INTRODUCTION TO ARRAYS AND ARRAY PROCESSING
Earl R. Westerlund
Eastman Kodak Company

Abstract
Arrays are a very useful part of the SAS® DATA step. With arrays, you can do repetitive processing on a number of SAS variables. You can also set up temporary arrays to hold working values that you would not want stored in a data set, or to hold constant values.

This paper will begin with a simple introduction to arrays: what they are good for, what they can't do, and how to set one up. From there, we will discuss how to use them to group sets of SAS variables and to set up temporary work spaces. Finally, we will discuss some more advanced features of arrays, including multidimensional arrays, array bounds, and related DATA step functions.

This talk is intended for SAS users who have some experience with the DATA step, and little or no experience with arrays.

Why Use Arrays?
Sooner or later, everyone who does DATA step programming encounters a situation where a group of variables must be processed in the same way. For example, suppose you have a data set consisting of average seasonal temperatures. The temperatures are in degrees Fahrenheit, and you would like to convert them to degrees Celsius. It is simple to write a DATA step to do this conversion:

```
DATA CELSIUS;
SET TEMPS.SEASONAL;
ARRAY DEGF{4} WINTERF SPRINGF SUMMERF AUTUMNF;
ARRAY DEGC{4} WINTERC SPRINGC SUMMERC AUTUMNC;
DO I = 1 TO 4;
   DEGC{I} = (DEGF{I}-32)*5/9;
END;
RUN;
```

But what if you had twelve months instead of four seasons? Or 52 weeks? Or 366 days? Even if you are very proficient with your favorite text editor, the task will become very tedious quickly.

This is why arrays were developed for the SAS DATA step. An array is a convenient way of grouping a set of variables in a SAS data set, in order to perform similar processing on them. There are a number of ways that you can use this feature:

- To group like variables for similar processing
- To reshape a SAS data set, such as to convert observations to variables for subsequent processing in a procedure
- To store temporary values without creating SAS variables
- To get around the "one-dimensional" nature of SAS observations

We will review all of these in this paper.

A Simple Example Using Arrays
Let us begin by examining a simple ARRAY statement in detail:

```
ARRAY DEGF{4} WINTERF SPRINGF SUMMERF AUTUMNF;
```

- ARRAY identifies the ARRAY statement.
- DEGF is the name of the array.
- { 4 } defines how many variables are in the array.
- WINTERF SPRINGF SUMMERF AUTUMNF are the variables in the array.

After the ARRAY statement has been defined, an element of the array corresponds to a SAS variable in the array. For example referring to array element DEGF[2] is the same as referring to SAS variable SPRINGF. Combining the array of Fahrenheit variables with an array of Celsius variables and a DO loop, the above example becomes:

```
DATA CELSIUS;
SET TEMPS.SEASONAL;
ARRAY DEGF{4} WINTERF SPRINGF SUMMERF AUTUMNF;
ARRAY DEGC{4} WINTERC SPRINGC SUMMERC AUTUMNC;
DO I = 1 TO 4;
   DEGC{I} = (DEGF{I}-32)*5/9;
END;
RUN;
```

In this case, the DATA step actually increased in size. As the number of variables to process becomes larger, however, the value of arrays becomes more apparent. Imagine processing 52 variables rather than four.

More Details
As with most other SAS statements, the ARRAY statement has many options. Here are some of the more common ones.

Specifying Variable Names in an Array
Any form of SAS variable list can be used to specify which variables are in an array, including numbered range,
name range and special SAS name lists. Of these, numbered range lists (VAR1-VARN) are most commonly used. For example:

```
ARRAY TEMP{52} TEMP1-TEMP52;
```

If your array name is the prefix of your numbered range, you can abbreviate the ARRAY statement by dropping the variable names. The above statement is the same as:

```
ARRAY TEMP{52};
```

Letting the SAS System Count the Number of Variables in an Array
To tell the SAS System to count the number of variables in an array, rather than specifying a number yourself, you can place an asterisk between the braces instead of a number. For example:

```
ARRAY TEMP{*} TEMP1-TEMP52;
```

You may have to do this if the number of variables in the array changes for any reason, such as a macro specification. You may just have a lot of variables, and you don't want to count them.

