Intro to Arrays: A Grad Student "How-To" Paper

Elisa L. Priest[1,2] , Jennifer S. Harper[1]

[1] Institute for Health Care Research and Improvement, Baylor Health Care System

[2] University of North Texas School of Public Health, Epidemiology Department

ABSTRACT

Grad students learn the basics of SAS programming in class or on their own. Often the lessons focus on statistical procedures. Data management techniques are limited to the DATA step and common methods like functions to create new datasets and variables. IF—THEN statements are also introduced as a simple way to create and recode variables. These programming tools allow students to perform most simple data management tasks needed to prepare data for analysis. Although these tools are sufficient for in-class exercises with small datasets, new tools such as arrays may be better suited for larger datasets.

Once students begin working on their own research projects with 'real' data, they quickly realize that manipulating the data so it can be analyzed is its own time consuming challenge. In the real world of messy data, we often spend more time preparing the data than performing the analysis. Students approach the task of recoding or performing a calculation on 100 variables in the same way they have been taught on 10 variables: copy and paste sections of code and make a change in the variable name. This method can be error prone, difficult to troubleshoot, and time consuming.

One tool that can save beginning programmers hours of effort and pages of code is a simple array. Arrays are especially high-impact when you have a large group of similar variables and you want to 'do the same thing' to all of the variables. This could be performing a calculation, using a SAS function, or simple recoding. Arrays can also be used to create new variables or to restructure datasets. This paper will teach beginning programmers to simplify their code with arrays and will use real world examples from the authors' experiences.

This paper is the fourth in the 'Grad Student How-To' series[1,2,3] and gives graduate students a useful tool for writing more efficient code with arrays.

OUTLINE

1. YOU MIGHT NEED AN ARRAY IF…

You have a list of variables and:

- You need to perform the same comparison, calculation, or function on all of them
- You need to recode all of them, like reverse the numbering
- You need to code all the '.' (numeric missing) or ''(character null) to a missing value like -9999 or vice versa
- You want to look through a list of variables and find a specific value or create a flag
- You have multiple records and you want to make a single record
- You have a single record and you want to make multiple records

Other signs you might need an array*

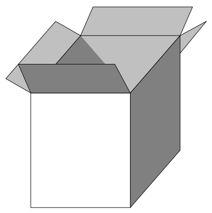
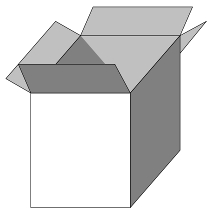
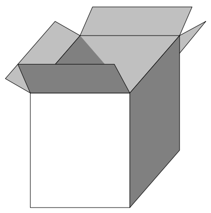
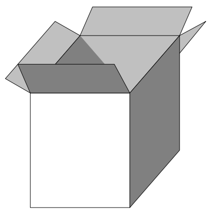
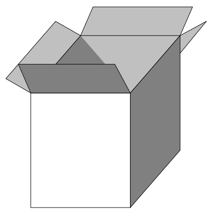
- You find yourself cutting and pasting code over and over
- You are ~~too lazy~~ too efficient to waste time typing repetitive code

**these are also signs you might need to learn basic macros, but that is another paper!*

2. HOW IT WORKS

To fully understand how an array works, you need to know how SAS compiles and executes programs. You can find this information in the SAS 9.2 Language Reference.[4,5] To understand how an array works well enough to use one most of the time, you need to know the following:

I imagine an array as a group of buckets or boxes. You give the array a name so that you have a 'nickname' way to reference each box. Next, you tell SAS how many boxes you need and what values to put into the boxes. Technically speaking, each of these boxes is called an 'element' and the nickname is the 'Array reference'. In the diagram below, I have an array called 'box' with 5 elements.

