Introduction to Arrays

Bob Virgile
Robert Virgile Associates, Inc.

Overview

This paper explains the basics of defining and using an array. The information and examples will be useful to the programmer who is either unfamiliar with or confused by arrays. The basics are simple enough that most programmers can begin to successfully use arrays immediately after reading this paper.

This paper illustrates the most common ARRAY statement syntax and usage. For all the details you might ever want to know, see **SAS**® **Language: Reference, Version 6, First Edition**, pp. 160-171 and pp. 292-306.

What Is an Array?

An array means "a subset of the variables that make up one observation of a SAS data set." A sample data set might consist of 11 variables, with 7 of those variables making up an array.

```
Variables within
SAS data set CITIES:

 STATE
 BIRDS       }
 BEES        } These
 FLOWERS     } variables
 TREES       } make up
 SKY         } an Array.
 ABOVE       }
 LOVE        }
 INDEX
 CITY
 POP
```

By defining these seven variables as an array, the program can process the variables easily and economically. Usually the array helps when all its variables will be processed in a similar fashion. In the following example, all seven variables are being processed in exactly the same fashion. Therefore, this program is a prime candidate for constructing and using an array:

```
DATA NEW;
SET CITIES;
IF BIRDS   = . THEN BIRDS   = 0;
IF BEES    = . THEN BEES    = 0;
IF FLOWERS = . THEN FLOWERS = 0;
IF TREES   = . THEN TREES   = 0;
IF SKY     = . THEN SKY     = 0;
IF ABOVE   = . THEN ABOVE   = 0;
IF LOVE    = . THEN LOVE    = 0;
```

A revised program would place all seven variables into an array and then process all variables within the array:

```
DATA NEW;
SET CITIES;
ARRAY LYRICS {7} BIRDS BEES FLOWERS
                      TREES SKY ABOVE
LOVE;
  DO I = 1 TO 7;
     IF LYRICS{I} = .
     THEN LYRICS{I} = 0;
  END;
```

This revised program produces the same result more economically. First, the program is three lines shorter. Second, it becomes very clear that the program processes exactly seven variables. Therefore, it becomes easier to understand and maintain the second program. (Note the difference if the array contained 80 variables instead of 7. The second program would still contain six statements, although the ARRAY statement would be longer. The first program, however, would add 73 more statements.)

"Economical" does not mean the program requires less CPU time. If anything, arrays require slightly more CPU time. However, this is a minor expense compared to the savings in the length (and maintainability) of the program.

Basic Rules

The ARRAY statement defines which variables are included in the array. The statement appears within a DATA step and defines the array for the duration of that DATA step. Array definitions do not carry over from one DATA step to the next.

The same DATA step can contain many ARRAY statements.

The word "element" is frequently used to refer to a variable in an array. The previous array contained seven elements. (Technically, one variable could be two elements if the array statement were to list that variable twice.)

A single array cannot contain both character and numeric variables. This make sense since the whole purpose of the array is to process many variables in the same fashion. After all, how much sense would this statement make:

```
IF TREES = . THEN TREES = 0;
```

if TREES were a character variable?

Finally, arrays work with one observation at a time. They never compare information in one observation with information in another observation. If your program must make such a comparison, use other standard tools such as a RETAIN statement, the LAG function, or BY variables.

Syntax for the ARRAY Statement

The ARRAY statement supplies the following information:

1. A name for the array.

2. The number of variables in the array.

3. A list of the variable names.

Variations exist. For example, the array statement can omit the element names as long as it specifies the number of elements. The software then creates the element names by appending numbers to the name of the array. If the array were named TEST, for example, the software would create the elements TEST1, TEST2, TEST3, etc.

These are valid ARRAY statements:

```
ARRAY ELEMENTS {5} ELEMENT1-ELEMENT5;
ARRAY LYRICS {7} BIRDS BEES FLOWERS
                      TREES SKY ABOVE
LOVE;
```

Use any valid SAS name as the name of the array, but avoid:

1. The name of a variable in the SAS data set. This is an error.

2. The name of a SAS function (such as LENGTH, COMPRESS, or TRIM). This is not an error, but it does disable that function for the duration of the DATA step.

Next, specify the number of variables in the array, putting the number in curly brackets. The ARRAY statements above used {5} and {7} to indicate the number of elements. Parentheses (5) or square brackets [5] are also permitted.

Due to laziness or other more complex factors, the number of elements in the array may be unknown. The asterisk can replace the actual number in the ARRAY statement. The DIM function then becomes very useful; it counts the elements in an array. For example, in the following program:

```
DATA NEW;
SET CITIES;
ARRAY LYRICS {*} BIRDS BEES FLOWERS
                      TREES SKY ABOVE
LOVE;
  SIZE = DIM(LYRICS);
```

the variable SIZE has a value of 7 because the array LYRICS contains 7 elements.

