Paper 215

Using ARRAYs: The Basics

George Matthews
The University of Georgia

ABSTRACT

As a Statistical Software Consultant in an academic environment, I often encounter novice users who need to process several variables in the same way. After scolding the user for attempting to perform this task with hundreds of IF-THEN statements, I suggest they consider an array. In the DATA step, you can put variables into a temporary group called an array. Arrays provide a very powerful tool for group processing. This paper will provide participants with the basics of array processing and is intended for SAS users who have some experience with the DATA step, and no or very little experience with arrays.

INTRODUCTION

Users of third generation programming languages are often quite pleased when they discover that the array statement can be utilized in the SAS Language but SAS arrays neither allocate storage or declare data structures. A SAS array is a collection of SAS variables that are grouped under a single name. Each variable of the array is

- called an *element*.
- identified by an index value that represents the position of the element in the array.

When you reference an array element in your SAS program the corresponding variable is substituted for the reference. Arrays are temporary in SAS existing only for the duration of the DATA step in which they are found. There are two primary forms of the array statement, one which declares an "implicitly subscripted" array and one an "explicitly subscripted" array. An implicit array reference consists of the array name only. An explicit array reference consists of the array name and subscript. You can use arrays to simplify the code needed to

- read data.
- compare variables.
- create many variables with the same attributes.
- perform repetitive calculations.
- perform a table lookup.
- rotate SAS datasets by making variables into observations or observations into variables.

The syntax for accomplishing a few of these tasks will be discussed in more detail in the next section. The most important aspect of arrays is that they will allow you to save time and typing by avoiding repetitious data step coding. The question you might ask at this time is "What is repetitious code?" Repetitious code occurs when you find yourself coding the same line over and over again with the only change being the variable name. You achieve code simplicity by grouping the variables into an array and performing array processing. Array Processing involves:

- grouping variables into arrays.
- performing an action, then repeating the action.
- selecting the current variable to be acted upon.

Array Processing is enhanced by using a DO loop. The DO loop is used to tell the SAS system to perform the same action several times. The syntax for the DO loop is as follows:

DO **index-variable**=1 TO **number-of-variables-in-array**;
    more SAS statements
END;

This is called an iterative DO loop. The DO loop begins with an iterative DO statement, followed by other SAS statements and ends with an END statement. The DO statement contains an **index-variable** whose value changes at each iteration of the loop and SAS automatically adds this variable to the data set being created.

SYNTAX

A simple array statement has the following syntax:

array *array-name {number of elements} varlist*;

where *array-name* is a SAS name that you choose to identify the group of variables, the *number-of-elements* enclosed in braces tells the SAS System how many variables you are grouping, and *varlist* (also referred to as *elements*) contains the variable names or _TEMPORARY_ to create a list of temporary elements. You can use square brackets [ ] or parentheses ( ) to surround the *number-of-elements*. The online help for the SAS System provides the following more complex syntax for the array statement:

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

where *array-name* is a SAS name that you choose to identify the group of variables, *n* is either the dimension of the array or an asterisk (*) to indicate the dimension is determined from the number of array elements or initial values, *$* indicates that the elements in the array are character variables, *length* defines the length for new variables, *elements* contains the variable names or _TEMPORARY_ to create a list of temporary elements and initial values gives the initial values for corresponding element. The following example creates an array name DIGTWO, containing three variables AGE, KIDS, and JOB:

```
array digtwo {3} age kids job;
```

The SAS System assigns each array element an *array reference* with the form *array-name {subscript}*, where *subscript* is the position of the variable in the list (in this case1, 2, or 3).

Table 1 -- Describes the array reference for each variable

Variable Name	Array Reference
AGE	DIGTWO{1}
KIDS	DIGTWO{2}
JOB	DIGTWO{3}

The subscript can be a number, the name of the variable whose value is the number or an expression.

