Sharpening Your Skills in Reshaping data: 
PROC TRANSPOSE vs. Array Processing

Arthur X. Li, City of Hope National Medical Center, Duarte, CA

ABSTRACT
A common data managing task for SAS® programmers is transposing data. One of the reasons for performing data transformation is that different statistical procedures require different data shapes. In SAS, two commonly used methods for transposing data are using either the TRANPOSE procedure or array processing in the DATA step. Using PROC TRANSPOSE mainly requires grasping the syntax and recognizing how to apply different statements and options in PROC TRANSPOSE to different types of data transposition. On the other hand, utilizing array processing in the DATA step requires programmers to understand how the DATA step processes data during the DATA step execution. In this talk, these two methods will be reviewed and compared through various examples.

INTRODUCTION
Reshaping data is a process of rearranging the contents of the data without modifying the data contents. One of the reasons for reshaping data is that different statistical procedures require a specific data shape. Reshaping data is not exactly equivalent to transposing data. When transposing data, data presented in rows from the input data is transposed into columns in the resulting data. For example, Dat1 (See Figure 1) contains the three English test scores for John and Mary. The scores are stored in three columns, E1 – E3, and two rows (for two observations) in Dat1. All the scores are presented in the form of a 2 X 3 matrix. To transpose the scores in Dat1, the scores in the rows need to be rotated to columns or scores in columns need to be rotated to rows. The dataset Dat1_Transpose1 is the transposed form of data set Dat1. Notice that all the scores are presented in the form of a 3 X 2 matrix in the transposed data.

You can also transpose Dat1 for each person. The values of E1 – E3 for each person/observation can also be considered as a group of scores, with each group being identified by the value of the NAME variable. The variable that is used to distinguish the groupings is called the BY-variable. The resulting transposed data set Dat1_Transpose2 is the transposed form of Dat1 by each level of the NAME variable. The variable TEST is used to distinguish the different scores.

Dat1:

<table>
<thead>
<tr>
<th>Name</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
</tr>
</tbody>
</table>

Dat1_Transpose1:

<table>
<thead>
<tr>
<th>Test</th>
<th>John</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E1</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>E2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>E3</td>
<td>92</td>
</tr>
</tbody>
</table>

Dat1_Transpose2:

<table>
<thead>
<tr>
<th>Name</th>
<th>Test</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>E1</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>E2</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>E3</td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>E1</td>
</tr>
<tr>
<td>5</td>
<td>Mary</td>
<td>E3</td>
</tr>
</tbody>
</table>

Figure 1. SAS data sets, Dat1, Dat1_Transpose1, and Dat1_Transpose2.

Sometimes reshaping data cannot be easily done by simply rotating the rows and columns of the data. For example, Dat4 (See Figure 2) contains three test scores for English (E1-E3) and math (M1-M3). In the resulting reshaped data, these scores are arranged vertically by their test number and occupied by four columns. In this situation, reshaping data from Dat4 to Dat4_Reshape requires transposing Dat4 more than once in order to achieve the desired result.
THE TRANSPOSE PROCEDURE
To transpose data, you need to follow the syntax below. The six statements in the TRANSPOSE procedure, which includes the PROC TRANPOSE, BY, COPY, ID, IDLABEL, and VAR statements, along with the eight options in the PROC TRANSPOSE statement, are used to apply different types of data transpositions and to give the resulting data set a different appearance. Most of the statements and options will be introduced via various examples in this paper.

The syntax for the PROC TRANSPOSE statement is:

```
PROC TRANSPOSE <DATA=input-data-set>
  <DELIMITER=delimiter>
  <LABEL=label>
  <LET>
  <NAME=name>
  <OUT=output-data-set>
  <PREFIX=prefix>
  <SUFFIX=suffix>
BY <DESCENDING> variable-1
  .. <DESCENDING> variable-n>
COPY variable(s);
ID variable;
IDLABEL variable;
VAR variable(s);
```

ARRAY PROCESSING
In this paper, only a brief introduction to array processing and its basic syntax is presented here. The readers need to refer to SAS documentation and other papers for a more in-depth discussion on array processing.

