Paper 032-2009

Using SAS® Arrays to Manipulate Data

Ben Cochran, The Bedford Group, Raleigh, NC

ABSTRACT

The DATA step has been described as the best data manipulator in the business. One of the constructs that gives the DATA step so much power is the SAS array. This presentation takes the user on a tour of SAS array applications starting from a very elementary level to more advanced examples. After brief explanations of each application, the attendees will get a chance to try their skills at solving an array of challenges.

INTRODUCTION

Often, SAS users need to manipulate data to get it ready for a report, application, or a data warehouse. As a matter of fact, most of the time someone spends doing SAS programming is spent manipulating data. In an earlier career, I found myself spending upwards of 80% of my time just getting data in the 'right shape'. I was a little bewildered in that I had already spent several years not only manipulating data, but teaching SAS courses on the subject. Not only teaching, but WRITING courses on the subject as well. I had taken an informal survey among my consulting friends, and they reassured me that data manipulation is a very timely task. Their records indicated that as much as 90% of their programming time was spent on data manipulation. This paper looks at several ways that arrays can be used to manipulate data.

DEFINITION

A SAS array is a set of variables that are grouped together and referred to by a single name. These variables are known as **elements** of the array. Each element is referred to by an index value which represents its position in the array. A common analogy is a grocery list. On one list (lets call it grocery_list) , I can have several items; like apples, bacon, chocolate, bread, eggs, coffee, milk, and ice cream. I can refer to the white beverage as milk, or the seventh item on the list. I can refer to the frozen desert as ice cream, or the eighth item on the list. I also need to go to the office supply store to make some purchases. I have a list for that trip, too… lets call it office_list. On that one list, I have laptop, printer, pencils, and paper. I can refer to the item that writes as a pencil, or the third element of the office_list.

SAS arrays are like these lists. They have names and contain a number of items (variables). You can refer to each item by (variable) name, or you can refer to each item by its number on the list (array).

You can do many things with SAS arrays. You can:

- perform repetitive calculations,
- create many variables with like attributes,
- read data,
- make the same comparison for several variables,
- perform table lookup.

. .

Let's start with a simple example. I have a sales dataset that has all its values in US Dollars. There is a French subsidiary that needs to have these values converted to Euros. For illustration purposes, lets use the following exchange rate:

1 USD = 0.75 Euros or 1 Euro = 1.333 USD

The dataset looks like this:

Obs	COMPANY	Sales_2001	Sales_2000	Sales_2002	Sales_2003
1	OUTWAY COMPANY	523.24	288.24	1,500.24	1,660.57
2	AUSTRALIAN WAY HOSPITAL	523.24	499.24	804.24	874.62
3	NATIONAL GROVE UNIVERSITY	523.24	198.24	1,308.24	1,340.82
4	STATE INCOME OFFICE	1,170.00	529.00	1,596.00	1,876.61
5	MELROSE TECH	251.29	628.71	1,178.29	1,320.58
6	MELROSE UNIVERSITY	251.29	149.71	1,144.29	1,237.36
7	MELROSE College	251.29	640.71	1,106.29	1,318.61
8	BUREAU OF TECHNOLOGY	251.29	418.71	954.29	1,077.62
9	LETELLO LABORATORIES	1,120.00	948.00	268.00	309.77

.

We need to write a DATA step to solve this problem. If we did not know anything about arrays, the program would look like this:

```
data euros;
     set sasuser.sales2003;
     esales_2000 = sales_2000 * 1.333;
     esales_2001 = sales_2001 * 1.333;
     esales_2002 = sales_2002 * 1.333;
     esales_2003 = sales_2003 * 1.333;
run;
```

Notice the repetitive calculations. This is not too bad if we have only a small number of calculations to perform. But, what if we have dozens, or even hundreds? Your program could become unruly and harder to maintain. The Use of arrays can simplify the process.

The typical syntax of an array statement is:

array *array-name* { n } [$] [length] *variable-names* ;

where:

array-name	is a valid SAS name.
n	is the number of elements in the array
$	is used when the array elements are character variables (the default type is numeric)
length	is used at compile time to assign lengths to character variables which are previously undefined.