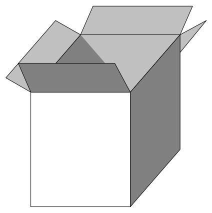
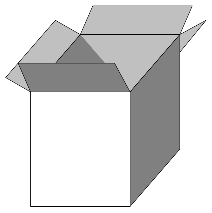
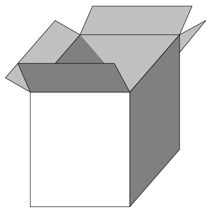
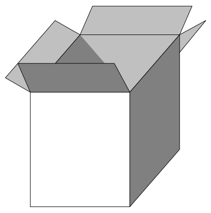
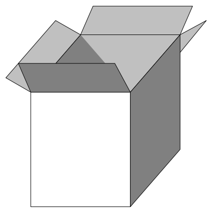
ARRAY: BOX					
Array Reference	**BOX[1]**	**BOX[2]**	**BOX[3]**	**BOX[4]**	**BOX[5]**
Element					

Now, let's show how this applies to variables in a SAS dataset. In this example, I have a simple laboratory values dataset with 6 variables: ID (participant ID), WBC (White Blood Cells), RBC (Red Blood Cells), HGB (Hemoglobin), HCT (Hematocrit), and Platelets. I tell SAS I want an array named 'LAB' with 5 elements. I also want the values of my 5 numeric lab measurements stored in those elements. In my data step, I define the array:

```
ARRAY LAB[5] WBC RBC HGB HCT Platelets;
```

work.laboratory_values

OBS	ID	WBC	RBC	HGB	HCT	Platelets
1	001	6.9	4.88	-9999	40.9	-9999
2	002	5.6	4.47	15.2	43.5	246
3	003	7.1	5.05	14.4	48.6	221

ARRAY: LAB					
Array Reference	**LAB[1]**	**LAB [2]**	**LAB[3]**	**LAB[4]**	**LAB[5]**
Element					
VAR	WBC	RBC	HGB	HCT	Platelets
Value 001	6.9	4.88	-9999	40.9	-9999
Value 002	5.6	4.47	15.2	43.5	246

As the data step executes the first time, each of the values from the laboratory variables for the first observation (participant ID 001) is now in the 'boxes' or elements of the array called 'LAB'. So now what? Now, we can use the nickname or 'array reference' instead of having to write out each variable name and we can use this to simplify our code. After the data step is done with the first observation, all the boxes are emptied out and as the data step executes for the second time, the values for the second observation (participant 002) are loaded in.
In this laboratory dataset, the data management group coded the missing values using -9999 to indicate that

these values are permanently missing. However, for our statistical analysis, we want these to be the SAS numeric missing value ‘.’. If we want to recode all of the missing values for all of these variables, we only have to write the SAS code one time instead of five times! This is usually done with a DO loop.

```
Do I= 1 to 5;
   If LAB[I] =-9999 then LAB[I]=.;
End;
```

This loop will run five times and the counter variable, I, will increment by 1 each time.

First loop, I=1:

```
   If LAB[1] =-9999 then LAB[1]=.;
```

In the first loop, LAB[1] is the value from WBC, 6.9. Since it does not =-9999, it stays the same

Second loop, I=2

```
   If LAB[2] =-9999 then LAB[2]=.;
```

In the second loop, LAB[2] is the value from RBC, 4.88. Since it does not equal -9999, it stays the same.

Third loop, I=3

```
   If LAB[3] =-9999 then LAB[3]=.;
```

In the third loop, LAB[3] is -9999, so it is recoded to the numeric missing ‘.’ .

Fourth loop, I=4

```
   If LAB[4] =-9999 then LAB[4]=.;
```

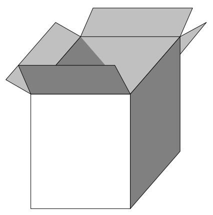
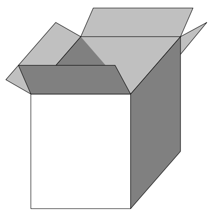
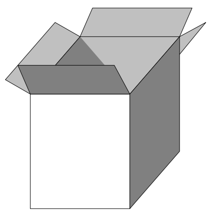
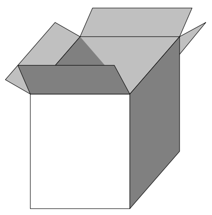
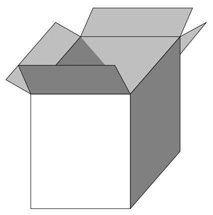
In the fourth loop, LAB[4] is 40.9. . Since it does not =-9999, it stays the same.

