PhUSE 2017

Paper CT07

Are you Still Afraid of Using Arrays? Let's Explore their Advantages

Vladyslav Khudov, Experis Clinical, Kharkiv, Ukraine

ABSTRACT

At first glance, arrays in SAS® seem to be a complicated and difficult tool, so many programmers try to avoid them in their code. However, knowledge of arrays and the ability to use them effectively can expand your programming techniques. Using arrays allows you to perform the same tasks with different variables, perform table lookups, restructure data, etc. Using arrays can simplify your code and can even help accomplish tasks which can't be easily done with other methods. This paper introduces some examples of how to solve popular tasks, such as finding the date of the last dose prior to event, LOCF, and others. In particular, this paper will provide possible solutions to these types of problems using arrays over alternative strategies. The advantages of each method will be investigated.

INTRODUCTION

An array in SAS is a temporary grouping of variables for the duration of the DATA step. Therefore, once you need to perform the same calculations on more than one variable, array should be the first idea that comes to your mind. Programming using arrays could replace a huge block of the repetitious code with a few simple lines in such situations. Beside of this, arrays allow us to solve other tasks, such as:

- reshaping data;
- creating new variables;
- creating table templates;
- performing table lookups.

In this paper array techniques are implemented to solve some tasks from simple ones to not so obvious.

BASIC CONCEPTS. DO'S AND DON'TS WHILE WORKING WITH ARRAYS

Suppose you have decided to use an array to accomplish a particular task. The first step you need to do is to define the array with an ARRAY statement or, in other words, to assign the name to a group of variables. The syntax of ARRAY statement is shown below.

```
ARRAY array-name {subscript} <$> <length> <_temporary_> <array-elements> <(initial-
value-list)>;
```

The most important points you need to keep in mind while declaring an array are listed below:

- You cannot define an array with both character and numeric elements. All variables that are grouped in an array should be either all character or all numeric.
- An array-name must follow all the rules for valid SAS variable names and the name of an array should be different from any other variable in the same DATA step. In addition, do not use SAS function names as an array-name if you do not want to lose its functionality.
- The ARRAY statement is not an executable statement. During the compilation phase, SAS either associates the array name with existing variables or creates new variables in a program data vector (PDV).
- The subscript can be a number, a range of numbers or an asterisk (*) sign. Functions, variables, etc. are not allowed to be placed as a subscript. This reflects the fact that during compilation, when the ARRAY statement is processed, SAS cannot calculate the value of a function or determine the value of a variable. Moreover, for the same reason, an asterisk cannot be used when defining a temporary or multidimensional array or if you do not list variables.
- You cannot specify the list of variables when _TEMPORARY_ keyword is used. You only can refer to the temporary data element by an array name and a subscript.
- SAS automatically retains the values of a temporary array. They are not reset to missing at the beginning of the next iteration of the DATA step.
- When you specify an initial value list, the values of all elements are retained; a RETAIN statement is not necessary.

After you have defined the array in the DATA step you need to refer to its elements. This can be easily done using the ARRAY reference statement: the array-name, followed by the subscript value in curly braces (parentheses or square brackets are also possible).

```
array-name {subscript}
```

A number, a numeric variable, SAS expression, or an asterisk can be placed as the subscript in the ARRAY reference statement. The ARRAY reference statement can be an argument for a number of SAS functions and CALL routines. The most useful among them are:

1) Functions to retrieve the number of array elements: DIM, LBOUND, and HBOUND. These functions are quite helpful for specifying bounds of a DO loop. As you might know, DIM function returns the number of elements in an array. LBOUND and HBOUND functions return lower and upper bounds respectively.
2) Statistical functions, such as SUM, MEAN, MIN, MAX, etc. Operator OF is often useful when applying these functions over the elements of an array.
3) Call routines, such as: CALL MISSING, CALL SORTN and others that accept a list of variables as an argument.
4) Functions to retrieve index: WHICHN, WHICHC, etc.

Some examples of how to use these functions effectively will be provided further in this paper.