The DIM function returns the number of variable (the DIMension) in the array. For example, consider a program where the variables to be converted to Celsius are defined in a macro variable:

```
%LET TEMPS = JAN FEB MAR;
ARRAY DEG{*} &TEMPS;
DO I = 1 TO DIM(DEG);

C(I)=LEFT(C(I));
END;
```

Initial values
If you are creating new variables in an array, you may want to set initial values for them. To do this, include the initial values in parentheses following the variable list, separated by blanks or commas. The following example defines an array of average temperatures, by month:

```
ARRAY AVG{12} AVG1-AVG12
(24 35 ... 24);
```

Initial values identified in an ARRAY statement are retained across observations, as if they were defined in a RETAIN statement.

Temporary Arrays
Arrays are particularly useful for defining temporary variables, to do interim calculations or to hold a set of numbers temporarily. As their name implies, the values of temporary arrays are not stored in the SAS data set. Using the above example, replacing the variable list with the special SAS name _TEMPORARY_ makes the list of average temperatures available only in this DATA step:

```
ARRAY AVG{12} _TEMPORARY_
(24 35 ... 24);
```

There is no need to specify variable names, as they will not be stored in the SAS data set.

You cannot use the asterisk to tell the SAS System to count the number of variables in a temporary array; you must tell SAS how many temporary variables there are.

Defining types and lengths of variables
If you are creating new variables in an array, you can define the type and length of the variables in the ARRAY statement. In the following examples, the first ARRAY statement defines a array of twelve numeric variables of length 4. The second ARRAY statement defines an array of six character variables of length 6.

```
ARRAY DEGC{12} 4 DEGC1-DEGC12;
ARRAY CHR{6} $6;
```

Beyond the Basics: More Details of the ARRAY Statement
Multidimensional Arrays
Variables in a SAS observation are stored as a vector of values, as a row in a table. Sometimes, the contents of variables are more easily thought of as a matrix of numbers. For example, we may have twelve months of temperature data for each of the years 1988 through 1990. Even though the SAS System stores these data elements
physically as a row, you can use a multidimensional array to refer to them as a matrix of values.

The ARRAY statement for our example becomes:

```
ARRAY AVG {3,12} Y88M1-Y88M12
   Y89M1-Y89M12 Y90M1-Y90M12;
```

You can then construct a logical set of DO loops to process the data:

```
DO I=1 TO 3;
   DO J=1 TO 12;
      AVG {I,J}=(AVG {I,J}-32)*5/9;
   END;
END;
```

This can be particularly useful for reshaping a SAS data set. See the example at the end of this paper.

Arrays elements are assigned in "row-major" order. That is, the SAS System fills the rows of the matrix first, then the columns. So ARRAY{1,1} contains the variable Y88M1, ARRAY{1,2} contains Y88M2, and so on. ARRAY{2,1} contains the variable Y89M1.

There is no limit on number of dimensions you can specify in a SAS array. After two dimensions, however, it becomes difficult to keep track of what is going on.

The DIM function returns the number of elements of the first dimension of a multidimensional array. To get the number of elements of any other dimension, add the number of the dimension to the DIM function. In the above example, DIM2(AVG) returns the value 12.

Different bounds

Normally, when you define an array, the range of array elements begins at 1 and ends at the number of variables in the array. Sometimes, it is helpful to use a different range. In the above example, it would be convenient if we could use the year number to identify the row of the array instead of the arbitrary numbers 1 to 3. The ARRAY statement provides a way to do this.

To use different bounds on array elements, specify two numbers in the ARRAY statement separated by a colon. The low bound is the first number, the high bound is the second number.

Using this feature, our ARRAY statement becomes:

```
ARRAY AVG {88:90,12} Y88M1-Y88M12
   Y89M1-Y89M12 Y90M1-Y90M12;
```

Use the LBOUND, HBOUND functions to determine low and high bounds of an array range. For the above ARRAY statement, LBOUND(AVG) returns 88.

A few notes on using different array bounds:

- The bounds must be integer values, but can be zero or negative. Of course, the low bound must be less than the high bound.
- The DIM function will return the number of elements in the array, not the high bound.
- As with the DIM function, use LBOUNDn and HBOUNDn to get the low and high bounds, respectively, of the nth dimension.
- If you allow the SAS System to name the variables, the numeric suffixes on the variables will still range from 1 to the number of variables, regardless of the low and high bounds. For example, the ARRAY statement

```
ARRAY YEAR {88:90};
```

will create variables named YEAR1-YEAR3, not YEAR88-YEAR90.