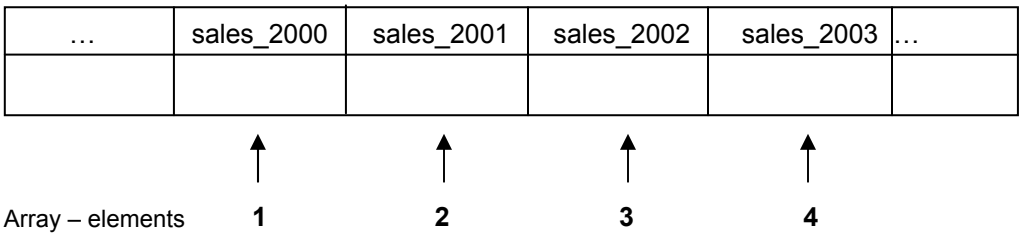
An array statement:

- must refer to all character or all numeric variables,
- must appear in the DATA step before the array elements are referenced,
- can be used to create variables,
- is a compile time statement. SAS does not see it at execution time..

This array statement defines the four sales variables as elements of a SAS array.

```
array sales { 4 } sales_2000 – sales_2003 ;
```

If we look at the program data vector, this is what we would see:



Now, values can be referred to by (variable) name, or by array element numbers.

DO LOOPS are typically used to process each element of an array. The value of the DO LOOP index variable identifies the array element to be processed as shown in the pseudo code below.

```
array  sales  { 4 }  sales_2000 – sales_2003 ;

do  index-variable = 1 to 4;
   …  sales{index-variable} …
end;
```

ARRAY APPLICATIONS

Application 1: The sales manager wants a report showing sales figures in Euros instead of US Dollars. Write a DATA step using an array to accomplish this task. .

```
data  task1(keep=company sales:  esales:)  ;
  array  sales   { 4 }  sales_2000 – sales_2003 ;
  array  esales { 4 }  esales_2000 – esales_2003;
  set sasuser.sales2003;
  do  i  = 1  to 4;
      esales{ i } = sales{ i } * 1.3333;
  end;
run;.
```

Notice the **sales:** on the keep= option. It is shorthand for all variables that start with the letters **sales.** The same is true for **esales:.**

The following PROC PRINT step is used to generate the output.

```
proc print data=task1(obs = 7)  ;
    var  sales_2000 – sales_2003  company  esales_2000 – esales_2003 ;
    format sales_2000 – sales_2003 dollar12.2  eales_2000 – esales_2003
euro12.2;
```

Obs	sales_2000	sales_2001	sales_2002	sales_2003	COMPANY
1	$288.24	$523.24	$1,500.24	$1,660.57	OUTWAY COMPANY
2	$499.24	$523.24	$804.24	$874.62	AUSTRALIAN WAY HOSPITAL
3	$198.24	$523.24	$1,308.24	$1,340.82	NATIONAL GROVE UNIVERSITY
4	$529.00	$1,170.00	$1,596.00	$1,876.61	STATE INCOME OFFICE
5	$628.71	$251.29	$1,178.29	$1,320.58	MELROSE TECH
6	$149.71	$251.29	$1,144.29	$1,237.36	MELROSE UNIVERSITY
7	$640.71	$251.29	$1,106.29	$1,318.61	MELROSE College

Obs	esales_2000	esales_2001	esales_2002	esales_2003
1	€384.22	€697.47	€1,999.81	€2,213.53
2	€665.48	€697.47	€1,072.05	€1,165.87
3	€264.25	€697.47	€1,743.88	€1,787.31
4	€705.16	€1,559.61	€2,127.47	€2,501.52
5	€838.07	€334.97	€1,570.66	€1,760.33
6	€199.56	€334.97	€1,525.34	€1,649.40
7	€854.07	€334.97	€1,474.69	€1,757.71

Notice the effect of the **euro** format.

