Paper 209-2010

SAS® Arrays, Both Implicit and Explicit

Carter Sevick, MS
DoD Center for Deployment Health Research, San Diego, CA

ABSTRACT

In SAS, arrays can be defined with implicit or explicit subscripting. Current documentation only covers the explicit version and the implicit version is only supported for backward compatibility. This can cause problems for those tasked with the maintenance of legacy code. One can also find examples of recent conference proceedings in which implicit array coding is used.

This paper will cover the syntax and basic usage of the implicit array. Additionally, several common array tasks will be presented in both forms. Familiarity with basic DATA step programming and array processing will be assumed.

INTRODUCTION

The SAS Institute is strongly committed to backward compatibility in the SAS language as it evolves. As new constructs are brought in to the language, old ones are sometimes dropped from the documentation even though they continue to be supported. Not only does “old” code continue to run, but individuals that had the benefit to have learned SAS before a change in the documentation can continue to compose programs without changing to the new standard.

This situation holds true for arrays in SAS. If one inspects the SAS documentation, or the various introductory texts, the style of array that is shown is explicitly subscripted. Looking back to the version 6 documentation, one will find a second type of array the subscript of which is defined implicitly. The documentation recommends that only the explicit array be used in the future, but the implicit array persists even in recent user group presentations. Those of us who came to SAS more recently are out of luck, unless we happen to have access to the old documentation, and do not have a resource to turn to for full understanding of this notation. Hopefully this paper will be of help to those unfamiliar with the implicit array. After analyzing this different take on array processing, some may even find a place for this construct in their toolbox.

IMPLICIT ARRAY SYNTAX

To declare an implicit array the syntax is as follows:

ARRAY array-name <(index-variable)> <$> <length> array-elements;

array-name:
: The name of the array. The chosen name must be a valid SAS name, and it cannot be the same as another variable in the current DATA step.

index-variable:
: If not included then an automatic variable, _I_, will be generated which is used to control which element in the array is accessed. Like all automatic variables, it is not written to the output dataset. Alternatively the programmer can specify a variable name of their choosing. If a variable is specified then it will be written to the output dataset unless DROP or KEEP are used.

$:
: Use to define the array elements as character. This is not necessary if the variables were defined as such prior to the ARRAY statement.

length:
: Use to define a length for the array elements, if not done prior to the ARRAY statement.

array-elements:
: Elements to include in the array are named here. Valid elements include variables and other implicit arrays. Numeric and character elements cannot be mixed. If a variable is named, and it does not exist, then it will be created. Variable lists (dx1 – dx7, _CHARACTER_, etc.) are also valid in this context.

REFERENCING AN IMPLICIT ARRAY

To reference an implicit array simply write the array-name without an index specification. Which element is accessed is controlled by the current value of the index-variable. In the following example “impl” is an implicit array and “expl” is explicit. Note that automatic index variable, _I_, needs to be set to the number of the desired element.

```
DATA refTest;

  INPUT (x1 - x5) ($1.);

  ARRAY impl x1 - x5;
  ARRAY expl(5) x1 - x5;

  _I_ = 3;

  PUT 'implicit result: ' impl;
  PUT 'explicit result: ' expl(3);

DATALINES;
abcde
;
```

From the log:

```
.
.
214
215  DATALINES;

implicit result: c
explicit result: c
NOTE: The data set WORK.REFTEST has 1 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

INCREMENTING THROUGH AN IMPLICIT ARRAY

In the above example we see the letters a – e loaded in to variables x1 – x5. Let's say we wanted to create on variable to include all letters: 'abcde'. Using an explicit array we could do the following:

```
DATA explIncrement;

  INPUT (x1 - x5) ($1.);
  LENGTH allLetters $ 5;
  ARRAY expl(5) x1 - x5;

  DO i = 1 TO 5;
    allLetters = STRIP(allLetters)||expl(i);
  END;

  PUT _ALL_;

  DROP i;
DATALINES;
abcde
;
```

From the Log:

```
x1=a x2=b x3=c x4=d x5=e allLetters=abcde i=6 _ERROR_=0 _N_=1
NOTE: The data set WORK.EXPLINCREMENT has 1 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds
```

The exact same result can be arrived at with an implicit array by incrementing its index variable.