A SAS array is a temporary grouping of SAS variables under a single name. Arrays only exist for the duration of the DATA step. The ARRAY statement is used to group previously-defined variables or can be used to create a group of new variables. The ARRAY statement has the following form:

```
ARRAY ARRAYNAME[DIMENSION] <$> <ELEMENTS>;
```

ARRAYNAME in the ARRAY statement must be a SAS name that is not the name of a SAS variable in the same DATA step. DIMENSION is the number of elements in the array. The optional $ sign indicates that the elements in the array are character elements; The $ sign is not necessary if the elements have been previously defined as character elements. ELEMENTS are the variables to be included in the array, which must either be all numeric or characters.

After an array is defined, you need to reference any element of an array by using the following syntax:

```
ARRAYNAME[INDEX]
```

INDEX can be specified as an integer, a numeric variable, or a SAS expression and must be within the lower and upper bounds of the DIMENSION of the array.

---

**Dat4:**

<table>
<thead>
<tr>
<th>Name</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>89</td>
<td>90</td>
<td>92</td>
<td>78</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>81</td>
<td>76</td>
<td>91</td>
<td>89</td>
</tr>
</tbody>
</table>

**Dat4_Reshape:**

<table>
<thead>
<tr>
<th>Test_num</th>
<th>John_e</th>
<th>John_m</th>
<th>Mary_e</th>
<th>Mary_m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>89</td>
<td>78</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>90</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>92</td>
<td>90</td>
<td>81</td>
</tr>
</tbody>
</table>

Figure 2. SAS data sets, **Dat4** and **Dat4_Reshape**.
Sometimes when you reshape data, you need to use multidimensional arrays. The syntax for creating multidimensional arrays is similar to the one for creating one-dimensional arrays. The only difference is using multiple numbers instead of one number for the array DIMENSION.

```
ARRAY ARRAYNAME[R, C, ...] <$> <ELEMENTS>;
```

In the ARRAY statement, R refers to the number of rows and C to the number of columns. If there are three dimensions, the next number will refer to the number of pages. For example, the following array groups E1-E3 and M1-M3 variables into a two-dimensional array.

```
array score[2,3] E1-E3 M1-M3;
```

<table>
<thead>
<tr>
<th>Rows</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To reference an element in a two-dimensional array, you need to use both row and column indices. For example, to reference E3, you need to write SCORE[1,3]. More explanations on array processing will be discussed via the examples in this paper.

**TRANSPOSING AN ENTIRE DATA SET**  
**TRANSPOSING AN ENTIRE DATA SET BY USING PROC TRANSPOSE**

Program 1a starts with creating the data set *Dat1*, which has two observations and four variables. E1–E3 contains three test scores for English1 – English3.

In the PROC TRANSPOSE statement, the OUT= option is used to specify the name of the transposed data set. Without using the OUT= option, PROC TRANSPOSE will create a data set that uses the DATA<naming convention>. By default, the transposed variable names will be stored in the _NAME_ variable in the transposed data. However, the _NAME_ variable is dropped by using the DROP= data set option.

The VAR statement is used to specify the variables to be transposed. Not using the VAR statement in this program will yield the same result because PROC TRANSPOSE will transpose all the numeric variables by default.

The ID statement is used to specify the variable from the input data set that contains the values to rename the transposed variables. Without using the ID statement, the transposed variables will have default names COL1 and COL2. You can also use the PREFIX= option in the PROC TRANSPOSE statement to place a prefix in the transposed variable names.