Lastly, the ARRAY statement lists the names of all variables that make up the array. The two most common methods for listing variables are:

1. Naming each variable, as in the ARRAY statement above.

2. Specifying a numbered list. For example, ELEMENT1-ELEMENT5 means the five variables ELEMENT1, ELEMENT2, ELEMENT3, ELEMENT4, and ELEMENT5.

The SAS system supports other methods for specifying a list of variable names. However, these methods are complex and unnecessary 99% of the time. ARRAY statements can utilize two additional features. First, the statement may define a default length for new variables. If ELEMENT1-ELEMENT5 are character variables,

and NEWVAR has never been defined, these two sets of statements would both define NEWVAR as character with a length of 12:

```
ARRAY ADD1 {6} $ 12 ELEMENT1-ELEMENT5
                    NEWVAR;

LENGTH NEWVAR $ 12;
ARRAY ADD1 {6} ELEMENT1-ELEMENT5
                    NEWVAR;
```

Lastly, you may encounter implicitly subscripted arrays. The syntax varies slightly:

```
ARRAY LYRICS (_I_) BIRDS BEES FLOWERS
                   TREES SKY ABOVE
LOVE;
```

Parentheses (not brackets) now contain a variable name rather than a number or an asterisk. Slight differences in syntax will arise when referring to an element of an implicitly subscripted array. See the next section of this paper for details.

Implicitly subscripted arrays are not recommended style. Any program that uses them could also use regular (explicitly subscripted) arrays just as easily or more easily. (For that matter, any program that uses arrays could be written without arrays. However, it might become a much longer program.) These arrays are described here so that you will recognize them when you see them, not to encourage you to use them.

Referring to an Array Element

Later statements in the DATA step refer to an array element by referring to the array name rather than the variable name. One previous program used this technique:

```
DATA NEW;
SET CITIES;
ARRAY LYRICS {7} BIRDS BEES FLOWERS
                   TREES SKY ABOVE
LOVE;
DO I = 1 TO 7;
    IF LYRICS{I}=. THEN LYRICS{I}=0;
END;
```

LYRICS{I} refers to one variable in the array, depending on the current value for the variable I. When I=4, LYRICS{I} means the variable TREES (the fourth element in the array). When I=7, LYRICS{I} means the variable LOVE (the seventh

element in the array). Since LYRICS contains seven variables, the statement:

```
DO I = 1 TO 7;
```

processes each element in the array, one by one.

If the number of array elements were unknown, the DIM function could count them. The following program produces an identical result:

```
DATA NEW;
SET CITIES;
ARRAY LYRICS {*} BIRDS BEES FLOWERS
                   TREES SKY ABOVE
LOVE;
DO I = 1 TO DIM(LYRICS);
    IF LYRICS{I}=. THEN LYRICS{I}=0;
END;
```

Finally, implicitly subscripted arrays use only the array name to refer to an element.

```
DATA NEW;
SET CITIES;
ARRAY LYRICS (_I_) BIRDS BEES FLOWERS
                   TREES SKY ABOVE
LOVE;
DO _I_ = 1 TO 7;
    IF LYRICS=. THEN LYRICS=0;
END;
```

or

```
DO OVER LYRICS;
    IF LYRICS=. THEN LYRICS=0;
END;
```

Herein lies the lone advantage of implicitly subscripted arrays over explicitly subscripted arrays. The DO OVER syntax (illegal with implicitly subscripted arrays) conveniently processes every element in the implicitly subscripted array. Still, implicitly subscripted arrays are not recommended. They are described here so that you will recognize them when you see them.

Usefulness of Arrays: A Sample Problem

In this sample problem, the SAS data set OLD contains 20 character variables named LINE1 through LINE20. Each has a length of 50. These variables contain text information and are intended to be printed one beneath the next with a statement like:

```
    PUT LINE1 / LINE2 / LINE3 / LINE4 /
        LINE5 / LINE6 / LINE7 / LINE8 /
        LINE9  / LINE10 / LINE11 /
        LINE12 / LINE13 / LINE14 /
        LINE15 / LINE16 / LINE17 /
        LINE18 / LINE19 / LINE20;
```

The three variables LINE10 through LINE12 always contain the following text:

LINE10 reads:

SUMMARY STATISTICS, ALL DIVISIONS

LINE11 reads:

(THESE ARE PRELIMINARY FIGURES ONLY.

LINE12 reads:

FINAL NUMBERS WILL ARRIVE SOON.)