You can refer to an array in almost any place of the DATA step. Exceptions are KEEP, DROP, FORMAT, LABEL, and LENGTH statements where the list of variables must be specified.

LAST OBSERVATION CARRIED FORWARD

Last Observation Carried Forward, or LOCF for short, is a common method when missing data should be imputed with the last non-missing value. Let's take a look at a simple example. Suppose systolic blood pressure was measured and some visits have missing data.

	SUBJECT	VISIT1	VISIT2	VISIT3	VISIT4
1	1	87	.	.	123
2	2	156	.	.	.
3	3	112	.	112	.
4	4	.	70	.	.

Display 1. Raw data set "sys_bp"

You may want to impute missing values using LOCF method and get the following.

	SUBJECT	VISIT1	VISIT2	VISIT3	VISIT4
1	1	87	87	87	123
2	2	156	156	156	156
3	3	112	112	112	112
4	4	.	70	70	70

Display 2. Result data set "sys_bp_locf"

Array technique can be quite useful to accomplish such a task. The simple code to solve this task is below.

```
data sys_bp_locf;
   set sys_bp;
   array visits{*} visit: ;
   do i=1 to dim(visits)-1;
    if visits{i+1} eq . then visits{i+1}=visits{i};
   end;
run;
```

FIND THE LAST DATE IN A GROUP OF DATES PRIOR TO ANOTHER DATE

Suppose you need to know the date of the last dose prior to some event, such as vital signs or laboratory assessment, adverse event and so on. In this paper, an example will be shown of how to find the latest medication administration date before the adverse event occurred. This can be done using one-dimensional array and two-dimensional temporary array. These two methods will be compared to SQL join technique. Each method will be examined to find the more suitable for a particular situation.

It is common to derive the Date of Last Dose variable for the data analysis. As you know, analysis data sets are based on the source data (SDTM). In this example, incoming data sets are, basically, the data in SDTM standard after some manipulations. To focus on implementing the aforementioned techniques these modifications include the following: all date/time variables are numeric values in DATETIME20. format. Only time was imputed. Partial dates are possible, so missing date/time values indicate that day, month and/or year are missing. Subject identifiers are numeric values.

A small excerpt of each data set is listed below to give you an idea about their contents.

	SBJID	AESEQ	AETERM	AESTDTN
1	1	1	ABDOMINAL CRAMPS	09NOV2015:08:20:59
2	1	2	DIARRHEA	09NOV2015:12:01:59
3	1	3	NAUSEA	09NOV2015:22:00:59
4	1	4	THROMBOCYTOPENIA	13NOV2015:08:12:59
5	1	5	RESTART OF CHRONIC DIVERTICULITIS	19NOV2015:06:00:59
6	1	6	FEBRILE NEUTROPENIA	07DEC2015:14:18:59
7	1	7	PYELONEPHRITIS	07DEC2015:19:45:59
8	1	8	FEVER	07DEC2015:23:37:59
9	2	1	HYPERGLYCEMIA	29MAR2016:22:00:59
10	2	2	THROMBOSIS LEFT SUBCLAVIA VEIN	01APR2016:08:20:59
11	2	3	HEMORRHAGIC DIATHESIS	06APR2016:03:30:59
12	2	4	NAUSEA	06APR2016:11:48:59
13	2	5	THROMBOCYTOPENIA	06APR2016:18:52:59
14	2	6	HYPOPHOSPHATEMIA	12APR2016:15:22:59
15	2	7	THROMBOCYTOPENIA	12APR2016:17:48:59
16	3	1	EPIGASTRALGIA	09FEB2016:14:25:59
17	3	2	INTERMITTENT DIARRHEA	15FEB2016:20:35:59
18	3	3	LOSS OF WEIGHT	15FEB2016:23:59:59
19	3	4	EPITAXIS	20FEB2016:23:59:59
20	3	5	ANOREXIA	26FEB2016:23:59:59
21	3	6	CUTANEOUS ECCHYMOSIS	26FEB2016:23:59:59