**Program 1a:**

```plaintext
data dat1;
  input name $ e1-e3;
datalines;
John 89 90 92
Mary 92 . 81
;
proc transpose data=dat1
  out=dat1_t1 (drop=_name_);
  var e1-e3;
  id name;
run;
```

```
title 'DAT1 in the Original Form';
proc print data=dat1;
run;
```

```
title 'Transposing DAT1 Using PROC TRANSPOSE';
proc print data=dat1_t1;
run;
```
<Sharpening Your Skills in Reshaping data: PROC TRANSPOSE vs. Array Processing>, continued

Output from Program 1a:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>81</td>
</tr>
</tbody>
</table>

Transposing DAT1 Using PROC TRANSPOSE

<table>
<thead>
<tr>
<th>Obs</th>
<th>John</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>81</td>
</tr>
</tbody>
</table>

TRANSPOSING AN ENTIRE DATA SET BY USING ARRAY PROCESSING

To transpose Dat1, three arrays are needed in the DATA step. The array SCORE is a one-dimensional array for grouping the variables E1-E3 from the input data Dat1. The array ALL is a temporary 2-by-3 array that creates 6 temporary elements A1-A6 in the DATA step; these temporary elements will hold all the values from E1-E3 for the 2 observations. Since this array is temporary, the elements from the array will not be outputted in the output data set. The one-dimensional NEW_SCORE array creates two variables named John and Mary.

To transpose an entire data set via array processing, you need to store all the data values in the two-dimensional array ALL. Storing all the data values is processed via two iterations of the DATA step. Within each iteration of the DATA step, an iterative DO loop is used to copy the values from E1-E3 to the temporary array ALL. After all the data values have been copied into array ALL, that is also when the DATA step reads the last observation of the input data, a nested do loop is used to copy the elements from array ALL to array NEW_SCORE and is output to the output data set. As you can see, using array processing for transposing an entire data set seems to be much more cumbersome compared to using PROC TRANSPOSE.

A more robust program to transpose Dat1 via the DATA step will consist of additional steps to determine the number of observations from the input data automatically and determine the names of the variables in the transposed data. There are many ways to accomplish this. For example, you can use PROC SQL to create a macro variable that contains the number of observations from the input data and another macro variable to contain the values of the NAME variable from the input to name the variables in the transposed data. Since this paper focuses on array processing, I won’t address these issues any further in this paper.

Program 1b:
```sas
data dat1_t2(keep= John Mary);
  set dat1 end=last;
  array score[3] e1-e3;
  i + 1;
  do j = 1 to dim(score);
    all[i,j] = score[j];
  end;
  if last then do;
    do j = 1 to dim(score);
      do i = 1 to 2;
        new_score[i] = all[i,j];
      end;
      output;
    end;
  end;
run;
```
title 'Transposing DAT1 Using Array Processing';
proc print data=dat1_t2;
run;

Output from Program 1a:

Transposing DAT1 Using Array Processing

<table>
<thead>
<tr>
<th>Obs</th>
<th>John</th>
<th>Mary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>81</td>
</tr>
</tbody>
</table>

TRANSPOSING BY-GROUPS (ONE OBSERVATION PER SUBJECT)
TRANSPOSING BY-GROUPS (ONE OBSERVATION PER SUBJECT) BY USING PROC TRANSPOSE

Program 2a transposes Dat1 by using NAME as the BY-variable. To use the BY statement in PROC TRANSPOSE, the data set must be previously sorted by using the same BY-variable. The BY-variable is not transposed. The number of observations in the transposed data set (6) equals the number of BY-groups (2) times the number of variables that are transposed (3). In this program, the WHERE=data set option is used for not outputting the observations with a missing score. Therefore, the number of observations in the transposed data set is 5. The number of transposed variables equals to the number of the observations within each BY-group in the input data set. Thus, in this example, the number of transposed variables is one. The NAME= option in the PROC TRANSPOSE statement is used to specify the name of the variable in the transposed data set that contains the name of the variable that is being transposed.

