Technical Paper

Using Arrays in SAS® Programming

Table of Contents

Overview **1**
Basic Syntax of the ARRAY Statement **1**
Basic Array Example: Calculating Net Income 2
Using Arrays with Functions and Operators 4
Common Tasks and Examples Using Arrays **6**
Assigning Initial Values to Array Variables or Elements 6
Specifying Lower and Upper Bounds of a Temporary Array 9
Creating a Temporary Array 9
Using SAS Variable Lists with Arrays 12
Expanding and Collapsing Observations 14
Finding a Minimum or Maximum Value As Well As the Corresponding Variable Name 17
Conclusion **18**
References **18**

Overview

DATA step programmers use arrays to simplify their code, which results in programs that are frequently more efficient and less error-prone. Consider, for example, a revenue-and-expense data set (Rev_Exp) that contains 24 monthly variables for a single year, 12 variables for revenues (Rev1–Rev12), and 12 variables for expenses (Exp1 - Exp12). To calculate the net income for each month, the SAS program needs 12 statements:

```
net_inc1 = rev1 - exp1;
net_inc2 = rev2 - exp2;
. . .eight other similar statements. . .
net_inc11 = rev11 - exp11;
net_inc12 = rev12 - exp12;
```

This method for calculating the net income is repetitive. What if the data set contains monthly data for 3 years, or even 10 years? As the amount of data increases, more statements are required to calculate net income for each month. As a result, the process becomes more tedious and error prone. Arrays can be used to perform these calculations with far fewer statements.

Keep these two points in mind as you explore the use of arrays:

- In nearly all cases, code that is written with arrays can also be written without arrays.
- Arrays simply provide an alternative method for referring to a variable rather than using the name of the variable.

Basic Syntax of the ARRAY Statement

To use arrays in SAS code, first make sure that you understand the basic syntax of the SAS ARRAY statement. This section describes the ARRAY statement and provides a number of examples to illustrate its use. Subsequent sections progress to more complex statements and examples.

In its simplest form, the ARRAY statement consists of the keyword ARRAY followed by the name of the array:

ARRAY *array-name*[ ];

The array name is followed by either a pair of parentheses ( ), braces { }, or square brackets [ ]. This document uses square brackets [ ].

By specifying a constant value within the brackets, you can specify the number of variables or elements that are to be associated with the array. As shown in the following example, you can follow the brackets with a variable list that you want to associate with or assign to the name of the array:

```
array revenue[12] rev1-rev12;
```

In this statement, the array REVENUE has 12 variables (Rev1 – Rev12) associated with it. Frequently, such an array is referred to as having 12 *elements*.

Variables that are associated with an array have certain characteristics:

- All variables that are associated with an array must be of the same type, either character or numeric. As a result, arrays are typically referred to as either *character arrays* or *numeric arrays*.
- Variables that are associated with an array do not have to be already existing variables. If they do not exist within a program data vector (PDV) when the ARRAY statement is compiled, SAS creates them for you.
- Variables that are not previously defined as character variables will default to numeric variables unless they are defined as character variables within the ARRAY statement. To define character variables within the ARRAY statement, place a dollar sign ($) after the brackets and before any of the variables, as illustrated in this example:

```
array my_name[3] $ first middle last;
```

By default, array variables or other elements in the array have a length of 8 bytes. To specify a different length, include the desired length after the $ for character arrays and after the brackets for numeric arrays, as shown in these statements:

```
array name[3] $10 first last middle;
array weight[*] 5 weight1 - weight10;
```

Notice the asterisk (*) inside the brackets in the WEIGHT array above. SAS must be able to determine the number of elements or variables in the array when it compiles the code. SAS determines this either by using the constant value that is specified in the brackets or by counting the number of variables in the variable list. When you want SAS to use the variable list, place an asterisk in the brackets. Alternatively, you can leave off the variable list and specify the constant value between brackets, as shown in this example:

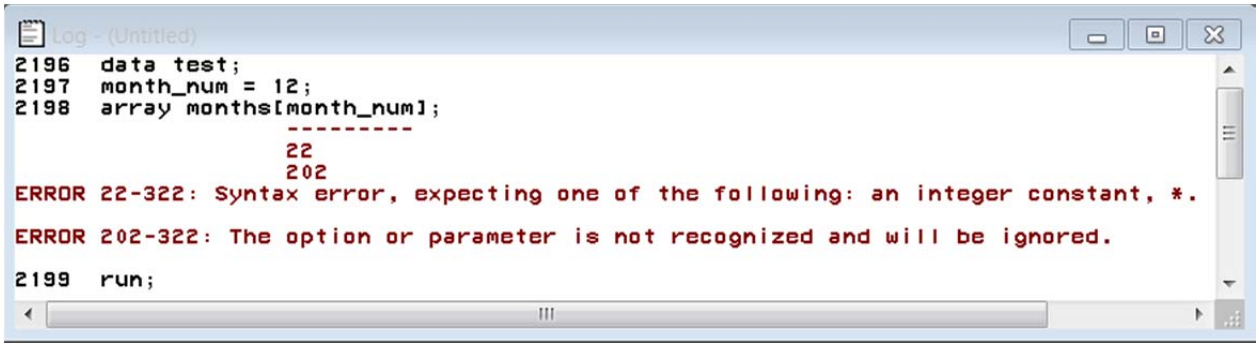
```
array weight[10] 5;
```

Because a variable list is not specified in this example, SAS uses the name of the array (WEIGHT) and adds a numeric suffix from 1 to 10 to associate or create the specified number of variables with the array.