Using an Array as a Variable List

A useful feature of arrays is their ability to be used as a variable list after the array is defined. This is useful if your variable list has to be used more than once in a DATA step. The following example shows how an array can be used in an INPUT statement and as a function reference:

```
DATA TEMP;
   ARRAY Y88TEMPS {88:90,12} Y88M1-Y88M12;
   INPUT Y88TEMPS {*};
   Y88AVG = MEAN ( OF Y88TEMPS {*});
```

The ARRAYNAME[*] specification cannot be used with a_TEMPORARY_array.

Implicitly subscripted arrays

In previous versions of the SAS System, the syntax of ARRAY statements was different. Instead of a dimension, an array was associated with a subscript. References to the array used the subscript implicitly, but was not part of the array reference. If old programs you encounter use these kind of arrays, refer to pp. 297-299 and pp. 303-306 of the SAS Language: Reference, Version 6 Edition.

The following is an example of using implicitly subscripted arrays:

```
DATA CELSIUS;
   SET TEMPS.SEASONAL;
   ARRAY DEGF {I} WINTERF SPRINGF;
```
Do not use implicitly subscripted arrays in new programs. They may not be supported in future releases of SAS software.

The Down Side of Arrays
Once you see what a time-saving feature arrays are, you will be disappointed to find out that they do not work anywhere else but in a DATA step. You cannot store array references in a SAS data set. Neither can you use an array as a variable list in a PROC step. You might consider storing your variable list in a macro variable, for reference in a PROC step.

One other warning: remember that the SAS System stores all variables that are read in or created in a DATA step, unless otherwise instructed. That includes index variables such as I that are often used in DO loops when processing arrays. Remember to use a DROP or KEEP statement to make sure that the SAS data set contains only the variables you want to save.

Some Examples Using Arrays
Reshaping a SAS Data Set for Plotting
In the following example, we have a SAS data set with the variables YEAR and TEMP1-TEMP12. TEMP1-TEMP12 contain the average monthly temperature for the given year. We want to plot temperature by month, for each year. To do this, we must reshape the SAS data set, making variables for temperature and month. We will use an index named MONTH, since we have to create it for the plot anyway.

DATA PLOTTEMP;
  SET TEMPERAT;
  ARRAY T(*) TEMP1-TEMP12;
  DO MONTH = 1 TO 12;
    TEMP = T(MONTH);
    OUTPUT;
  END;
  KEEP YEAR MONTH TEMP;
RUN;
PROC GPLOT DATA=PLOTTEMP;
  PLOT TEMP*MONTH=YEAR;
RUN;

Report Writing using Multidimensional Arrays
In a previous example, we had a data set with 36 variables: average temperatures for each of twelve months, for the three years 1988 to 1990. We would like to print out the deviation from the monthly average temperature. We will take advantage of multidimensional arrays, different low and high bounds, temporary arrays and initial values.

DATA _NULL_;
  SET THREEYR;
  ARRAY TEMPS[88:90,12] Y88M1-Y88M12 Y89M1-Y89M12 Y90M1-Y90M12;
  ARRAY AVG[12] TEMPORARY_ (27 29 39 50 60 68 72 71 64 53 42 31);
  ARRAY DEV{12} DEV1-DEV12;
  ARRAY MONTHNAM{12} $3 MN1-MN12 ('Jan' 'Feb' 'Mar' 'Apr' 'May' 'Jun' 'Jul' 'Aug' 'Sep' 'Oct' 'Nov' 'Dec');
  FILE PRINT;
  PUT 'Deviations from Average Monthly Temperatures';
  PUT 'Year ' (MONTHNAM{*}) (8.2);
  DO YEAR=LBBOUND(TEMPS) TO HBOUND(TEMPS);
    DO MONTH=1 TO 12;
      DEV(MONTH) = TEMPS{YEAR,MONTH} - AVG(MONTH);
    END;
    PUT YEAR 4.2 (DEV{*}) (3.2);
  END;
RUN;

References


Acknowledgments
SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA

No endorsement of SAS software by Eastman Kodak Company is implied.