Now another month has passed, the final numbers are in, and the note in parentheses (LINE11 and LINE12) no longer applies. With or without arrays, a program should blank out the note in parentheses. Without arrays, the program would be:

```
    DATA NEW;
    SET OLD;
    LINE11=' ';
    LINE12=' ';
```

With arrays, the program would be:

```
    DATA NEW (DROP=I);
    SET OLD;
    ARRAY LINES {20} LINE1-LINE20;
    DO I=11 TO 12;
        LINES{I}=' ';
    END;
```

The second program is longer and more complex. So why bother with arrays?

As the program's objective becomes more and more complex, arrays can simplify the program considerably. As an example, consider one shortfall of the previous programs. When printing the report, there would now be two blank lines in the middle. A more complex objective would be to remove the note, without leaving any blank lines in the middle. Without arrays, the program would be:

```
    DATA NEW;
    SET OLD;
    LINE11 = LINE13;
    LINE12 = LINE14;
    LINE13 = LINE15;
    LINE14 = LINE16;
    LINE15 = LINE17;
    LINE16 = LINE18;
    LINE17 = LINE19;
    LINE18 = LINE20;
    LINE19 = ' ';
    LINE20 = ' ';
```

Using arrays, the program becomes:

```
    DATA NEW (DROP=I);
    SET OLD;
    ARRAY LINES {20} LINE1-LINE20;
    DO I=11 TO 18;
        LINES{I} = LINES{I+2};
    END;
    DO I=19 TO 20;
        LINES{I}=' ';
    END;
```

This program is shorter and more flexible. If the number of variables increases from 20 to 50, the first program (without arrays) would have to add 30 lines. But this program (with arrays) would remain virtually unchanged.

Let's add one more wrinkle to the original problem. Suppose the three key lines don't necessarily begin with LINE10. The text:

    SUMMARY STATISTICS, ALL DIVISIONS

appears anywhere from LINE5 through LINE15. Now the program must first locate the text and then change all subsequent variables. The program with arrays takes 12 statements:

```
DATA NEW (DROP=I START);
SET OLD;
ARRAY LINES {20} LINE1-LINE20;
DO I=5 TO 15;
    IF LINES{I}=
    'SUMMARY STATISTICS, ALL DIVISIONS'
    THEN START=I+1;
END;
DO I=START TO 18;
    LINES{I}=LINES{I+2};
END;
DO I=19 TO 20;
    LINES{I}=' ';
END;
```

The first DO group locates which of the variables among LINE1-LINE15 contains the key text. The program will change the values of all "subsequent" variables. For example, if LINE8 contains the key text, then the program will change the values of LINE9 through LINE20. Therefore, the program notes (and assigns to the variable START) the number of the first variable to be changed. In this case, START would receive a value of 9. The last two DO groups work as before, modifying values of the variables.

The same program without arrays would take about 80 lines! (Being extremely clever, you could write this program in 30 lines without arrays. But if you are that clever, you don't need to be reading this paper. I will honor all written requests for 30-line solutions.)

For the Future

In most applications which use arrays, arrays are 10% of the pie and other tools make up 90%. Therefore, knowledge of arrays must be combined with knowledge of other tools. DATA step tools, especially different forms of the DO statement, are most important.

Consider the sample application, for example. It eliminated two lines, shifting a block of text "up" by two lines. In practice, this program is likely to be part of a more complex system that uses a much greater variety of tools. A similar program might "insert" blank lines by shifting a block of text "down." A third program might change the order of the variable values, equivalent to "moving a paragraph." Finally, add macro language statements to generate menus for running these programs interactively. The menus might allow users to make requests such as:

- For observation 5, insert 3 blank lines after line 4.

- For observation 25, move lines 14 through 18 to after line 3.

To a small extent, a SAS-based system is now operating as a word processing system! But arrays constitute one of many tools needed to accomplish this.

In your future programs, more complex uses for arrays may come into play, including:

1. Creating arrays to hold sets of constants instead of sets of variables.

2. Relating multiple arrays in one DATA step. For example, based on the 80 variables HEIGHT1 through HEIGHT80 and 80 more variables WIDTH1 through WIDTH80, compute 80 variables AREA1 through AREA80 (AREA1 = HEIGHT1 * WIDTH1, etc.).

3. Defining multidimensional arrays.

4. Reading or writing array elements with INPUT or PUT statements.

These types of capabilities are invaluable for solving certain problems. However, for most problems, the introductory concepts and techniques in this paper will be more than sufficient.

The author welcomes questions, comments, and requests for 30-line solutions. Feel free to call or write:

Bob Virgile
Robert Virgile Associates, Inc.
3 Rock Street
Woburn, MA 01801
(781) 938-0307

SAS is a registered trademark of SAS Institute Inc.