Note: SAS must be able determine the number of elements or variables in the array when it compiles the code. Therefore, you cannot place a variable name in the brackets, as illustrated in the following statement, with the intent of SAS referencing the name to determine the number of elements:

```
array months[month_num];
```

SAS cannot determine the value of the Month_Num variable until run time. Using such a statement results in a syntax error, as shown here:



Basic Array Example: Calculating Net Income

This section presents a basic example that uses arrays. This example assumes a revenue-and-expense data set (Rev_Exp) that contains 24 monthly variables: 12 variables for revenue (Rev1–Rev12) and 12 variables for expenses (Exp1–Exp12).

The task is to calculate the net income (revenue minus expenses) for each of the 12 months, thereby creating 12 new net-income variables (Net_Inc1–Net_Inc12).

```
data net_income;
   set rev_exp;
   array revenue[*] rev1-rev12;
   array exp[12];
   array net_inc[12];
   do i=1 to 12;
      net_inc[i]=revenue[i] - exp[i];
   end;
run;
```

This example defines three arrays:

- The first ARRAY statement defines an array called REVENUE and associates the existing 12 variables (Rev1–Rev12) with the array.
- The second ARRAY statement defines an array called EXP. A variable list is not provided for this array, so SAS uses the array name and adds a numeric suffix (from 1–12) to associate the existing variables (Exp1 – Exp12) with the array.
- The third ARRAY statement defines an array called NET_INC. A variable list is not provided for this array, so SAS adds a suffix from 1–12 to the array name to associate the variables Net_Inc1–Net_Inc12 with the array. These variables do not exist in the Rev_Exp data set, so they are created as new variables in the DATA step.

After the arrays are defined, the iterative DO group iterates 12 times. For each iteration, the value of I increases incrementally by 1, from **1** to **12**. During each iteration, SAS uses the name of the array and the value of I to reference a specific element or variable in each array.

During the first iteration, the SAS statement uses Rev1 and Exp1 and uses them to calculate the net income and assign that value to Net_Inc1. SAS starts with this statement:

```
net_inc[i] = revenue[i] - exp[i];
```

Because the value of I is **1** during the first iteration, the statement effectively becomes the following:

```
net_inc[1]= rev[1] - exp[1];
```

Finally, the references to the first elements in each of the arrays results in this statement:

```
net_inc1 = rev1 – exp1;
```

These iterations continue until SAS calculates Net_Inc12, as shown in this statement:

```
net_inc12 = rev12 - exp12;
```

In the Overview of this document, two points were emphasized:

- SAS programs that are written with arrays can be written without arrays. For example, suppose you have these three statements that use arrays:

  ```
  do i=1 to 12;
    net_inc[i] = revenue[i] - exp[i];
  end;
  ```

  The same task can be accomplished without arrays by using these 12 statements:

  ```
  net_inc1  = rev1 - exp1;
  net_inc2  = rev2 - exp2;
   . . . eight similar statements . . .
  net_inc11  = rev11 - exp11;
  net_inc12  = rev12 - exp12;
  ```

- Arrays provide an alternative method of referring to variables. Instead of referring to the first revenue variable as Rev1, you can refer to it by using the array name and an index into the array, such as REVENUE[I] (assuming that I has a value of **1**).

Using Arrays with Functions and Operators

SAS has many functions and operators that you can use with arrays to perform common tasks. This section discusses the following functions and operators as they relate to arrays:

- DIM function
- OF operator
- IN operator
- VNAME function

DIM Function

One of the most common tasks involving arrays is to iterate (or, to loop) through each element of an array by using a DO group and then performing an operation on each element. The basic example in the previous section uses the following DO group, which performs 12 iterations:

```
do i=1 to 12;
   net_inc[i]=revenue[i]-exp[i];
end;
```

The purpose of the DO group is to access each of the 12 elements in the arrays. In this DO group, the iteration value **1** is the START argument, whereas **12** is the STOP argument. With this method, you must change the STOP argument in your DO group whenever the number of elements in the array changes.

The DIM function presents a more dynamic way to determine the STOP argument, as illustrated by the following example:

```
do i=1 to dim(net_inc);  /* The DIM function returns a value of 12. */
  net_inc[i]=revenue[i]-exp[i];
end;
```

When you specify the array name as the single argument for the DIM function, the function returns the number of elements in the array. This example using the DIM function returns the same STOP value (**12**) as does the example in the previous section "Basic Array Example: Calculating Net Income." However, by using the DIM function, you do not have to update the STOP value if the number of array elements changes.

OF Operator

It is common to perform a calculation using all of the variables or elements that are associated with an array. For example, the code shown previously in the section "Basic Array Example: Calculating Net Income" calculates the net income for each month. However, suppose that you want to sum each of the 12 monthly net incomes. To do that, you pass the name of the array using the [*] syntax to the SUM function by using the OF operator.

```
sum_net_inc=sum(of net_inc[*]);
```

You can use the OF operator with functions such as MEAN, MIN, and MAX or any other functions that operate on a list of variables.