```
DATA implIncrement;

  INPUT (x1 - x5) ($1.);
  LENGTH allLetters $ 5;
  ARRAY impl x1 - x5;

  DO _I_ = 1 TO 5;
    allLetters = STRIP(allLetters)||impl;
  END;

  PUT _ALL_;
DATALINES;
abcde
;
```

However, there is a special form of DO loop especially for the implicit array:

```
DO OVER array-name;

   programming statements;

END;
```

The following is equivalent to the previous example:

```
DATA withDoOver;

  INPUT (x1 - x5) ($1.);
  LENGTH allLetters $ 5;
  ARRAY impl x1 - x5;

  DO OVER impl;
    allLetters = STRIP(allLetters)||impl;
  END;

  PUT _ALL_;
DATALINES;
abcde
;
```

The DO OVER statement automatically iterates through an implicit array, using the index variable to do so. Thus, one should take care not to write code within the DO OVER loop that would act on the value of the index. More than one implicit array can be used in a DO OVER block, as long as they have the same assigned index variable. DO OVER cannot be used with explicit arrays.

FURTHER EXAMPLES AND DATA

In order to demonstrate the usage of implicit array coding, several tasks (commonly performed using arrays) will be coded first using the explicit notation that is typical in SAS programming. The task will then be recoded using the implicit form.

The data for the example will be borrowed from the National Center for Health Statistics (NCHS). The center provides a wide variety of public use data files that are available for free and can provide wonderful grist for practicing programming methods. The data set chosen for these examples is the 2006 National Hospital Discharge Survey. The dataset is a national sample of hospital discharges conducted yearly. The sample design is beyond the scope of this paper, but the procedures are well documented. See the reference section for a citation and information on how to find the data and documentation. Be sure to read, and respect, the data user agreement.

The dataset is organized as one discharge per row and the set is de-identified such that if a single individual had two discharges there is no way to link them together. The files are stored as fixed width text files and when read in to SAS a row number (rowNum) counter was added to serve as a record identifier.

TASK 1: FIND RECORDS OF ASTHMA HOSPITALIZATIONS

There are 7 fields (dx1 – dx7) for diagnoses relating to the hospital stay. Each field is coded using ICD_9_CM (International Classification of Diseases) standards, see the reference section if you would like additional information on ICD_9_CM. Briefly, the coding system provides a standardized way to indicate clinical diagnoses. Each diagnosis is coded as a string 3 – 5 characters in length. For example, '493' is the general classification for asthma. Fourth and fifth digits are added for greater detail of the condition.

The following code uses an explicit array to check each diagnostic field for codes beginning with '493'. If one is found then a flag variable is set to 1 and the loop is exited. PROC MEANS is then used to count the number of records obtained and the number of discharges represented.

```
DATA nhsdAsthma;
  SET nhds.nhds2006;
  ARRAY expl(7) dx1 - dx7;

  asthma = 0;

  DO i = 1 TO 7;
    IF SUBSTR(expl(i),1,3) = '493' THEN DO;

      asthma = 1;
      LEAVE;

    END;
  END;

  DROP i;
RUN;

PROC MEANS DATA = nhsdAsthma N SUM;
  WHERE asthma = 1;
  VAR weight;
RUN;

                    The MEANS Procedure

         Analysis Variable : weight Analysis weight

                      N            Sum
                  ______________________

                  20262      1905364.00
                  ______________________
```

The next example illustrates the same task, but with an implicit array:

```
DATA nhsdAsthma;
  SET nhds.nhds2006;
  ARRAY impl dx1 - dx7;

  asthma = 0;

  DO OVER impl;
    IF SUBSTR(impl,1,3) = '493' THEN DO;

      asthma = 1;
      LEAVE;

    END;
  END;

RUN;

PROC MEANS DATA = nhsdAsthma N SUM;
  WHERE asthma = 1;
  VAR weight;
RUN;

                     The MEANS Procedure

          Analysis Variable : weight Analysis weight

                       N            Sum
                    ______________________

                   20262      1905364.00
                    ______________________
```

TASK 2: TURN THE BLOCK

The need to transpose a data set is common. PROC TRANSPOSE was developed just for that purpose but many SAS programmers prefer to do it with a DATA step using an array, a DO loop and an OUTPUT statement. The treatment of the diagnostic fields depends largely on the requirements of the analysis. In Task 1 we were looking for a diagnosis in any field (no hierarchy), and this is often the case with this type of data. Continuing with this idea it can be most helpful to restructure medical records in to one row per diagnosis, or “wide to skinny”. The new dataset will have three fields:

1) rowNum – the record ID.
2) Diag – the ICD-9-CM diagnostic code.
3) dxNum – the number of the diagnostic field the row was derived from.

The first attempt will be with an explicit array:

```
DATA turnBlock;
  SET nhds.nhds2006 (KEEP = rowNum dx1 - dx7);
  LENGTH diag $ 5;
  ARRAY expl(7) dx1 - dx7;

  DO dxNum = 1 TO 7;
    IF expl(dxNum) NE ' ' THEN DO;

      diag = expl(dxNum);
      OUTPUT;

    END;
  END;

  KEEP rowNum diag dxNum;
RUN;
**************************************;
From the Log:

NOTE: There were 376328 observations read from the data set
NHDS.NHDS2006.
NOTE: The data set WORK.TURNBLOCK has 1864898 observations and 3
variables.
NOTE: DATA statement used (Total process time):
      real time           2.93 seconds
      cpu time            0.46 seconds
```

Let's look at what happened to the first five records:

```
PROC PRINT DATA = nhds.nhds2006 (OBS = 5);
  VAR rowNum dx1 - dx7;
RUN;

              row
       Obs    Num      dx1      dx2      dx3      dx4      dx5      dx6      dx7

         1     1      07070    7813-    30551    4019-    30521    6929-    311--
         2     2      88110    9583-
         3     3      8505-    9160-    9130-    9120-
         4     4      1748-
         5     5      29644    7295-
```

And after the DATA step:

```
            row                 dx
   Obs      Num     diag        Num

     1       1      07070        1
     2       1      7813-        2
     3       1      30551        3
     4       1      4019-        4
     5       1      30521        5
     6       1      6929-        6
     7       1      311--        7
     8       2      88110        1
     9       2      9583-        2
    10       3      8505-        1
    11       3      9160-        2
    12       3      9130-        3
    13       3      9120-        4
    14       4      1748-        1
    15       5      29644        1
    16       5      7295-        2
```

This would be a step in an attempt to normalize the medical record, and the benefit of this type of structure is clear when one considers the comparative ease in searching for cases:

```
PROC SQL;

SELECT COUNT(*) AS rowCount,
       SUM(weight) AS dischargeCount
FROM nhds.nhds2006
WHERE rowNum IN (SELECT rowNum
                 FROM turnBlock
                 WHERE SUBSTR(diag,1,3) = '493');

QUIT;

***************************************************;
List Output:

                                  discharge
                    rowCount        Count
                  ________________________

                   20262         1905364
```

The same result can be achieved using an implicit array and DO OVER statement as we are simply checking each element in an array, in an orderly fashion. After implementing the new code, PROC COMPARE will be used to show that the result is exactly the same.

```
DATA turnBlock2;
  SET nhds.nhds2006 (KEEP = rowNum dx1 - dx7);
  LENGTH diag $ 5;
  ARRAY impl(dxNum) dx1 - dx7;

  DO OVER impl;
    IF impl NE ' ' THEN DO;

      diag = impl;
       OUTPUT;

    END;
  END;

  KEEP rowNum diag dxNum;
RUN;

PROC COMPARE DATA = turnBlock COMPARE = turnBlock2;
RUN;
****************************************************;
Partial PROC COMPARE output:

 Number of Observations in Common: 1864898.
 Total Number of Observations Read from WORK.TURNBLOCK: 1864898.
 Total Number of Observations Read from WORK.TURNBLOCK2: 1864898.

 Number of Observations with Some Compared Variables Unequal: 0.
 Number of Observations with All Compared Variables Equal: 1864898.

 NOTE: No unequal values were found. All values compared are exactly equal.
```

DISCUSSION AND CONCLUSION

The implicit array is a construct that continues to find use in the SAS programming community. While it is not difficult to understand its overall use, some aspects (such as the automatic index) are not clear just by inspection of code. If you have been faced with the implicit array in someone else's code and been at a loss, then hopefully this paper has helped to provide clarity.

ACKNOWLEDGMENTS

The author would like to thank his wife Myrna for her support and patience throughout this process. Also, Tyler Smith PhD., and the members of the Data Team at the DoD Center for Deployment Health Research who have been an invaluable source of inspiration, support, and assistance. The members include Besa Smith, Isabel Jacobson, Cynthia LeardMann, Charlene Wong, Anna Bukowinski, LtCol Christopher Phillips, Martin White, Melissa Bagnell, Sydney Lee, Nisara Granado, Jamie Horton, Teresa Powell, Amanda Pietrucha, Amber Seelig, Kelly Jones, and Donald Sandweiss.

REFERENCES

SAS Institute 2009, "SAS 9.2 Language Reference: Dictionary",
http://support.sas.com/documentation/cdl/en/lrdict/62618/HTML/default/titlepage.htm

SAS Institute 2009, "SAS 9.2 Language Reference: Concepts",
http://support.sas.com/documentation/cdl/en/lrcon/61722/HTML/default/titlepage.htm

SAS Institute 1990, *SAS Language: Reference, Version 6.* Cary, North Carolina: SAS Institute.
National Center for Health Statistics 2008, "National Hospital Discharge Survey 2006 Public Use Data File Documentation",
ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Dataset_Documentation/NHDS/NHDS_2006_Documentation.pdf

And the data file can be found at:
ftp://ftp.cdc.gov/pub/Health_Statistics/NCHS/Datasets/NHDS/nhds06/

Information on ICD 9 CM codes can be found at:
http://www.cdc.gov/nchs/icd/icd9cm.htm

CONTACT INFORMATION

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

Carter J. Sevick, MS
Biostatistician
DoD Center for Deployment Health Research
Naval Health Research Center
Phone: 619-767-4762
Fax: 619-553-7601
carter.sevick@med.navy.mil