Fifth loop, I=5

```
   If LAB[5] =-9999 then LAB[5]=.;
```

In the fifth loop, LAB[5] is -9999, so it is recoded to the numeric missing ‘.’ .

The original and final values from the do loop processing are shown below. This example used a simple DO loop, which is what I usually use with arrays. However, you can use more complex versions of the DO, DO WHILE, DO UNTIL loops. See the SAS Language Reference for more details.[4,6]

ARRAY: LAB					
Array Reference	**LAB[1]**	**LAB [2]**	**LAB[3]**	**LAB[4]**	**LAB[5]**
Element					
VAR	WBC	RBC	HGB	HCT	Platelets
Value 001 original	6.9	4.88	-9999	40.9	-9999
Value 001 After	6.9	4.88	**.**	40.9	**.**

3. HOW TO CODE IT

DEFINE IT

The syntax to define the array that we used in the previous example is straightforward.

```
ARRAY LAB[5] WBC RBC HGB HCT Platelets;
```

ARRAY *array-name* {*number-of-elements*} <*array-elements*> ;

There are also additional options for defining the array.

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

Like all SAS code, there are certain rules that should be used.

1. Array name
   a. Follow the same rules as for naming variables
   b. do not use the same name as a variable or a function
2. Subscript/Number of elements: the number and arrangement of array elements
   a. The number of array elements {4} or (4) or [4]
   b. A range of numbers {100:110}
   c. {*} this notation makes SAS count the number of elements
3. $
   a. An array must be all numeric or all character
   b. Character arrays use the $
4. Length
   a. Used with character arrays
5. Elements
   a. List each variable name

   b. A variable list of variables in the dataset (all numeric or all character)
      var1-var5
      _NUMERIC
      CHARACTER
      ALL

6. Initial Value List
   a. Instead of having an array use values from variables, you assign them.

REFERENCE IT

Reference your array element in the same data step that it was defined in. It only exists in that data step.
To reference an array element:

```
If LAB[I] =-9999 then LAB[I]=.;
```

array-name { *subscript* }

4. EXAMPLES

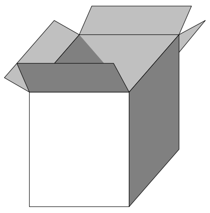
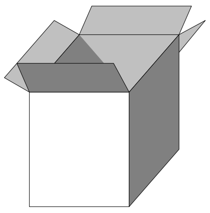
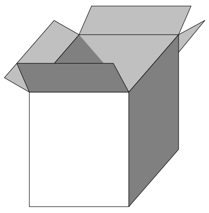
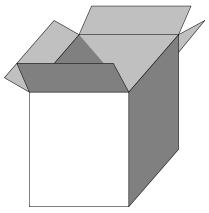
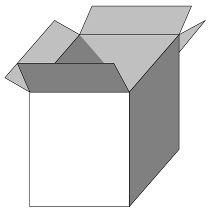
Now, let's look at examples of using arrays in real situations

- Recode and calculate values
- Create simple counts and flags
- More complex flags using _TEMPORARY_ arrays
- Multiple arrays, PROC COMPARE, and flags
- Create new variables and multiple arrays
- Compare values across multiple arrays
- Restructure datasets

RECODE AND CALCULATE VALUES

This is the most common way that I've used arrays. It is also where you can get a lot of impact from very little code. Often, you will see examples where all the values of the variables in a likert-scale survey need to be reversed. In section 2, we saw an array with a simple DO loop and IF—THEN statement to recode multiple laboratory variables.

```
ARRAY LAB[5] WBC RBC HGB HCT Platelets;
Do I= 1 to 5;
   If LAB[I] =-9999 then LAB[I]=.;
End;
```

ARRAY: LAB					
Array Reference	**LAB[1]**	**LAB [2]**	**LAB[3]**	**LAB[4]**	**LAB[5]**
Element					
VAR	WBC	RBC	HGB	HCT	Platelets
Value 001 original	6.9	4.88	-9999	40.9	-9999
Value 001 After	6.9	4.88	.	40.9	.