RULES FOR ARRAY STATEMENTS

Some important rules to keep in mind when using arrays in your SAS programs:

1. An array statement must contain all numeric or all character elements.
2. An array statement must be used to define an array before the array name can be referenced.
3. An array statement creates variables if they do not exist in the Program Data Vector.
4. An array statement is not executable.
5. If you use the asterisk (*) to designate the number of elements, you cannot use it with _TEMPORARY_ arrays.
6. If you use the asterisk (*) to designate the number of elements, you must list each array element.
7. The array name can be any name as long as it does not match any of the variable names in your data set or any SAS keywords and it must adhere to the SAS naming convention.
8. Array names cannot be used in *label*, *format*, *drop*, *keep* or *length* statements.

Within array processing you are provided with additional functions that will help you in setting up your arrays and repetitive processing loops. These are:

DIM: represents the total count and automatically changes the upper bound of the index variable.
Syntax: DIM(index-variable)

LBOUND: automatically changes the lower bound of the index-variable.
Syntax: LBOUND(index-variable)

HBOUND: automatically changes the upper bound of the index-variable.
Syntax: HBOUND(index-variable)

The following array statement defines an array with a total of six elements, a lower bound of 89 and an upper bound of 94; it represents the calendar years 1989 through 1994:

```
array years {89:94} est pst cst ost jst zst;
```

To process the array elements in an iterative DO loop, use the following statement:

```
do j=lbound(years) to hbound(years);
   more SAS statements
end;
```

The value of lbound(years) is 89 and the value of hbound(years) is 94. The DIM function, in this example, would return a value of 5, therefore, the hbound function is more appropriate for this example.

ARRAY EXAMPLES

Using SAS to create the Variable Names - Example 1

```
array temp {4};
```

This array statement creates elements temp1, temp2, temp3, and temp4.

Using SAS to create the Variable Names - Example 2

```
array temp{2:4};
```

This array statement would be the same as previous example if the lower bound was 1, but 2 is the lower bound and 4 is the upper bound of the dimension of this array. This array statement creates elements temp2, temp3, and temp4.

Repetive Calculations - Example 3

This code is used to compute the amount of the monthly bonus checks for employees at Zebra Motor Corporation.

```
array  bonus {12} mnthly1-mnthly12;
 do j=1 to 12;
 bonus{j}=bonus{j}*.20;
 end;
```

Creating Variables with an Array - Example 4

The code in the previous example was used to refer to existing variables. Utilizing the same code in the previous example, this example demonstrates how to create new variables using arrays. The variables QPERCENT1-QPERCENT12 are created in this example.

```
array  bonus {12} mnthly1-mnthly12;
 array  qpercent{12};
 do j=1 to 12;
 bonus{j}=bonus{j}*.20;
 qpercent{j}=bonus{j}/5000;
 end;
```

Recoding Likert Scale Observations - Example 5

This code is used to reverse a five point likert value by subtracting the current value of each question from 6, therefore, a value of 5 becomes 1, 4 becomes 2, 2 becomes 4, and 1 becomes 5.

```
array gapoll{35} q1-q35;
do j =1 to 35;
gapoll{j} = 6 - gapoll{j};
end;
```

CONCLUSION

In this paper we have briefly covered the basic concepts of array processing. This paper demonstrates that you can save considerable amounts of programming time by using arrays.

ACKNOWLEDGMENT

The author wishes to thank Jeannie McElhannon for word processing and clerical support, Barbie Matthews for comments, suggestions, and support throughout the development of this paper. The author can be contacted at:

The University of Georgia
UCNS
Computer Services Annex
Athens, Ga. 30602-1911
(706) 542-5359
gmatthew@uga.edu

REFERENCES

1. SAS Language Guide, Version 6, First Edition, SAS Institute.

2. Jaffe, Jay (1994), Mastering the SAS System, Second Edition, 391-398.

3. Cody, Ron & Pass, Ray, SAS Programming by Example, First Edition, 135-145.