Display 3. Raw data set “ae”

	SBJID	EXSEQ	EXDOSE	EXSTDTN	EXENDTN
1	1	1	60	09NOV2015:08:00:00	09NOV2015:08:00:00
2	1	2	60	11NOV2015:08:01:00	11NOV2015:08:01:00
3	1	3	45	.	.
4	1	4	30	16NOV2015:08:02:00	16NOV2015:08:02:00
5	1	5	15	18NOV2015:08:10:00	18NOV2015:08:10:00
6	1	6	15	01DEC2015:08:11:00	01DEC2015:08:11:00
7	1	7	30	02DEC2015:08:03:00	02DEC2015:08:03:00
8	2	1	60	29MAR2016:09:25:00	29MAR2016:09:25:00
9	2	2	60	05APR2016:09:26:00	05APR2016:09:26:00
10	2	3	15	07APR2016:09:10:00	07APR2016:09:10:00
11	2	4	15	12APR2016:09:11:00	12APR2016:09:11:00
12	3	1	60	01FEB2016:09:00:00	01FEB2016:09:00:00
13	3	2	60	08MAR2016:09:00:00	08MAR2016:09:00:00
14	3	3	15	10MAR2016:08:00:00	10MAR2016:08:00:00
15	3	4	15	15MAR2016:08:00:00	15MAR2016:08:00:00
16	4	1	60	12OCT2015:09:14:00	12OCT2015:09:14:00
17	4	2	60	23OCT2015:09:15:00	23OCT2015:09:15:00
18	4	3	15	24OCT2015:08:10:00	24OCT2015:08:10:00
19	4	4	15	26OCT2015:09:11:00	26OCT2015:09:11:00
20	5	1	60	12FEB2016:12:15:00	12FEB2016:12:15:00
21	5	2	60	26FEB2016:09:17:00	26FEB2016:09:17:00

Display 4. Raw data set “ex”

One and probably the most common solution is to merge data sets so that we can compare start date of an adverse event with each treatment dosing date. The closest date before the adverse event will be what we are looking for. Since many-to-many merge is required, the PROC SQL is being used. You can find the code in the Appendix, since PROC SQL is not the topic of the current paper. The result is shown in Display 5.

	SBJID	AESEQ	AETERM	AESTDTN	AEMDTN
1	1	1	ABDOMINAL CRAMPS	09NOV2015:08:20:59	09NOV2015:08:00:00
2	1	2	DIARRHEA	09NOV2015:12:01:59	09NOV2015:08:00:00
3	1	3	NAUSEA	09NOV2015:22:00:59	09NOV2015:08:00:00
4	1	4	THROMBOCYTOPENIA	13NOV2015:08:12:59	11NOV2015:08:01:00
5	1	5	RESTART OF CHRONIC DIVERTICULITIS	19NOV2015:06:00:59	18NOV2015:08:10:00
6	1	6	FEBRILE NEUTROPENIA	07DEC2015:14:18:59	02DEC2015:08:03:00
7	1	7	PYELONEPHRITIS	07DEC2015:19:45:59	02DEC2015:08:03:00
8	1	8	FEVER	07DEC2015:23:37:59	02DEC2015:08:03:00
9	2	1	HYPERGLYCEMIA	29MAR2016:22:00:59	29MAR2016:09:25:00
10	2	2	THROMBOSIS LEFT SUBCLAVIA VEIN	01APR2016:08:20:59	29MAR2016:09:25:00
11	2	3	HEMORRHAGIC DIATHESIS	06APR2016:03:30:59	05APR2016:09:26:00
12	2	4	NAUSEA	06APR2016:11:48:59	05APR2016:09:26:00
13	2	5	THROMBOCYTOPENIA	06APR2016:18:52:59	05APR2016:09:26:00
14	2	6	HYPOPHOSPHATEMIA	12APR2016:15:22:59	12APR2016:09:11:00
15	2	7	THROMBOCYTOPENIA	12APR2016:17:48:59	12APR2016:09:11:00
16	3	1	EPIGASTRALGIA	09FEB2016:14:25:59	01FEB2016:09:00:00
17	3	2	INTERMITTENT DIARRHEA	15FEB2016:20:35:59	01FEB2016:09:00:00
18	3	3	LOSS OF WEIGHT	15FEB2016:23:59:59	01FEB2016:09:00:00
19	3	4	EPITAXIS	20FEB2016:23:59:59	01FEB2016:09:00:00
20	3	5	ANOREXIA	26FEB2016:23:59:59	01FEB2016:09:00:00
21	3	6	CUTANEOUS ECCHYMOSIS	26FEB2016:23:59:59	01FEB2016:09:00:00