We could also modify this DO loop to include additional statements with calculations or functions. For example, a dataset was produced from microarray data. Each variable contains expression level data for an individual gene. These values needed to be transformed for analysis. This code contains two interesting features. First, each of the individual genes are not listed. Instead, we used a shortcut notation of Gene1-Gene225. These variables are all consecutive in the dataset, so we used a shortcut to reference them all instead of typing (Gene1-Gene225). Also, instead of specifying the number of elements in the array, we tell SAS to figure it out form the variable list. We do this by using the asterisk [*] and then use the DIM function to count the number of elements in the array.

```
ARRAY gene [*] Gene1-Gene225;
do j= 1 to DIM(gene);
      if gene[j]=0 then gene[j]=0.5;
      gene[j]=log(gene[j]);
end;
```

CREATE SIMPLE COUNTS AND FLAGS

Another similar application of the array is to create a variable that counts the values or flags the in an array that meet a certain criteria. For example, I had to count the number of tests that had a value in a large laboratory processing dataset with 112 variables. Also, my laboratory variables were all consecutive in the dataset, so I used the shortcut WBC—CRP to reference the entire list instead of typing it.

```
*First, initialize the Total to 0;
TestsTotal=0;

*array*;
Array Tests [112] WBC--CRP;
*do loop processing to count tests*;
Do I = 1 to 112;
       If Tests[I]  >0 then TestsTotal+1;
end;
```

Example data for simple counts and flags

OBS	ID	WBC	RBC	HGB	HCT	Platelets	………..	TestsTotal
1	001	6.9	4.88	-9999	40.9	-9999	………..	25
2	002	5.6	4.47	15.2	43.5	246	………..	9
3	003	7.1	5.05	14.4	48.6	221	………..	18

MORE COMPLEX FLAGS USING _TEMPORARY_ ARRAYS

This example is from the same project of processing data from a laboratory information system. We were screening lab test results and wanted to create a flag when the footnote for the test contained a specific word that indicated that the sample was bad. We had 15 words that indicated that a sample was bad. These were read into an array and each footnote was screened for those words.

In this array example, we use a temporary, character array and assign values to the array elements. A temporary array is useful if you do not need to keep any of the values in your array in the final dataset. For example, you could create a temporary array to store values for a calculation. The code to define this array is a little more complicated than the basic array we have been using. We must tell SAS 1) the array is temporary 2) the array is character (use $) and 3) the values for the elements in the array.

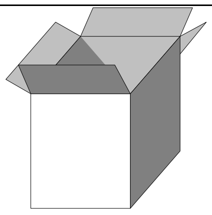
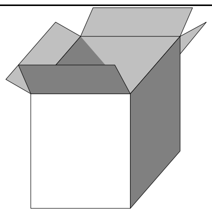
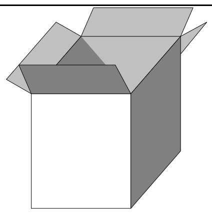
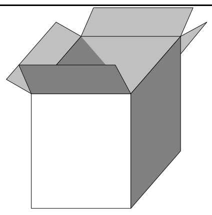
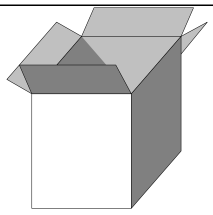
There are a few considerations for a temporary array. 1) temporary array element do not appear in the output data set. 2) temporary array elements do not have names 3) temporary array elements are automatically retained instead of being reset to missing at the beginning of each iteration of the DATA step.

ARRAY *_TEMPORARY_* <$> <*length*> <(*initial-value-list*)>;

```
*First, upcase the footnote so it is all standard case*;
Footnote=upcase(Footnote);

*Next, define the array as temporary and allow 40 characters per element*
array _TEMPORARY_  $40
("RECOLLECTED", "AIRFUGED","WRONG","SOURCE UNKNOWN", "BROWN SERUM",
"ENCOUNTER", "CONTAMINATED", "MICROTAINER","MISLABELED", "MISLABELLED",
"CONTAMINATION", "BLOCKED", "ERROR", "CREDIT", "INCORRECT");
Do I=1 to 15;
       IF index (Footnote,strip(_TEMPORARY_[i])) then footnoteflag=1 ;
End;
```

The output dataset and the first 5 elements of the temporary array are shown below. As each do loop executes, the function INDEX is used to see if the variable footnote contains the words loaded into the _TEMPORARY_ array.