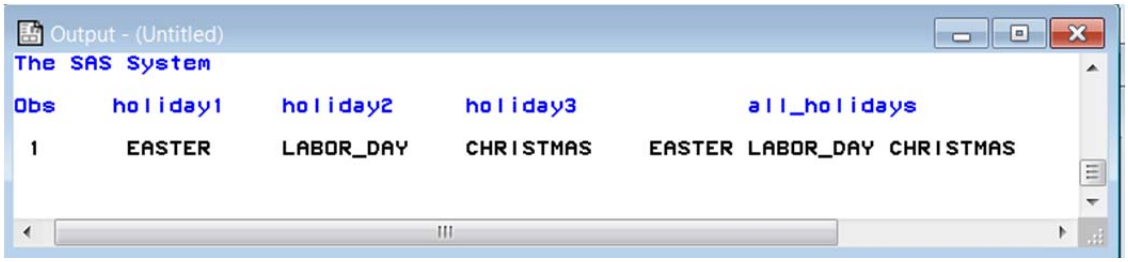
- `mean_net_inc=mean(of net_inc[*]); /* Arithmetic mean (average) */`
- `min_net_inc=min(of net_inc[*]);   /* Smallest value */`

- `max_net_inc=max(of net_inc[*]);  /* Largest value  */`
- `call missing(of net_inc[*]);  /* Call routine that assigns a missing */`
  `/* value to all elements  */`

You can pass arrays to functions (such as concatenation functions) that expect character arguments. The following example uses the CATX function to create a character string that contains the concatenation of three holidays (Easter, Labor Day, and Christmas).

```
data holidays;
   input (holiday1-holiday3) (: $9.);
   datalines;
EASTER LABOR_DAY CHRISTMAS
;
run;

data find_christmas;
   set holidays;
      /* Note that the $ sign is not necessary within the ARRAY statement */
      /* because the HOLIDAY variables are defined previously as          */
      /* character variables.                                             */
   array holiday_list[*] holiday1-holiday3;
   all_holidays=catx(' ', of holiday_list[*]);
run;

proc print;
run;
```

This code generates the following output:



IN Operator

The IN operator can test whether a specific value occurs within any element or variable of an array. You can use this operator with numeric as well as character arrays. When you use an array with the IN operator, only the name of the array is used. For example, you can write IF statements similar to the following:

Example 1

```
   /* This example uses the previously defined array NET_INC. */
if 1234 in net_inc then put 'found';
```

Example 2

```
   /* This example uses the previously defined array HOLIDAY_LIST. */
if 'CHRISTMAS' in holiday_list then put 'found';
```

VNAME Function

You can use the VNAME function to identify the name of a variable that is associated with an element of an array. The name of the array, along with the element number that appears within brackets, is passed as the single argument to the VNAME function, as shown in this example:

```
array my_array[*] A B C;
i=2;
var_name=name(my_array[i]);
```

In this case, the value of the VAR_NAME= variable is **B**.

Common Tasks and Examples Using Arrays

Assigning Initial Values to Array Variables or Elements

For some applications, it can be beneficial to assign initial values to the variables or elements of an array at the time that the array is defined. To do this, enclose the initial values in parentheses at the end of the ARRAY statement. Separate the values either with commas or spaces and enclose character values in either single or double quotation marks. The following ARRAY statements illustrate the initialization of numeric and character values:

```
array sizes[*] petite small medium large extra_large (2, 4, 6, 8, 10);
array cities[4] $10 ('New York' 'Los Angeles' 'Dallas' 'Chicago');
```

You can also initialize the elements of an array with the same value using an iteration factor, as shown in the following example that initializes 10 elements with a value of **0**:

```
array values[10] 10*0;
```

When elements are initialized within an ARRAY statement, the values are automatically retained from one iteration of the DATA step to another; a RETAIN statement is not necessary.

Example: Determining Whether Antibiotics Are Referenced in Patient Prescriptions

Consider the following example in which you have a data set with a character variable that contains comments about drug prescriptions and dosages. In this example, the task is to determine whether certain antibiotics are referred to in the

prescriptions. An array is initialized with a list of seven antibiotics. Then a DO group loops through the array, checking for each of the antibiotics.

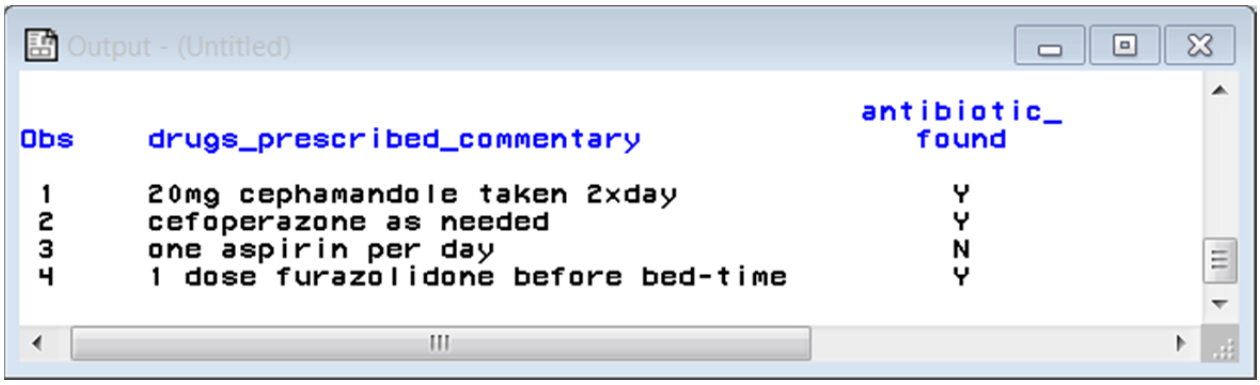
Key Tasks in This Example

- Initialize a character array with the names of antibiotics.
- Initialize a flag variable to indicate that no antibiotics are found.
- Use the INDEXW and UPCASE functions to search for each antibiotic.
- Reset the value of the flag variable if an antibiotic is found.