Program 2a:
proc sort data=dat1 out=dat1_sort;
  by name;
run;

proc transpose data=dat1_sort
  out=dat1_bygrp1 (rename=(col1=score)
  where=(score ne .))
  name=test;
  by name;
  var e1-e3;
run;

title 'Transposing By-groups by Using PROC TRANSPOSE';
proc print data=dat1_bygrp1;
run;

Output from Program 2a:

Transposing By-groups by Using PROC TRANSPOSE

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>e1</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>e2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>e3</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>e1</td>
<td>92</td>
</tr>
<tr>
<td>5</td>
<td>Mary</td>
<td>e3</td>
<td>81</td>
</tr>
</tbody>
</table>
TRANSPOSING BY-GROUPS (ONE OBSERVATION PER SUBJECT) BY USING ARRAY PROCESSING

To transpose Dat1 by the values of the NAME variable, you only need one array E to group the E1-E3 variables. Since there is only one observation per person, you only need one DATA step iteration for each subject. Within each iteration of the DATA step, an iterative DO loop iterates three times to generate the TEST variable via the CATS function, to assign values from the array E to the SCORE variable, and to output the non-missing SCORE variable. The entire process is much simpler compared to the one that transposed the entire data set in the previous example.

Program 2b:
```
data dat1_bygrp2 (drop=e1-e3 i);
set dat1;
array e[3];
do i =1 to 3;
    test=cats("e", i);
    score = e[i];
    if not missing(score) then output;
end;
run;
title 'Transposing By-groups by Using Array Processing';
proc print data=dat1_bygrp2;
run;
```

Output from Program 2b:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>e1</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>e2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>e3</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>e1</td>
<td>92</td>
</tr>
<tr>
<td>5</td>
<td>Mary</td>
<td>e3</td>
<td>81</td>
</tr>
</tbody>
</table>

TRANSPOSING BY-GROUPS (MULTIPLE OBSERVATIONS PER SUBJECT)

The Dat2 data set, that is created in Program 3a, contains three test scores for each person. Notice that the second test score is not entered for Mary in the program. Program 3a transposes Dat2 by the values of the NAME variable. The PREFIX= option in the PROC TRANSPOSE statement is used to add "TEST" as the prefix for transposed variable names. The ID statement is used to specify the variable that contains the values to rename the transposed variables. The use of the ID statement is necessary in this situation due to the incomplete entry for the second exam for Mary. Without using the IDs statement, the third test score (81) for Mary will be placed in the location for the second exam in the transposed data set. The resulting transposed data set has two observations, which equals the number of BY-groups (2) times the number of variables that are transposed (1).

Program 3a:
```
data dat2;
    input name $ exam score;
datalines;
John 1 89
John 2 90
John 3 92
Mary 1 92
Mary 3 81
;```
<Sharpening Your Skills in Reshaping data: PROC TRANSPOSE vs. Array Processing>, continued

```sas
proc sort data=dat2 out=dat2_sort;
  by name;
run;

proc transpose data=dat2_sort
  out=dat2_t1 (drop=_name_)
  prefix=test;
  var score;
  by name;
  id exam;
run;

title 'DAT2 in the Original Form';
proc print data=dat2;
run;

title 'Transposing DAT2 By-groups Using PROC TRANSPOSE';
proc print data=dat2_t1;
run;

Output from Program 3a:

DAT2 in the Original Form

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>exam</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>3</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>Mary</td>
<td>1</td>
<td>92</td>
</tr>
<tr>
<td>5</td>
<td>Mary</td>
<td>3</td>
<td>81</td>
</tr>
</tbody>
</table>

Transposing DAT2 By-groups Using PROC TRANSPOSE

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test1</th>
<th>test2</th>
<th>test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>81</td>
</tr>
</tbody>
</table>

TRANSPOSING BY-GROUPS (MULTIPLE OBSERVATIONS PER SUBJECT) BY USING ARRAY PROCESSING

When transposing Dat2 by using array processing, you only need to have one array TEST which is used to create three variables TEST1-TEST3 in the transposed data set. You also need to use NAME as the by-variable. Since you are not outputting the data until all the observations have been read for each subject, it is important to retain the array TEST in the DATA step; otherwise, these newly-created variables will be set to missing at the beginning of each iteration of the DATA step. Since TEST1-TEST3 are retained, you also need to initialize the values of these three values to missing when reading the first observation of each subject; otherwise, values from previous subjects will be carried down to the current object when the SCORE variable is missing for a certain test in the currently-processed subject.