OBS	ID	FOOTNOTE	FOOTNOTEFLAG
1	001	ADEQUATE	
2	002	SAMPLE WAS RECOLLECTED	1
3	003	WRONG TUBE COLLECTED	1

ARRAY: _TEMPORARY_					
Array Reference	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**
Element					
Value	RECOLLECTED	AIRFUGED	WRONG	SOURCE UNKNOWN	BROWN SERUM

MULTIPLE ARRAYS, PROC COMPARE, AND FLAGS

This dataset was part of another laboratory data processing project. We were screening lab tests for duplicates. These duplicates had the same test name and accession number, and we were trying to distinguish why these duplicates were in the data and we hoped to identify which duplicate value was correct value. When the data was output from the laboratory information system, they did not extract any data on record update date/time. We output the duplicates into two separate files (dup_problems1 and dup_problems2) and used PROC COMPARE to flag the differences. Proc compare created a dataset called dup.compare, This dataset contains observations that flag the differences in values between the two datasets. Differences in character values are indicated with a ‘XXX’ and differences in numeric values are indicated with a number. We created two arrays and two flag variables, one for the character variables, and one for the numeric variables we were interested in. Finally, we output the duplicates to another dataset where they could be examined.

This example illustrates a more complicated scenario that integrates the use of more than one simple array.

```
*****Compare duplicates and create compare dup_problems1 and 2***;
PROC COMPARE base=work.dup_problems1 compare=work.dup_problems2 maxprint=5000
out=work.dup_compare noprint outnoequal outbase outcomp outdif;
id accession test_name;
run;

*****Look at compare file and create list of accession numbers to delete***;
data work.dup_compare1 work.dup_compare2;
set work.dup_compare;

if _TYPE_="DIF";
*initialize flags*;
DupNum=.; DupChar=.;

Array CompareNum [8] BIRTHDATE FIN_NBR CLIENT ADMIT_DATE DISCHARGE_DATE DRAWN_DATE
DRAWN_TIME RESULT ;
Array CompareChar [8] $ PAT_NAME GENDER MED_NBR PATIENT_TYPE MEDICAL_SERVICE
ORDER_LOCATION RESULT_C DTLNBR;
```

```
Do I= 1 to 8;
       if compareNum(i)>.Z then DupNum=1;
       else if index (comparechar(i), "X") then DupChar=1 ;

end;
If dupnum=1 or dupchar=1 then output work.dup_compare1;
else output work.dup_compare2;

run;
```

CREATE NEW VARIABLES AND MULTIPLE ARRAYS

This dataset was a set of medication information from a clinical trial. Participants could record up to 18 different medications. The investigator wanted a prefix "0" in all of the variable names to indicate the baseline visit. Thus, all the variables needed to be renamed. Originally, this code was done by hand and similar code was repeated throughout the database. The code was reduced significantly using arrays and I wish I had known arrays! This example creates new variables with new names, but there are other ways to rename variables.

To create variables with an array, specify names for variables that do not exist in the database. These variables will be numbered sequentially. For example, we created new variables for medication name that were called medtyp01, medtype02..and so on.

```
Array Med_type_new     (18) $ medtyp01-medtyp018;
```

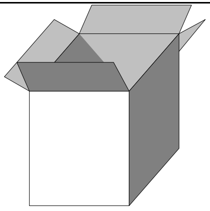
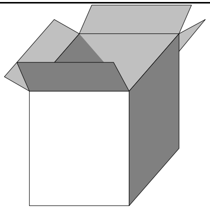
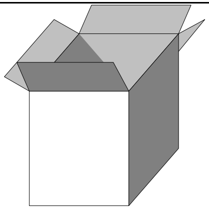
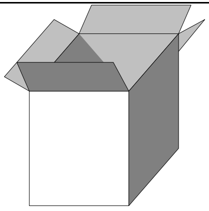
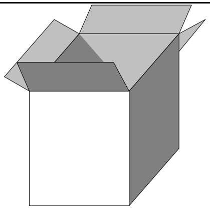
```
**creating variables**;
MEDTYP01=MEDTYP1; MEDDOS01=MEDDOS1; MEDDOSDAY01=DOSDAY1; MEDBGYR01=MDBGYR1; MEDENDYR01=MDENYR1;
MEDTYP02=MEDTYP2; MEDDOS02=MEDDOS2; MEDDOSDAY02=DOSDAY2; MEDBGYR02=MDBGYR2; MEDENDYR02=MDENYR2;
MEDTYP03=MEDTYP3; MEDDOS03=MEDDOS3; MEDDOSDAY03=DOSDAY3;MEDBGYR03=MDBGYR3; MEDENDYR03=MDENYR3;
MEDTYP04=MEDTYP4; MEDDOS04=MEDDOS4; MEDDOSDAY04=DOSDAY4; MEDBGYR04=MDBGYR4; MEDENDYR04=MDENYR4;
MEDTYP05=MEDTYP5; MEDDOS05=MEDDOS5; MEDDOSDAY05=DOSDAY5; MEDBGYR05=MDBGYR5; MEDENDYR05=MDENYR5;
***repeat this code***
```