Program

```
data drug_comments;
   input drugs_prescribed_commentary $char80.;
   datalines;
20mg cephamandole taken 2xday
cefoperazone as needed
one aspirin per day
1 dose furazolidone before bed-time
;
run;

data find_antibiotics;
   set drug_comments;
      /* Initialize a character array with the names of antibiotics. */
   array antibiotics[7] $12 ('metronidazole', 'tinidazole', 'cephamandole',
                             'latamoxef', 'cefoperazone', 'cefmenoxime',
                             'furazolidone');
      /* Initialize a flag variable to N, meaning that no antibiotic */          /*
      is found.                                                      */
   antibiotic_found='N';
      /* Cycle through the array of antibiotics. */
   do i=1 to dim(antibiotics);
     /* Use the INDEXW and the UPCASE functions to check for each drug. */
   if indexw(upcase(drugs_prescribed_commentary), upcase(antibiotics[i]))
     then do;
             /* When an antibiotic is found, change the flag variable to Y,*/
             /* meaning that an antibiotic is found.                       */
        antibiotic_found = 'Y';
             /* No need to continue checking because an antibiotic is */
             /* found. Terminate the DO group.                        */
         leave;
       end;
   end;
   keep drugs_prescribed_commentary antibiotic_found;
run;

proc print;
run;
```

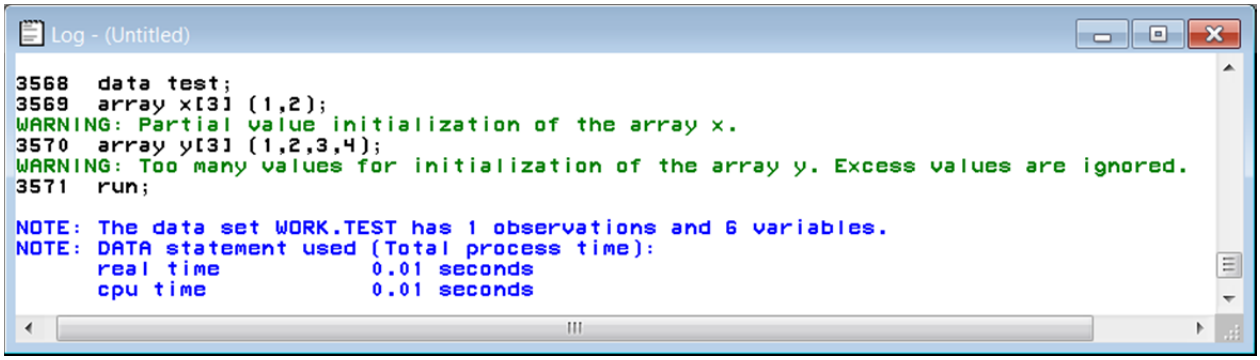
The PRINT procedure in this example generates the following output:



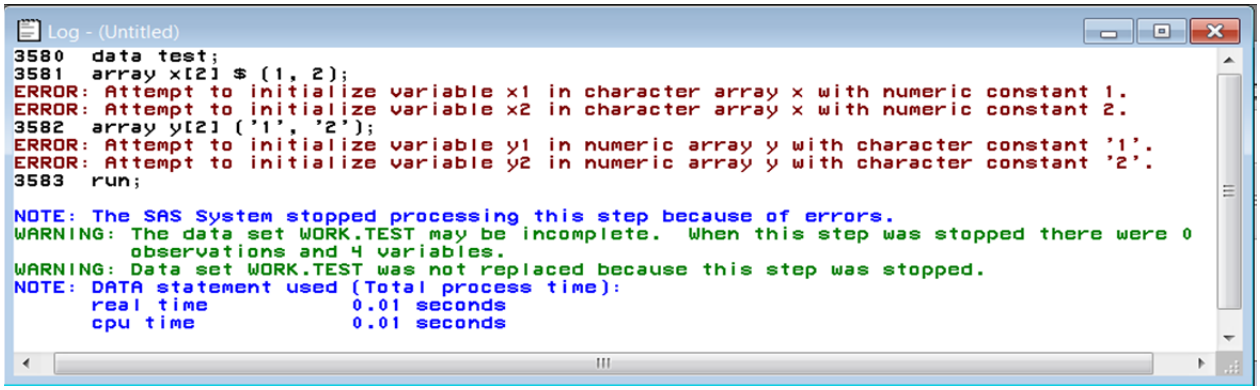
If you inadvertently initialize more elements or variables than exist in an array, SAS generates a warning. For example, the following DATA step defines two arrays, each with three elements or variables.

```
data test;
   array x[3] (1,2);
   array y[3] (1,2,3,4)
run;
```

Both of these definitions also initialize the values of the array elements. The first definition initializes only two of the three elements, whereas the second definition attempts to initialize four elements. As a result, the SAS log displays the following, self-explanatory warnings:



It is important to remember that all of the variables associated with an array must be of the same type. If you write your code in such a way that SAS attempts to initialize it or assign a numeric value to a character array or attempts to initialize or assign a character value to a numeric array, errors similar to the following occur:



Specifying Lower and Upper Bounds of a Temporary Array

The arrays discussed previously in this paper use either a single value or an asterisk within the array brackets. When this is done, by default, the lower bound is **1** and the upper bound is the number of elements in the array. For example, in both of the following ARRAY statements, the lower bound is **1** and the upper bound is **5**:

```
array years[5] yr2006-yr2010;
array years[*] yr2006-yr2010;
```

This means that the first element is referred to as element 1, the second element as element 2, and so on.