Application 2: A certain dataset has all its date values in character variables (not true SAS dates). Write a DATA step to convert a series of character variables to numeric values.

```
data dates;
    length date1 – date3 $10;
    input  date1 $  date2 $  date3 $;
    datalines;
11jun08   11jun2008    06/11/2008
10jul08    10jul2008    07/10/2008
;
```

```
data convert;
    set dates;
    array c_dates { 3 } $ 10 date1 – date3;
    array n_dates { 3 } n_date1 - n_date3;
    do i = 1 to 3;
       n_dates{ I } = input(c_dates{ i },  anydtdte10.);
    end;
 run;
proc print data=convert;
run;
```

Obs	date1	date2	date3	n_date1	n_date2	n_date3
1	11jun08	11jun2008	06/11/2008	17694	17694	17694
2	10jul08	10jul2008	07/11/2008	17723	17723	17724

Notice all the **N_DATE** variables have been converted to SAS dates and stored as numbers.

The next application uses the data below. It is characterized by a series of cholesterol readings generated on a series of dates. Some patients have more readings than others. The head physician at the clinic wants to know the difference in readings for each patient from one month to the next.

Obs	patient_ id	date_1	reading_ 1	date_2	reading_ 2	date_3	reading_ 3	date_4	reading_ 4
1	1009	03JAN	216.9	06FEB	212.3	09MAR	209.6	08APR	207.8
2	1017	02JAN	190.2	04FEB	189.5	05MAR	192.8	.	.
3	1023	04JAN	256.3	05FEB	249.5	06MAR	243.5	03APR	241.2
4	1234	05JAN	196.2	06FEB	199.9	07MAR	197.5	.	.
5	1333	06JAN	192.5	07FEB	196.5	08MAR	195.3	.	.
6	1354	07JAN	222.6	03FEB	226.2	04MAR	229.8	05APR	226.3
7	1378	03JAN	212.5	02FEB	210.8	05MAR	207.9	.	.
8	1444	04JAN	206.2	01FEB	202.3	02MAR	200.2	.	.
9	1545	05JAN	188.8	02FEB	189.0	03MAR	190.4	.	.
10	1812	06JAN	199.2	03FEB	198.2	04MAR	196.8	04APR	195.2

Application 3: Use array processing to calculate monthly differences in Cholesterol readings.

```
data difference (drop = i);
   array chol   { 4 }  reading_1 – reading_3;
   array diff    { 3 }  ;
   set sasuser.cholesterol(drop=date_2 – date_4);
      do i = 1 to 3;
         diff{ i } = chol{ i + 1 } – chol{ i };
      end;
   rename date_1 = Starting_Date;
 run;
```

The Program Data Vector looks like this:

reading_1	reading_2	reading_3	reading_4	diff1	diff2	diff3	patient_id	date_1	i

Notice that there are two ARRAY statements. On the second ARRAY statement, notice that there are no variables listed. This is an example of an ARRAY statement creating new variables (DIFF1, DIFF2, and DIFF3). Also, with 4 readings, there will only be 3 differences... (between 1 and 2, between 2 and 3, and between 3 and 4).

The following PROC PRINT step is used to generate the report.

```
proc print data=difference(obs=7) width=min label;
  var patient_id Starting_Date diff1 - diff3;
  format starting_date mmddyy10.;
  title 'Changes in Cholesterol Readings';
  label diff1 = 'Change From Read_1 to Read_2'
        diff2 = 'Change From Read_2 to Read_3'
        diff3 = 'Change From Read_3 to Read_4';
run;
```

Changes in Cholesterol Readings

Obs	patient_ id	Starting_ Date	Change From Read_1 to Read_2	Change From Read_2 to Read_3	Change From Read_3 to Read_4
1	1009	01/03/2001	-4.6	-2.7	-1.8
2	1017	01/02/2001	-0.7	3.3	.
3	1023	01/04/2001	-6.8	-6.0	-2.3
4	1234	01/05/2001	3.7	-2.4	.
5	1333	01/06/2001	4.0	-1.2	.
6	1354	01/07/2001	3.6	3.6	-3.5
7	1378	01/03/2001	-1.7	-2.9	.

The doctor was so pleased with this report that she now wants to see the **percent** difference from reading to reading.