Display 5. Result data set “last_dose”

Actually, the task where you need to determine the value of one variable based on the value of another one is called a table lookup. DATA step arrays is one of the methods used to solve such tasks.

ONE-DIMENSIONAL ARRAY METHOD

Let’s consider the idea of this approach in order to understand the code provided below. Rather than performing many-to-many merge, you can reshape “ex” data set to get one observation per subject structure. After merging this ‘wide’ data set with the adverse event data, start dates of the adverse event will be compared with dosing dates. Assuming both dates are not null, the closest dosing date will be what we are looking for.

Below is SAS code to find the last dose date before an adverse event occurred.

```
proc sort data=ex;
   by sbjid;
run;
proc sort data=ae;
   by sbjid;
run;
```

```
proc transpose data=ex out=tex prefix=dt;
   var exendtn;
   by sbjid;
run;
data _null_;
set tex;
   array doses{*} dt: ;
   call symputx('n_doses', dim(doses));
run;
data last_dose;
   merge ae (in=inAE)
         tex;
   by sbjid;
   if inAE;
   array doses{&n_doses.} dt: ;
   array diffs{&n_doses.};
   do i=1 to dim(doses);
     if aestdtn ne . and doses{i} ne . then diffs{i}=aestdtn-doses{i};
     if diffs{i} >=0 then min_value=min(min_value, diffs{i});
   end;
   if min_value ne . then aemdtn=doses{whichn(min_value, of diffs{*})};
   format aemdtn datetime20.;
   keep sbjid aeseq aeterm aestdtn aemdtn;
run;
```

Since this method requires transposing and merging, at first we need to use two PROC SORTs. The array that contains differences should have the same dimension as the array with dosing dates. That is why macro variable that contains the dimension was created. To find the index of the minimum value in the array of values SAS function WHICHN was used.

You could ask why the array for differences cannot be temporary as none of difference values is stored in output data set. The answer is obvious. As was mentioned above, values of the temporary array are always automatically retained. Therefore, if _TEMPORARY_ keyword is specified, difference in dates will be retained from one subject to another, which will not be the expected behavior.

The main advantage of this solution is that the data set is not expanded by creating extra observations as was done when method with SQL procedure was used. Furthermore, many-to-many join was replaced by many-to-one merge which is less prone to errors. On the other hand, PROC TRANSPOSE and PROC SORT consume some resources and increase processing time.

TWO-DIMENSIONAL ARRAY METHOD

The same result can be accomplished without sorting, merging and transposing. Let's see how to do this using two-dimensional temporary array.

```
data _null_;
   set ex end=last;
   retain min_sbj max_sbj dim_ex;

   dim_ex=max(dim_ex, exseq);
   min_sbj=min(min_sbj, sbjid);
   max_sbj=max(max_sbj, sbjid);

   if last then do;
     call symputx('n_doses', dim_ex);
     call symputx('l_sbj', min_sbj);
     call symputx('h_sbj', max_sbj);
   end;
 run;

data last_dose;
   array sbj_dt{&l_sbj. : &h_sbj., &n_doses.} _temporary_;

   if _n_=1 then do until(exDone);
    set ex end=exDone;
      sbj_dt{sbjid,exseq}=exendtn;
   end;

   set ae;
```

```
    array diffs{&n_doses.};

    do j=1 to dim(diffs);
      if aestdtn ne . and sbj_dt{sbjid, j} ne . then
        diffs{j}=sum(aestdtn, -sbj_dt{sbjid, j});
      if diffs{j}>=0 then min_value=min(min_value, diffs{j});
    end;

    if min_value ne . then aemdtn=sbj_dt{sbjid, whichn(min_value, of diffs{*})};
    format aemdtn datetime20.;
    keep sbjid aeseq aeterm aestdtn aemdtn;
  run;
```