For some applications, DATA step code might be more efficient or intuitive if an element is referenced by another value. For example, in the ARRAY statements above, it might be useful to refer to the first element as element 2006, the second element as element 2007, and so on. To do this, specify the lower and upper bounds using a colon between the two values, as shown in this example:

```
array years[2006:2010] yr2006 - yr2010;
```

When YR=2006, this statement assigns a value of **200** to the first element of the array, variable Yr2006.

```
years[yr]=200;
```

When YR=2010, this statement assigns the value of element 2010 (the fifth element of the array) to the variable X.

```
x=years[yr];
```

It is important to remember that when lower and upper bounds are specified but the names of the variables are omitted, SAS does not create variable names with the bound values as the suffix. For example, consider the following array:

```
array years[2006:2010];
```

Such an ARRAY statement creates or associates the variables Years1, Years2, Years3, Years4, and Years5 with the array. It will not create or associate the variables Years2006, Years2007, Years2008, Years2009, and Years2010 with the array.

An example of specifying lower and upper bounds of an array is included in the following section.

Creating a Temporary Array

Occasionally, you might need an array to hold values temporarily for subsequent calculations but have no need for the actual variables. In such cases, it is beneficial to create arrays in which specific variables are never created or associated with

elements of an array. Arrays of this nature are referred to frequently as non-variable-based arrays. To define such an array, use the _TEMPORARY_ keyword, as shown in this example:

```
array my_array[25] _temporary_;
```

Temporary arrays can be either numeric or character. Just as you do with non-temporary character arrays, you create a temporary character array by including the dollar sign ($) after the array brackets.

```
array my_array[25] $ _temporary_;
```

When you reference elements of a temporary array, you must always use the array name. Since there are no variables associated with the array (non-variable-based), a reference to MY_ARRAY1 would not refer to the first element of the array. In addition, when defining a temporary array, you must always explicitly specify the number of elements within the array using a constant value within the array brackets. You cannot use an asterisk (*).

Temporary arrays have two other characteristics:

- Values within the array elements are retained automatically between the iterations of the DATA step – there is no need to use a RETAIN statement.
- Values within the array elements are never written to the output data set.

Example: Summarizing Medication Dosages for Patients

In some cases, data must be summarized before it can be analyzed. In this next example, the doses of medication for each patient must be summarized by date in order to determine the maximum dosage for any particular date.

Because this example uses SAS dates, a quick refresher about how SAS stored dates might be worthwhile. The actual value that is stored in a SAS date variable is simply an integer that represents the number of days since January 1, 1960. For example, the actual value of a SAS date that represents November 15, 2011 is **18946** because there are 18946 days between January 1, 1960 and November 15, 2011. You can associate a SAS format with a SAS date variable so that the date can be displayed in a conventional manner such as 11/15/2011 or 15NOV2011. A string of characters such as 11/15/2011 can be interpreted or read into a SAS date variable using a SAS informat such as MMDDYY10. However, neither a format nor an informat changes the actual value that is stored in the associated variable.

This example uses the SAS date integers as the lower and upper bounds of an array to facilitate the summarization of data by date.

Key Tasks in This Example

- Sort the data by patient ID and dates.
- Identify the earliest and latest dates by using the SQL procedure to assign these values to the macro variables FIRST_DATE and LAST_DATE.
- Reference these macro variables as the lower and upper bounds when defining a temporary array.
- Clear any accumulated values within the array at the beginning of each patient group.
- Accumulate dosages by date.
- Use the MAX function with the array to determine the maximum dosage for each date.

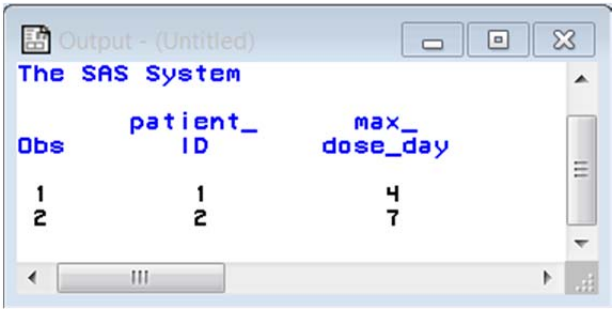
Program

```
data patient_medication;
   input patient_ID Medication $ Dose (Date_begin Date_end) (: mmddyy10.);
   format Date_begin Date_end mmddyy10.;
   datalines;
1 A 3 05/08/2009 06/09/2010
1 B 1 04/04/2009 12/12/2009
2 X 5 06/08/2009 09/09/2010
2 Y 2 08/04/2010 10/10/2010
;
run;
```