Application 4: Use array processing to calculate the **percent** difference in readings from month to month.

```
data difference (drop = i);
    array chol   { 4 }  reading_1 – reading_3;
    array diff    { 3 }  ;
    array percent { 3 };
    set sasuser.cholesterol(drop=date_2 – date_4);
        do i = 1 to 3;
            diff{ i } = chol{ i + 1 } – chol{ i };
            percent { i } = diff{ I } / chol { I };
        end;
    rename date_1 = Starting_Date;
 run;
```

This program is essentially the same as the previous one except for two statements. They are shown above in boldface font. A similar PROC PRINT is used to create the output.

```
proc print data=p_difference(obs=7) width=min label;
  var patient_id Starting_Date percent1 - percent3;
  format starting_date mmddyy10. percent1 - percent3 percent8.1;
  title 'Percent Difference in Cholesterol Readings';
  label percent1 = '% Change From Read_1 to Read_2'
        percent2 = '% Change From Read_2 to Read_3'
        percent3 = '% Change From Read_3 to Read_4';
run;
```

Percent Difference in Cholesterol Readings

Obs	patient_ id	Starting_ Date	% Change From Read_1 to Read_2	% Change From Read_2 to Read_3	% Change From Read_3 to Read_4
1	1009	01/03/2001	(2.1%)	(1.3%)	(0.9%)
2	1017	01/02/2001	(0.4%)	1.7%	.
3	1023	01/04/2001	(2.7%)	(2.4%)	(0.9%)
4	1234	01/05/2001	1.9%	(1.2%)	.
5	1333	01/06/2001	2.1%	(0.6%)	.
6	1354	01/07/2001	1.6%	1.6%	(1.5%)
7	1378	01/03/2001	(0.8%)	(1.4%)	.

Notice the effects of the **percent** format. The negative values are in parenthesis.

Suppose we need to find out the average cholesterol reading. What does that mean? In order to do this, we need to get all the reading values in one column.

First, we need to look at the **DIM** function. The DIM function returns the number of elements in an array. The typical syntax is:

```
Dim < n > (array – name) ...
```

where **n** represents the dimension of the array. The default value is 1 or blank.

Application 5: Use array processing to transform a data set from one row per **patient**, to one row per **reading** per patient. Looking at the data, we need to take the data as it is and go from this shape...

Obs	patient_id	date_1	reading_1	date_2	reading_2	date_3	reading_3	date_4	reading_4
1	1009	03JAN	216.9	06FEB	212.3	09MAR	209.6	08APR	207.8
2	1017	02JAN	190.2	04FEB	189.5	05MAR	192.8	.	.
3	1023	04JAN	256.3	05FEB	249.5	06MAR	243.5	03APR	241.2
4	1234	05JAN	196.2	06FEB	199.9	07MAR	197.5	.	.
5	1333	06JAN	192.5	07FEB	196.5	08MAR	195.3	.	.
6	1354	07JAN	222.6	03FEB	226.2	04MAR	229.8	05APR	226.3
7	1378	03JAN	212.5	02FEB	210.8	05MAR	207.9	.	.
8	1444	04JAN	206.2	01FEB	202.3	02MAR	200.2	.	.
9	1545	05JAN	188.8	02FEB	189.0	03MAR	190.4	.	.
10	1812	06JAN	199.2	03FEB	198.2	04MAR	196.8	04APR	195.2

... where there is one row per **patient,** to this shape...

Obs	patient_id	date	reading
1	1009	03JAN2001	216.9
2	1009	06FEB2001	212.3
3	1009	09MAR2001	209.6
4	1009	08APR2001	207.8
5	1017	02JAN2001	190.2
6	1017	04FEB2001	189.5
7	1017	05MAR2001	192.8
8	1023	04JAN2001	256.3
9	1023	05FEB2001	249.5

... where there is one row per reading.

