STAT 541

Chapters 11 & 12: Arrays, PROC TRANSPOSE, and Hash Objects

One-Dimensional Arrays

- An array is a group of SAS columns.
- Using an array can facilitate repetitive programming tasks.
- You define an array with an ARRAY statement.
- You can work with elements of the array by referencing the array and listing the desired column(s) in square brackets.

Purpose of arrays

- Arrays are often used in DO loops to perform the same task on numerous elements (columns) of the array.
- When initializing the array in the ARRAY statement, you give the name of the array, the number of elements, and the names of the columns.
- The column names could be listed explicitly or by using shortcut lists.

Array Elements

- If the number of array elements is unknown, this number can be specified with an asterisk *.
- The DIM function will return the number of elements in a created array.
- You can explicitly specify the lower and upper bounds of the index of the array (see lookup table example later).

Initial Values

- You can provide initial values of the elements of the array in parentheses at the end of the ARRAY statement.
- Example: `ARRAY Goal[3] (200 250 300);`
- Or if the values do not need to be kept in the final table, temporary initial values can be specified:
- `ARRAY Names[25] $15 _temporary_;`

Using Lookup Tables to Match Data

- Multidimensional arrays
- PROC TRANSPOSE
- Hash objects

Using Multidimensional Arrays

ARRAY *array-name [rows, cols,…]<$><length>*

{ *subscript* } <$><*length*> <*array-elements*> <(*initial-value-list*)>;

- *array-name* names the array
- *rows* specifies the number of elements in a row arrangement
- *cols* specifies the number of elements in a column arrangement
- *array-elements* names the variables that make up the array
- *initial values* specifies initial values for the corresponding elements in the array that are separated by commas or spaces

When Working with Arrays

- The name of the array cannot be the name of a SAS variable in the DATA step.
- The variables listed as array elements must all be the same type (either all numeric or all character).
- The initial values specified can be numbers or character strings. Enclose all character strings in quotation marks.

Keyword _TEMPORARY_

- The keyword can be used in lieu of listing *array-elements* in the syntax, which would avoid creating new data set variables. Only temporary elements are produced using this keyword.
- The temporary elements behave like DATA SET variables, but they don't have names. Refer to the elements using the array name and dimension. Because they are not DATA set variables, they do not appear in the output data set.

Scoring Example with 2-Dimensional SAS Array

	Adjusted Score	Raw Score on a Test				
Rule for Obtaining Adjusted Score	Age	1	2	3	4	5
raw score + 3 (not to exceed 5)	13	4	5	5	5	5
raw score +2 (not to exceed 5)	14	3	4	5	5	5
raw score + 1 (not to exceed 5)	15	2	3	4	5	5

```
data one;
input age score;
array answer {13:15,1:5} _temporary_
        (4 5 5 5 5
         3 4 5 5 5
         2 3 4 5 5);
adjusted = answer(age,score);
```

Scoring Example with 3-Dimensional SAS Array

Adjusted Score	Female Student's Raw Score				
Age	1	2	3	4	5
12.5	4	4	5	5	5
13	3	4	4	5	5
13.5	2	3	4	4	5
Adjusted Score	Male Student's Raw Score				
Age	1	2	3	4	5
12.5	4	5	5	5	5
13	3	4	5	5	5
13.5	2	3	4	5	5

Used INPUT function and informats to specify correct array index values.

```
proc format;
invalue gender 'F'=1 'M'=2;
invalue age 12.5=1 13.0=2 13.5=3;

data one;
set inputdata;
array answer {1:2,0:3,0:6} _temporary_
        (1    1 2 3 4 5 .
         1    4 4 5 5 5 12.5
         1    3 4 4 5 5 13.0
         1    2 3 4 4 5 13.5
         2    1 2 3 4 5 .
         2    4 5 5 5 5 12.5
         2    3 4 5 5 5 13.0
         2    2 3 4 5 5 13.5);
adjusted=answer(input(sex,gender.),
input(age,age.),score);
```

Using Stored Array Values

- Arrays can be stored in a data set. Reasons for doing so are:
  - There might be too many values to initialize easily in the array.
  - The values change frequently.
  - The same values are used in many programs.
  - Example

Using Stored Array Values

```
data twoadj (keep=age score finalscore);
   array adj{13:15,5} _temporary_;
if _n_=1 then do i=1 to 3;
set lookup;
```

Using Stored Array Values

```
array adjscore{*} adjscore1-adjscore5;
do j=1 to dim(adjscore);
adj{age,j}=adjscore{j};
end; end;
set two;
finalscore=adj{age,score};
```

Using PROC TRANSPOSE

- PROC TRANSPOSE transforms horizontal data sets to vertical data sets and vice versa
- This makes it ideal for match-merging data stored in different formats
- Remember that PROC TRANSPOSE requires much “clean-up” of intermediate data sets

Using PROC TRANSPOSE

ID	Length (mm)	Weight (g)
1	523	1340
2	535	1297
3	397	1020
4	615	2115

Age	Weight				
Length	<900	900-1300	1300-2000	2000-2500	>2500
<400	1	1			
400-500	1	2	2	3	
500-600		2	3	3	4
600-675			3	4	4
>675			3	4	5

Using PROC TRANSPOSE

```
data agechart;
input length weight1-weight5;
proc sort data=agechart; by length;
proc transpose data=agechart out=tchart
   (rename=(age1=age)) name=weight
   prefix=age;
by length;
```

Using Hash Objects

- Temporary storage object
- Flexible formatting and indexing
- Remember that PROC TRANSPOSE requires much “clean-up” of intermediate data sets

Hash Object Structure

- Key component
  - must be unique
  - maps data values to data set
- Data component
  - "ragged array" of numeric or character values
- Is a DATA step object with specific *attributes* and *methods*

Hash Object

- There are several steps to creation of a hash object
  - DECLARE (i.e., define) the hash object
  - Define and load keys and data
- Matched data can then be retrieved

Hash Object

- Gym members example
  - Essentially a match merge
  - The book uses the data component more actively than in this example

Hash Object

```
data workout;
format name $20.;
if _N_=1 then do;
declare  hash members();
members.definekey("id");
members.definedata("name");
members.definedone();
call missing(id,name);
members.add(key:787,data:'Sam Crump');
...
members.add(key:9877,data:'Ron cole');
end;
set attendance;
members.find();
run;
```

Hash Object

- Hash objects can also be retrieved from existing SAS data sets
  - Keys can be used to retrieve data values for more than one variable
  - Keys can be assigned to more than one data component

Hash Iterator Objects

- A hash iterator object (or hiter) is associated with a hash object and allows you to retrieve data from the hash object in forward or reverse order with respect to the key.
- After declaring the hash object, (say, `myhash`): `DECLARE hiter C('myhash');` creates a hash object iterator associated with `myhash`.

Using hash iterator objects

- Then `C.first();` returns the first data value in the hash object.
- `C.next();` returns the next value (in order of the key variable) in the hash object.
- `C.prev();` returns the previous value, in reverse key order.
- `C.last();` returns the last value in the hash object.