(code continued)

```
   /*Sort the data set by patient ID and dates.*/
proc sort data=patient_medication;
   by patient_ID Date_begin Date_end;
run;

   /* Use the SQL procedure to identify the earliest beginning date      */
   /* and the latest ending date within the data set. Assign the values  */
   /* to the macro variables, FIRST_DATE and LAST_DATE.                  */
proc sql noprint;
   select min(Date_begin), max(Date_end) into :first_date, :last_date
   from patient_medication;
quit;

data max_drug;
   set patient_medication;
   by patient_ID Date_begin Date_end;
      /* Reference the macro variables that are created in previous the   */
      /* PROC SQL step to define the lower and upper bounds of an array.  */
      /* Create a temporary array because a variable-based array is not   */
      /* needed and the values need to be retained.                       */
   array drug_day[&first_date : &last_date] _temporary_;
      /* Use the CALL MISSING routine to clear the accumulated values at  */
      /* the beginning of each patient group.                             */
   if first.patient_ID then call missing(of drug_day[*]);
      /* For each drug, loop from the beginning date to the ending date.  */
      /* Use the date value to accumulate the dosage for each date (an    */
      /* element of the array).                                           */
   do dt=Date_begin to Date_end;
       drug_day[dt]+dose;
   end;
      /* After processing the last observation for a patient, use the MAX  */
      /* function within the array to determine the maximum dosage on any  */
      /* particular date.                                                  */
   if last.patient_ID then do;
      max_dose_day=max(of drug_day[*]);
      output;
   end;
   keep patient_ID max_dose_day;
run;

proc print;
run;
```

This program generates the following output:



Using SAS Variable Lists with Arrays

A *SAS variable list* is an abbreviated method of referring to a list of variable names. Two variable lists, _NUMERIC_ and _CHARACTER_, are especially relevant for arrays. As the names imply, these variable lists reflect all of the numeric or character variables that are defined in the current DATA step when the variable list is specified. You can use these variable lists in the ARRAY statement, as shown here:

```
array my_nums[*] _numeric_;
array my_chars[*] _character_;
```

These variable lists are especially helpful if you want your array to contain all of the numeric or character variables in a SAS data set referenced in a SAS statement that reads the data set. It is important to emphasize that the variable list refers to all variables of that type that are previously defined in the DATA step, not just those that exist in a previously referenced data set.

Example: Changing Missing Values in All Numeric Variables to Zero

Applications frequently need to perform a task on every numeric or every character variable in a SAS data set. For example, an application might consider missing values for numeric variables to be inappropriate. In this case, the application requires that missing values be changed to zero. You can easily program this task by using the _NUMERIC_ variable list rather than specifying the actual variable names. This method is particularly useful when the data set contains too many numeric variables to list practically in the ARRAY statement.

Key Tasks in This Example

- Use the _NUMERIC_ variable list to associate all of the numeric variables with the array.
- Loop through the array and change the missing values to **0**.

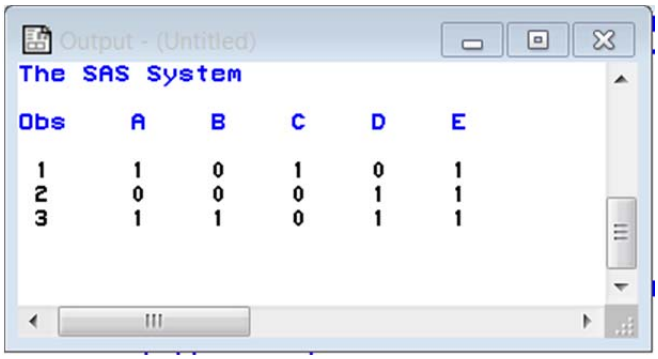
Program

```
data test;
   input A B C D E;
   datalines;
1 . 1 0 1
0 . . 1 1
1 1 0 1 1
;
run;
```

(code continued)

```
data test;
   set test;
      /* Use the _NUMERIC_ variable list to associate all of */
      /* the numeric variables with an array.                */
   array vars[*] _numeric_;
      /* Loop through the array, changing all missing values to 0. */
   do i=1 to dim(vars);
      if vars[i]= . then vars[i]=0;
   end;
   drop i;
run;

proc print;
run;
```

This example generates the following output:



It is important to emphasize that the _NUMERIC_ and _CHARACTER_ variables lists reflect those numeric or character variables that are defined when the _NUMERIC_ and _CHARACTER_ variables lists are first referenced in a SAS statement. They do not necessarily reflect every variable of that type in the DATA step nor only the variables defined in a previously referenced data set.

The following example uses the DIM function to show how two different arrays, both defined with the _NUMERIC_ variable list at different points in the code, contain a different number of variables.