The DATA step to do this is shown below.

```
data transform(keep=patient_id date reading);
  array chol    {*} reading_1 - reading_4;
  array dates   {*} date_1 - date_4;
  set p_sug.cholesterol;
  do i = 1 to dim(dates) while (dates{i} ne .);
    date = dates{i};
    reading = chol{i};
    output;
  end;
run;

proc print data=transform(obs=9) width=min label;
  format date date9.;
  title 'Transformed Data Set';
run;
```

Notice the {*} syntax on the ARRAY statement, and the DIM function in the DO Loop. The {*} tells SAS to figure out how many elements are in the array. The DIM function returns the STOP value as the number of elements in the array. So, in this case, SAS figures out that there are four elements in the array. The DIM function tells SAS to execute loop four times.

With this DATA step, there will be four observations written out for every observation read in. Notice the placement of the SET statement and the OUTPUT statement. For every iteration of this DATA step, the SET statement executes once, and the OUTPUT statement execution executes either 3 or 4 times.

Obs	patient_ id	date	reading
1	1009	03JAN2001	216.9
2	1009	06FEB2001	212.3
3	1009	09MAR2001	209.6
4	1009	08APR2001	207.8
5	1017	02JAN2001	190.2
6	1017	04FEB2001	189.5
7	1017	05MAR2001	192.8
8	1023	04JAN2001	256.3

The next set of applications uses the following data.

Obs	store	year	qtr	amount
1	050	2005	1	$23,508.99
2	050	2005	2	$26,172.41
3	050	2005	3	$25,676.63
4	050	2005	4	$27,431.05
5	050	2006	1	$45,968.31
6	050	2006	2	$51,178.09
7	050	2006	3	$53,602.54
8	050	2006	4	$53,728.53
9	050	2007	1	$50,389.34
10	050	2007	2	$56,524.56
11	050	2007	3	$59,063.14
12	050	2007	4	$62,038.21
13	050	2008	1	$55,953.56
14	050	2008	2	$63,808.45
15	050	2008	3	$66,917.95
16	050	2008	4	$66,967.36

Application 6: The VP of Sales predicts quarterly sales growth next year will be 1.1%, 2.2%, 0.1%, and 3.3%. Write a program that shows what the sales will be if this estimate is correct.

```
data _4cast(drop=i);
    array qtrs     {4} y2008q1  y2008q2  y2008q3  y2008q4;
    array _4cast {4} y2009q1  y2009q2  y2009q3  y2009q4;
    array pc_inc {4} _temporary_ (1.011  1.022  1.001  1.033)  ;
    set summed_q;
    do i = 1 to dim(qtrs);
        _4cast(i) = qtrs(i) * pc_inc(i);
     end;
run;
title 'Predicting Quarterly Increases of  1.1%  2.2%  0.1%  3.3%';
proc print data=_4cast(obs=8) width=min;
   format _numeric_ dollar10.;
   var store y:;
run;
```

Notice the use of the _TEMPORARY_ keyword in the third ARRAY statement. This sets up temporary storage locations (not variables) for the 4 values of the projected sales change. Also notice the VAR statement in the print procedure. It can be translated to state: print the STORE variable, and then all those that start with the letter 'y'.

Predicting Quarterly Increases of 1.1% 2.2% 0.1% 3.3%

Obs	store	y2008q1	y2008q2	y2008q3	y2008q4	y2009q1	y2009q2	y2009q3	y2009q4
1	050	$55,954	$63,808	$66,918	$66,967	$56,569	$65,212	$66,985	$69,177
2	100	$52,810	$57,927	$60,315	$61,997	$53,391	$59,201	$60,376	$64,043
3	150	$98,781	$109,786	$111,562	$116,401	$99,867	$112,201	$111,674	$120,243
4	200	$72,717	$82,473	$82,752	$84,426	$73,517	$84,288	$82,835	$87,213
5	250	$53,893	$60,228	$61,781	$64,732	$54,486	$61,553	$61,842	$66,868
6	300	$44,693	$50,089	$49,578	$51,987	$45,184	$51,191	$49,628	$53,703
7	350	$35,863	$39,651	$42,868	$42,041	$36,258	$40,523	$42,911	$43,429
8	400	$47,254	$51,455	$53,053	$55,954	$47,773	$52,587	$53,106	$57,801