The first DATA step creates three macro variables to store number of subjects and number of doses. The array with dosing dates is populated during the first iteration of the DATA step. This array has two dimensions, where first dimension is the subject id and second dimension is the dose number. At each iteration of the DATA step, dosing dates are compared to date of the adverse event. The results are the same as shown in Display 5.

The main advantage of this elegant technique is speed of running. At the same time, arrays require numeric index values, so we would not have been able to use this approach if any of index variables had been character. So arrays have their disadvantages as well.

COMPARISON OF METHODS

Advanced programmer should consider various programming techniques based on their ability to conserve specific resources, such as CPU time and memory usage. To compare performance of each method these characteristics will be assessed. CPU time is the amount of time that the central processing unit, or CPU, uses to perform tasks such as calculations, reading and writing data and includes user CPU time and system CPU time. Memory represents the amount of memory allocated to that job/step.

Five tests had been run on each technique and the average of five tests is used to measure the performance. Tests were run on medium and large data sets. You can see the results in tables below.

Number of observations: AE: 68497 EX: 142857.

	One-dimensional array method	Two-dimensional array method	Program with PROC SQL
CPU time (s)	0.77	0.27	2.4
Memory (k)	33630.45	4196.99	344231.35

Number of observations: AE: 532467 EX: 255307.

	One-dimensional array method	Two-dimensional array method	Program with PROC SQL
CPU time (s)	2.48	1.05	12.91
Memory (k)	128450.69	8813	1052423.79

CONCLUSION

SAS allows you to solve the same task using various techniques. Each method has its own pros and cons. You can choose straightforward solutions, but you can spend extra resources and time, waiting while the program executes. However, using advanced techniques, such as an array processing, will allow you to save time and typing.

REFERENCES

Zaizai Lu, David Shen. Advanced Array Applications in Clinical Data Manipulation.

Shaoan Yu and Joyce Gui. An Approach for Deriving a Timing Variable in SDTM Standards. SESUG proceedings

Art Carpenter. Carpenter's Guide to Innovative SAS® Techniques.

SAS Institute Inc. 2011. "Array Processing." In SAS® 9.3 Language Reference: Concepts. Cary, NC: SAS Institute Inc.

Ben Cochran. Using SAS® Arrays to Manipulate Data.

Arthur L. Carpenter. Using Arrays to Quickly Perform Fuzzy Merge Look-ups: Case Studies in Efficiency.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Vladyslav Khudov
Company: Experis Clinical
Address: 43/2 Gagarin Avenue
City / Postcode: Ukraine, Kharkiv, 61001
Work Phone: +38 044 500 7020 ext. 2428
Fax: +38 057 728 30 35

Email: vladyslav.khudov@intego-group.com

APPENDIX

```
proc sql;
    create table step1 as select
      a.sbjid, a.aestdtn, a.aeseq, a.aeterm,
      b.exendtn, (a.aestdtn-b.exendtn) as diff,
      (case when (a.aestdtn-b.exendtn) >= 0 then 'Y' else 'N' end) as flag
    from ae as a
    left join ex as b
    on a.sbjid = b.sbjid
    order by sbjid, aeseq, flag desc, diff;
quit;

data last_dose_sql;
   set step1;
   by sbjid aeseq descending flag diff ;

   if flag eq 'Y' then aemdtn=exendtn;
   if first.aeseq;
   format aemdtn datetime20.;
   keep sbjid aeseq aeterm aestdtn aemdtn;
run;
```