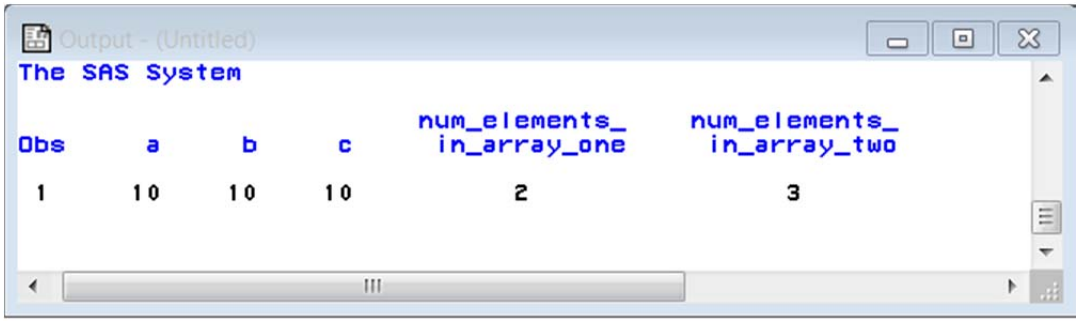
Program

```
data test;
   a-10;
   b=10;
      /*The array ONE is defined after two numeric variables are defined.    */
   array one[*] _numeric_;
   c=10;
      /* The array TWO is defined after three numeric variables are defined. */
   array two[*] _numeric_;
```

(code continued)

```
      /* Use the DIM function to determine the number of elements in */
      /* the array.                                                  */
   num_elements_in_array_one = dim(one);
   num_elements_in_array_two = dim(two);
run;

proc print;
run;
```

This code generates the following output:



Expanding and Collapsing Observations

Two common tasks that arrays frequently help with are expanding single observations into multiple observations and collapsing multiple observations into a single observation. In this context, a single observation has only one variable that represents an ID of some nature and then several variables that represent similar data, such as dates. In expanding this single observation, the intent is to create one observation for each date value, with each observation having the same ID value. Similarly, collapsing multiple observations into a single observation suggests creating multiple variables for each of the dates from the multiple observations.

Examples: Creating Multiple Observations for Each Patient Visit and Creating a Single Observation Containing All Visits

These examples start with a data set that has one observation per patient ID (Patient_ID) with several variables that represent the multiple dates on which visits to the doctor occurred. The first example shows how to expand each single observation into multiple observations, one for each visit. The subsequent example shows how to collapse those multiple observations back into a single observation.

Key Tasks in This Expansion Example

- Define an array for the visits.
- Use a DO group to iterate through the array and assign each visit to a new Date variable.
- Generate one observation for each visit.

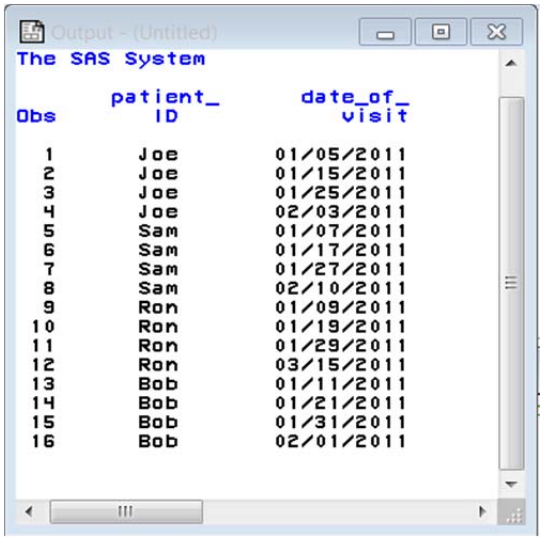
Program

```
data patient_visits;
   input patient_ID $ (visit1 – visit4) (: mmddyy10.);
   format visit1 – visit4 mmddyy10.;
```

(code continued)

```
   datalines;
Joe 01/05/2011 01/15/2011 01/25/2011 02/03/2011
Sam 01/07/2011 01/17/2011 01/27/2011 02/10/2011
Ron 01/09/2011 01/19/2011 01/29/2011 03/15/2011
Bob 01/11/2011 01/21/2011 01/31/2011 02/01/2011
;
run;

data expand;
   set patient_visits;
      /* Define an array to contain the visits. */
   array visit[4];
      /* Loop through the array, assigning each element (visit) */
      /* to the Date_of_Visit variable and then outputting it.  */
   do i=1 to dim(visit);
      date_of_visit = visit[i];
      output;
   end;
      /* Format and drop variables, as desired. */
   format date_of_visit mmddyy10.;
   drop visit1-visit4 i;
run;

proc print;
run;
```

This example generates the following output:



Key Tasks in This Collapse Example

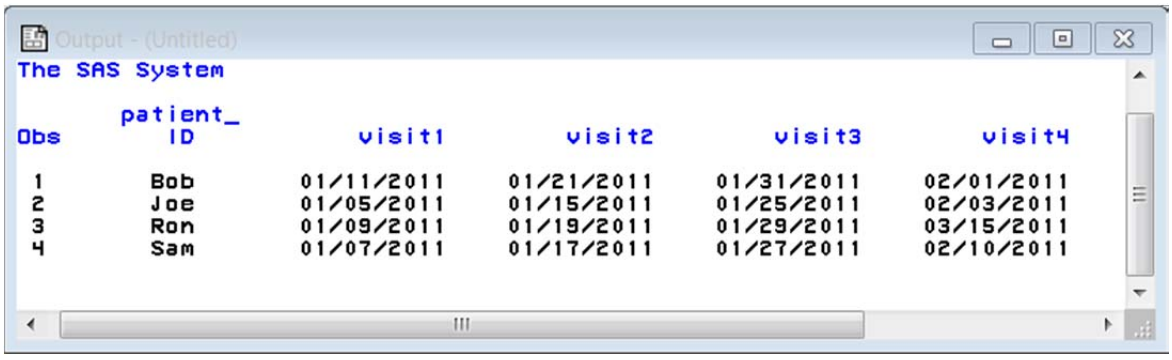
- Sort the expand data set by Patient_ID.
- Clear the array and the array index variable (counter) at the beginning of each BY group.