Program 3b:
```sas
data dat2_t2 (drop=exam score i);
  set dat2_sort;
  by name;
  array test[3];
  retain test;
  if first.name then do;
    do i = 1 to 3;
      test[i] = .;
    end;
```
end;
test[exam] = score;
if last.name;
run;

title 'Transposing DAT2 By-groups Using Array Processing';
proc print data=dat2_t2;
run;

Output from Program 3b:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test1</th>
<th>test2</th>
<th>test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>81</td>
</tr>
</tbody>
</table>

HANDLING DUPLICATES

HANDLING DUPLICATES BY USING PROC TRANSPOSE

Consider the example in Program 4a. There are double entries of scores for certain tests in the Dat3 data set. For situations with duplicated records, you may want to keep only one record, such as keeping the largest or the smallest of the duplicated entries. The LET option from the PROC TRANSPOSE statement allows you to keep the last occurrence of a particular ID value within either the entire data set or a BY group. Program 4a transposes Dat3 by keeping the largest value of each EXAM within each group of the NAME variable. Thus, it is necessary to sort the data by NAME first, followed by EXAM, and then SCORE in ascending order. Since the LET option only keeps the last occurrence of the ID value, PROC TRANSPOSE correctly transposes data with only the largest score within each EXAM. SAS detected the duplicated values that occurred in certain tests in the same BY group; a WARNING message is generated in the log.

Program 4a:
data dat3;
    input name $ exam score;
datalines;
John 1 89
John 2 90
John 2 89
John 3 92
John 3 95
Mary 1 92
Mary 3 81
Mary 3 85
;
proc sort data=dat3 out=dat3_sort1;
    by name exam score;
run;

proc transpose data=dat3_sort1
    out=dat3_t1 (drop=_name_)
    prefix=test
    let;
    var score;
    by name;
    id exam;
run;

title 'DAT3 in the Original Form';
proc print data=dat3;
run;
**title 'Handling Duplicates By Using the LET Option';**

**proc print data=dat3_t1;**

**run;**

Output from Program 4a:

<table>
<thead>
<tr>
<th>DAT3 in the Original Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Obs</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Handling Duplicates By Using the LET Option

<table>
<thead>
<tr>
<th><strong>Obs</strong></th>
<th><strong>name</strong></th>
<th><strong>test1</strong></th>
<th><strong>test2</strong></th>
<th><strong>test3</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>85</td>
</tr>
</tbody>
</table>

Log from Program 4a:

682 proc transpose data=dat3_sort1
683   out=dat3_t1 (drop=_name_) prefix=test
685  let;
686  var score;
687 by name;
688  id exam;
689 run;

**WARNING:** The ID value 'test' occurs twice in the same BY group.

**WARNING:** The ID value 'test2' occurs twice in the same BY group.

**NOTE:** The above message was for the following BY group:
**name=John**

**WARNING:** The ID value 'test3' occurs twice in the same BY group.

**NOTE:** The above message was for the following BY group:
**name=Mary**

**NOTE:** There were 8 observations read from the data set WORK.DAT3_SORT1.

**NOTE:** The data set WORK.DAT3_T1 has 2 observations and 4 variables.

**NOTE:** PROCEDURE TRANSPOSE used (Total process time):

<table>
<thead>
<tr>
<th>real time</th>
<th>0.00 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu time</td>
<td>0.00 seconds</td>
</tr>
</tbody>
</table>

