td>-22.12</td>
<td>1.00</td>
<td>1.05</td>
</tr>
<tr>
<td>HOTEL8</td>
<td>$245.98</td>
<td>...</td>
<td>-65.98</td>
<td>13.00</td>
<td>2.00</td>
</tr>
<tr>
<td>HOTEL9</td>
<td>$312.10</td>
<td>...</td>
<td>-137.10</td>
<td>10.00</td>
<td>4.00</td>
</tr>
<tr>
<td>HOTEL10</td>
<td>$187.98</td>
<td>...</td>
<td>-12.98</td>
<td>1.00</td>
<td>0.35</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Expense</th>
<th>TypeOfExpense</th>
<th>Expenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Room Rate</td>
<td>$165.29</td>
</tr>
<tr>
<td>2</td>
<td>Daily Food Allowance</td>
<td>$45.50</td>
</tr>
<tr>
<td>3</td>
<td>Round of Golf</td>
<td>$72.00</td>
</tr>
<tr>
<td>4</td>
<td>Spa Visit</td>
<td>$45.00</td>
</tr>
<tr>
<td>5</td>
<td>Horse Back Riding</td>
<td>$12.00</td>
</tr>
<tr>
<td>6</td>
<td>Theme Park Admission</td>
<td>$35.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

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

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.

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.

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:

```sas
array ex{100:105} expense1 - expense6;
  do i = 1bound(ex) to hbound(ex);
    output;
  end;
```

Assign the array value in EX(expense) to the variable AMOUNT.
Assign the array value in NAME(expense) to the variable TYPEOFEXPENSE.

Use an OUTPUT statement in the DO loop in order to get an observation for each iteration of the loop.

The resulting data:

<table>
<thead>
<tr>
<th>Resort</th>
<th>Expense</th>
<th>Amount</th>
<th>TypeOfExpense</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOTEL1</td>
<td>1</td>
<td>165.89</td>
<td>Room Rate</td>
</tr>
<tr>
<td>HOTEL1</td>
<td>2</td>
<td>45.50</td>
<td>Daily Food Allowance</td>
</tr>
<tr>
<td>HOTEL1</td>
<td>3</td>
<td>78.00</td>
<td>Round of Golf</td>
</tr>
<tr>
<td>HOTEL1</td>
<td>4</td>
<td>45.00</td>
<td>Spa Visit</td>
</tr>
<tr>
<td>HOTEL1</td>
<td>5</td>
<td>12.00</td>
<td>Horse Back Riding</td>
</tr>
<tr>
<td>HOTEL1</td>
<td>6</td>
<td>35.00</td>
<td>Theme Park Admission</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>1</td>
<td>215.32</td>
<td>Room Rate</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>2</td>
<td>64.00</td>
<td>Daily Food Allowance</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>3</td>
<td>54.00</td>
<td>Round of Golf</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>4</td>
<td>0.00</td>
<td>Spa Visit</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>5</td>
<td>19.00</td>
<td>Horse Back Riding</td>
</tr>
<tr>
<td>HOTEL2</td>
<td>6</td>
<td>30.00</td>
<td>Theme Park Admission</td>
</tr>
<tr>
<td>HOTEL3</td>
<td>1</td>
<td>178.90</td>
<td>Room Rate</td>
</tr>
<tr>
<td>HOTEL3</td>
<td>2</td>
<td>75.00</td>
<td>Daily Food Allowance</td>
</tr>
<tr>
<td>HOTEL3</td>
<td>3</td>
<td>32.00</td>
<td>Round of Golf</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

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

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

```sas
data work.total;
    drop num;
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;
num = substr(resort,6);
tax = charge{num,1};
gratuity = charge{num,2};
total = sum(of expense1-expense6,tax,gratuity);
run;
```

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.
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