- Increment the array index variable as each observation is read.
- Assign each date to an element of the array.
- Generate one observation per patient ID.

Program

```
   /* Sort data set by Patient_ID and Date_of_Visit. */
proc sort data=expand;
   by patient_ID date_of_visit;
run;
data collapse;
   set expand;
   by patient_ID;
      /* Define an array for the new visit variables. */
   array visit[4];
      /* Retain the variables that are associated with the array. */
   retain visit;
      /* Clear the visit variables and the counter for each new BY */
      /* group (Patient_ID).                                       */
   if first.patient_ID then call missing(of visit[*], counter);
      /* Increment the counter that is used to reference the element */
      /* of array to assign date.                                    */
   counter + 1;
      /* Assign the date to the proper element of the array. */
   visit[counter] = date_of_visit;
      /* Output one observation per BY group (Patient_ID).   */
   if last.patient_ID then output;
      /* Format and drop variables, as desired. */
   format visit: mmddyy10.;
   drop date_of_visit counter;
run;

proc print;
run;
```

This example generates the following output:



Obs	patient_ID	visit1	visit2	visit3	visit4
1	Bob	01/11/2011	01/21/2011	01/31/2011	02/01/2011
2	Joe	01/05/2011	01/15/2011	01/25/2011	02/03/2011
3	Ron	01/09/2011	01/19/2011	01/29/2011	03/15/2011
4	Sam	01/07/2011	01/17/2011	01/27/2011	02/10/2011

Finding a Minimum or Maximum Value As Well As the Corresponding Variable Name

Applications frequently need to find the minimum or maximum value among related variables and, once found, to identify the name of the corresponding variable. By associating the related variables with an array, you can use the MIN or MAX function to first find the value and then use the VNAME function within a DO loop to identify the name of the variable.

Example: Using the MAX and VNAME Functions to Identify the Salesperson Who Has the Largest Sales

This example uses yearly sales data for multiple salespeople to demonstrate the use of the MAX and VNAME functions. Each observation in the sample data contains one year of sales figures for four salespeople. The task is to identify the name of the salesperson who has the largest sales figure for each year.

Key Tasks in This Example

- Define an array that contains the variable name for each salesperson.
- Use the MAX function to identify the largest sales amount (MAX_AMT) within the array.
- Loop through the array to find the specific MAX_AMT.
- Upon finding MAX_AMT, use the VNAME function to obtain the name of the variable with that amount.

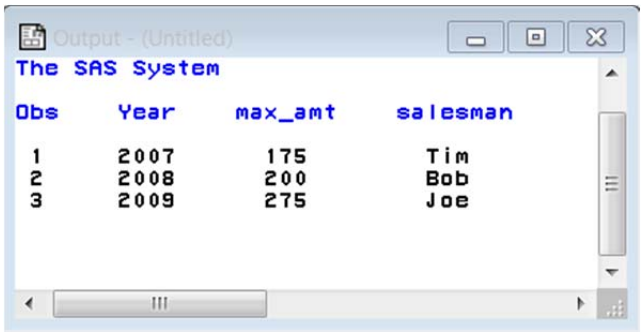
Program

```
data sales;
   input Year Bob Fred Joe Tim;
   datalines;
2007 100 150 125 175
2008 200 130 150 190
2009 250 140 275 200
;
run;

data largest_sales;
   set sales;
      /* Create an array that contains sales figures for each salesman. */
   array sales_amounts[*] Bob Fred Joe Tim;
      /* Use the MAX function to identify the largest sales amount.     */
   max_amt=max(of sales_amounts[*]);
      /* Loop through the array looking for the previously identified   */
      /* amount (value of MAX_AMT).                                     */
      /* Once found, use the VNAME function to retrieve the             */
      /* name of the variable (Salesman). Then output the resulting     */
      /* observation, and leave the DO loop.                            */
```

(code continued)

```
    do i=1 to dim(sales_amounts);
       if sales_amounts[i] = max_amt then do;
          salesman = vname(sales_amounts[i]);
          output;
          leave;
       end;
    end;
       /* Keep the variables as desired. */
    keep year salesman max_amt;
run;

proc print;
run;
```

This example generates the following output:



Conclusion

When you are working with groups of similar data and performing common operations on each element of the group, arrays can save you significant time in coding an application. The use of arrays typically reduces the number of lines of code needed to perform a task, resulting in code that is less error-prone and is more easily maintained by other programmers.

References

SAS Institute Inc. 2011. “Array Processing.” In *SAS® 9.3 Language Reference: Concepts.* Cary, NC: SAS Institute Inc. Available at **support.sas.com/documentation/cdl/en/lrcon/62753/PDF/default/lrcon.pdf**.

SAS Institute Inc. 2011. "Array Reference Statement." In *SAS® 9.3 Statements: Reference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/documentation/cdl/en/lestmtsref/63323/PDF/default/lestmtsref.pdf**.

SAS Institute Inc. 2011 "ARRAY Statement." In *SAS® 9.3 Statements: Reference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/documentation/cdl/en/lestmtsref/63323/PDF/default/lestmtsref.pdf**.

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.
Copyright © 2011 SAS Institute Inc., Cary, NC, USA. All rights reserved.