If you want to keep the smallest SCORE instead of the largest in the transposed data, all you need to do is sort NAME and EXAM in ascending order and then sort SCORE in descending order.
HANDLING DUPLICATES BY USING ARRAY PROCESSING

Keeping the highest score for each exam can also be done by using array processing. You need to use both NAME and EXAM as the by-variables. The only array that you need is TEST, which is used to generate variables TEST1-TEST3. Just like the previous example, TEST1-TEST3 are initialized to missing values when reading the first observation of each subject. When reading the first exam for each subject, the SCORE value is assigned to the SCORE_HI variable. When there are more than one scores entered in each exam, the SCORE value will be compared with SCORE_HI. If SCORE value is greater than SCORE_HI, then you need to reassign SCORE_HI with the current SCORE value. That is to say, SCORE_HI will contain the highest score for each exam. When reading the last score of each exam, the value from SCORE_HI will be assigned to the corresponding test variable. In the end, the data will be outputted to the output data set when reading the last observation of each subject.

Program 4b:

```plaintext
data dat3_t2 (keep=name test1-test3);
  set dat3_sort1;
  by name exam;
  array test[3];
  retain test;
  if first.name then do;
    do i = 1 to 3;
      test[i] = .;
    end;
  end;
  if first.exam then do;
    score_hi = score;
  end;
  if not first.exam then do;
    if score>score_hi then score_hi = score;
  end;
  if last.exam then do;
    test[exam] = score_hi;
  end;
  if last.name;
run;
```

title 'Handling Duplicates By Using the Array Processing';
proc print data=dat3_t2;
run;

Output from Program 4b:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test1</th>
<th>test2</th>
<th>test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>90</td>
<td>95</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>85</td>
</tr>
</tbody>
</table>

Using the LET option from the PROC TRANSPOSE statement can only allow you to keep the highest or the lowest score within each exam. Suppose that you would like to use the mean scores for duplicated scores of each exam; you can only use array processing in the DATA step to accomplish this task. For example, Program 4c utilizes array processing to transpose Dat3 that contains the mean scores for duplicated entries. The approach in Program 4c is very similar to Program 4b. When reading the first entry score of each exam, variable SCORE_TOTAL (for accumulating total scores) and variable EXAM_N (for counting the number of duplicated entries) are initialized to 0. During each iteration of the DATA step, SCORE_TOTAL is accumulated with SCORE value and EXAM_N is incremented by 1. When the last entry score is encountered, the SCORE_MEAN variable (for mean of the scores) is calculated and assigned to the corresponding test.
Program 4c:

data dat3_t3 (keep=name test1-test3);
set dat3_sort1;
by name exam;
array test[3];
retain test;
if first.name then do;
  do i = 1 to 3;
  test[i] = .;
  end;
if first.exam then do;
  score_total = 0;
  exam_n = 0;
  end;
score_total + score;
  exam_n + 1;
if last.exam then do;
  score_mean = score_total/exam_n;
  test[exam] = score_mean;
end;
if last.name;
run;

title 'Keeping the Mean Score By Using the Array Processing';
proc print data=dat3_t3;
run;

Output from Program 4c:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>test1</th>
<th>test2</th>
<th>test3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>89</td>
<td>89.5</td>
<td>93.5</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>92</td>
<td>.</td>
<td>83.0</td>
</tr>
</tbody>
</table>

RESHAPING DATA WHICH REQUIRES TRANPOSING DATA MORE THAN ONCE
RESHAPING DATA BY USING PROC TRANSPOSE

In some applications, simply transposing data once will not produce the desired results. For example, to transpose Dat4 to Dat4_Reshape (See Figure 2, page2), you need to use PROC TRANSPOSE twice.