When the processing of data depends on more than one factor, you can use **multidimensional** arrays. The next set of applications will illustrate processing data with multidimensional arrays. The typical syntax of a multidimensional ARRAY statement is:

```
array  array-name {... ,  rows, columns }  $  length  elements  (initial values) ;
```

If you write the following ARRAY statement...

```
array  test  { 2 , 4 }  ( 11, 22, 33, 44, 55, 66, 77, 88 ) ;
```

... conceptionally creates the following two dimensional 'table'.

test {1,1} 11	test {1,2} 22	test {1,3} 33	test{1,4} 44
test {2,1} 55	test {2,2} 66	test{2,3} 77	test{2,4} 88

The following application uses the following data.

Obs	store	year	qtr	_FREQ_	total_sales
1	50	2000	1	2971	$301,910.00
2	50	2000	2	3390	$341,540.00
3	50	2000	3	3420	$339,800.00
4	50	2000	4	3694	$377,610.00
5	50	2001	1	3332	$324,370.00
6	50	2001	2	3711	$402,570.00
7	50	2001	3	3866	$385,760.00
8	50	2001	4	3945	$379,810.00

The VP of Sales calculated that the average profit over the last several years per quarter was 9.1%, 10.3%, 11.5%, and 7.8% for quarters 1 through 4 respectively. Based on these calculations, she wants to determine the profits for the 8 quarters in this data set.

Application 7: Write a DATA step to calculate profit for 2 years, 2000 and 2001.

```
data qtr_profit(drop= year qtr _freq_ total_sales
                      yr2000_q1 -- yr2001_q4);
  array qtrs {2000:2001,4} yr2000_q1 - yr2000_q4
                           yr2001_q1 - yr2001_q4;
  array q_pc {4} _temporary_ (.091, .103, .113, .078);
  array q_profit {2000:2001,4} prof_y00q1 - prof_y00q4
                               prof_y01q1 - prof_y01q4;
  do until(last.store=1);
     set qtrly_sales;
     by store;
     qtrs{year,qtr} = total_sales;
     q_profit{year,qtr}=total_sales * q_pc{qtr};
  end;
run;
```

In the first and third ARRAY statements, the 2 dimensional array is 2 years (row dimension) and 4 quarters (column dimension). The second ARRAY statement contains the projected percent increases. The output just contains the columns with the projected profit and is shown below.

5

Quarterly Profits for 2000 and 2001

Obs	store	prof_ y00q1	prof_ y00q2	prof_ y00q3	prof_ y00q4	prof_ y01q1	prof_ y01q2	prof_ y01q3	prof_ y01q4
1	50	$27,474	$35,179	$38,397	$29,454	$29,518	$41,465	$43,591	$29,625
2	100	$43,007	$55,066	$58,729	$43,224	$48,862	$60,063	$66,673	$50,325
3	150	$62,477	$73,776	$87,140	$60,189	$65,938	$83,770	$95,822	$66,443
4	200	$35,512	$42,319	$48,190	$34,489	$36,313	$49,323	$53,987	$37,280
5	250	$36,008	$42,940	$49,155	$35,150	$39,652	$48,505	$55,853	$39,708
6	300	$22,922	$27,412	$33,923	$23,490	$24,474	$34,586	$36,826	$25,577
7	350	$19,227	$25,388	$29,046	$19,540	$22,954	$27,195	$30,356	$21,938
8	400	$22,938	$31,042	$31,917	$24,185	$23,190	$30,785	$36,394	$26,705

CONCLUSION

Using arrays in SAS can allow the user a great deal of flexibility to manipulate data in a number of different ways.

ACKNOWLEDGMENTS

As always, I would like to thank the people in the Technical Support department of SAS Institute for their kind and helpful assistance for this past year. A special thank you goes to Larry Stewart of the Education Division of SAS Institute for being my first SAS mentor and teaching me a lot about the DATA step.

CONTACT INFORMATION

If you have any questions or comments, the author can be reached at:

Ben Cochran
The Bedford Group
3224 Bedford Avenue
Raleigh, NC 27607
Work Phone: 919.741.0370
Email: bedfordgroup@nc.rr.com
Web: www.bedford-group.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.