Program 5a1 transposes Dat4 by variable NAME. In the next step, you need to transpose COL1 from Dat4_out1 into three rows. Before performing a second transposing, you need to sort the data by the test number and NAME. For example, the first observation (John, E1) should be followed by the 4th, 7th, and 10th rows. You also need to create a variable that contains the test number, which is the last character of the _NAME_ variable in Dat4_out1.

Program 5a1:

data dat4;
  input name $ e1-e3 m1-m3;
dataines;
  John 89 90 92 78 89 90
  Mary 92 . 81 76 91 89
;proc sort data=dat4 out=dat4_sort1;
  by name;
run;
proc transpose data=dat4_sort1 out=dat4_out1;
  by name;
run;

title 'First use of PROC TRANSPOSE for dat4';
proc print data=dat4_out1;
run;

Output from Program 5a1:

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th><em>NAME</em></th>
<th>COL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>e1</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>e2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>e3</td>
<td>92</td>
</tr>
<tr>
<td>4</td>
<td>John</td>
<td>m1</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td>John</td>
<td>m2</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>m3</td>
<td>90</td>
</tr>
<tr>
<td>7</td>
<td>Mary</td>
<td>e1</td>
<td>92</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>e2</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>Mary</td>
<td>e3</td>
<td>81</td>
</tr>
<tr>
<td>10</td>
<td>Mary</td>
<td>m1</td>
<td>76</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
<td>m2</td>
<td>91</td>
</tr>
<tr>
<td>12</td>
<td>Mary</td>
<td>m3</td>
<td>89</td>
</tr>
</tbody>
</table>

Program 5a2 uses the SUBSTR function to create the TEST_NUM and CLASS variables by taking the last
and first characters of the _NAME_ variable.

Program 5a2:

data dat4_out1a;
  set dat4_out1;
  test_num=substr(_name_,2);
  class=substr(_name_,1,1);
run;

title 'Creating TEST_NUM and CLASS variables';
proc print data=dat4_out1a;
run;

Output from Program 5a2:

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th><em>NAME</em></th>
<th>COL1</th>
<th>test_num</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>e1</td>
<td>89</td>
<td>1</td>
<td>e</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>e2</td>
<td>90</td>
<td>2</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>John</td>
<td>e3</td>
<td>92</td>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>4</td>
<td>John</td>
<td>m1</td>
<td>78</td>
<td>1</td>
<td>m</td>
</tr>
<tr>
<td>5</td>
<td>John</td>
<td>m2</td>
<td>89</td>
<td>2</td>
<td>m</td>
</tr>
<tr>
<td>6</td>
<td>John</td>
<td>m3</td>
<td>90</td>
<td>3</td>
<td>m</td>
</tr>
<tr>
<td>7</td>
<td>Mary</td>
<td>e1</td>
<td>92</td>
<td>1</td>
<td>e</td>
</tr>
<tr>
<td>8</td>
<td>Mary</td>
<td>e2</td>
<td>.</td>
<td>2</td>
<td>e</td>
</tr>
<tr>
<td>9</td>
<td>Mary</td>
<td>e3</td>
<td>81</td>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>10</td>
<td>Mary</td>
<td>m1</td>
<td>76</td>
<td>1</td>
<td>m</td>
</tr>
<tr>
<td>11</td>
<td>Mary</td>
<td>m2</td>
<td>91</td>
<td>2</td>
<td>m</td>
</tr>
<tr>
<td>12</td>
<td>Mary</td>
<td>m3</td>
<td>89</td>
<td>3</td>
<td>m</td>
</tr>
</tbody>
</table>
<Sharpening Your Skills in Reshaping data: PROC TRANSPOSE vs. Array Processing>, continued

Program 5a3 sorts the data by TEST_NUM and NAME. Notice that the test scores in COL1 have the desired order.

Program 5a3:
```
proc sort data=dat4_out1a out=dat4_sort2;
  by test_num name;
run;
```
```
title 'Sort data by TEST_NUM and NAME';
proc print data=dat4_sort2;
run;
```

Output from Program 5a3:

```
Sort data by TEST_NUM and NAME
Obs name _NAME_ COL1 test_num class
1 John e1 89 1 e
2 John m1 78 1 m
3 Mary e1 92 1 e
4 Mary m1 76 1 m
5 John e2 90 2 e
6 John m2 89 2 m
7 Mary e2 . 2 e
8 Mary m2 91 2 m
9 John e3 92 3 e
10 John m3 90 3 m
11 Mary e3 81 3 e
12 Mary m3 89 3 m
```

PROC TRANSPOSE in Program 5a4 transposes COL1 by variable TEST and uses NAME and CLASS as the ID variables. The names of the transposed variables are separated by the underscore from the DELIMITER= option.

Program 5a4:
```
proc transpose data=dat4_sort2
  out=dat4_t1(drop=_name_)
  delimiter=_;
  by test_num;
  var col1;
  id name class;
run;
```
```
title 'Second use of PROC TRANSPOSE for dat4';
proc print data=dat4_t1;
run;
```

Output from Program 5a4:

```
Second use of PROC TRANSPOSE for dat4
Obs test_num John_e John_m Mary_e Mary_m
1 1  89    78    92    76
2 2  90    89     .    91
3 3  92    90    81    89
```
RESHAPING DATA BY USING ARRAY PROCESSING

Program 5b, which transposes Dat4 by using array processing, appears to be complicated. However, the idea for transposing Dat4 is similar to transposing the entire data set in Program 1b. Similar to what we did in Program 1b, three arrays are needed for the transposing process; but each of these arrays has an additional dimension because there are two types of tests (E & M) in Dat4 instead of only one type in Dat1. The detailed explanation of this program will not be reviewed here.

Program 5b:

```sas
data dat4_t2 (keep=test_num John_e John_m Mary_e Mary_m);
  set dat4 end=last;
  array score [2,3] e1-e3 m1-m3;
  array all [2,2,3] _temporary_; 
  array new_score[2,2] _temporary_; John_e John_m Mary_e Mary_m;
  retain all; i + 1;
  do j = 1 to 2;
    do k = 1 to 3;
      all[i,j,k]=score[j,k];
    end;
  end;
  if last then do;
    do i = 1 to 2;
      do j = 1 to 2;
        new_score[i,j] =all[i,j,k];
      end;
    end;
    test_num = k;
  output;
  end;
run;

title 'Reshaping Data By Using Array Processing';
proc print data=dat4_t2;
run;
```

Output from Program 5b:

```
<table>
<thead>
<tr>
<th>Obs</th>
<th>John_e</th>
<th>John_m</th>
<th>Mary_e</th>
<th>Mary_m</th>
<th>test_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89</td>
<td>78</td>
<td>92</td>
<td>76</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>89</td>
<td>.</td>
<td>91</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>90</td>
<td>81</td>
<td>89</td>
<td>3</td>
</tr>
</tbody>
</table>
```

CONCLUSION

Simulation has shown that using PROC TRANSPOSE is more efficient compared to using array processing in the DATA step. Furthermore, the syntax of PROC TRANSPOSE is much simpler, especially for simple data transposition. On the other hand, using DATA step allows you to perform table lookups, add new variables in the output data, and conduct more flexible calculations. You should choose the right method depending on your programming objectives and efficiency.

REFERENCES

ACKNOWLEDGMENTS
I would like to thank Jim Groeneveld, John King (Data _null_), Joseph Hinson, and Ron Fehd in SAS-L for providing valuable discussions on transposing data.

CONTACT INFORMATION
Arthur Li
City of Hope Comprehensive Cancer Center
Division of Information Science
1500 East Duarte Road
Duarte, CA 91010 - 3000
Work Phone: (626) 256-4673 ext. 65121
Fax: (626) 471-7106
E-mail: arhuri@coh.org

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.