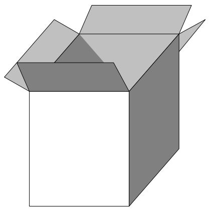
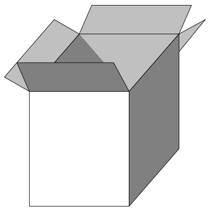
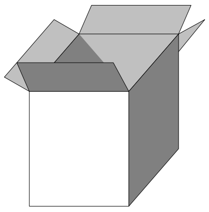
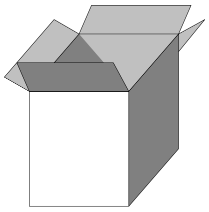
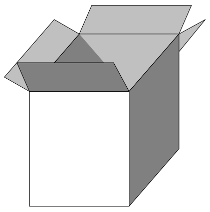
For the code with arrays, we defined 10 different arrays. The first 5 arrays were the old variables, and the next 5 arrays were the new variables that we wanted to create.

```
**new variables**;
Array Med_type_new     (18) $ Medtyp01-medtyp018;
Array Med_dose_new     (18) $ Meddos01-meddos018;
Array Med_begin_new    (18)   Medbgyr01-medbgyr018;
Array Med_end_new      (18)   Medendyr01-medendyr018;
Array Med_dose_day_new (18)   Meddosday01-meddosday018;

**original variables**;
Array Med_type     (18)$ MEDTYP1-MEDTYP18;
Array Med_dose     (18)$ MEDDOS1-MEDDOS18;
Array Med_begin    (18)  MDBGYR1-MDBGYR18;
Array Med_end      (18)  Mdenyr1-mdenyr18;
Array Med_dose_day (18)  Dosday1-dosday18;
```

In the DO loop, the values from the original arrays are copied into the new arrays and then only the values from the new arrays are kept. In the figure we see the first five elements for array Med_Type (original variable names) and Med_Type_New (new variable names). You can see that the elements in Med_Type_New are null and then contain the values from the Med_Type array.

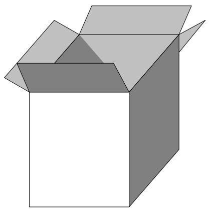
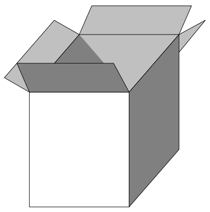
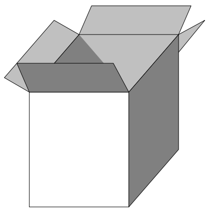
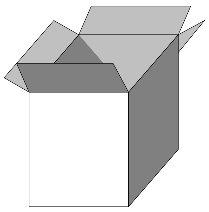
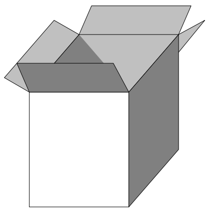
```
**do loop**;
do I =1 to 18;
  Med_type_new[i]=Med_type[i];
  med_dose_new[i]=med_dose[i];
  med_begin_new[i]=med_begin[i];
  med_end_new[i]=med_end[i];
  Med_dose_day_new [i]=med_dose_day[i];
end;
```

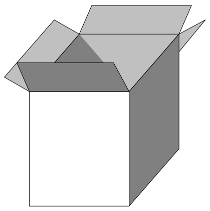
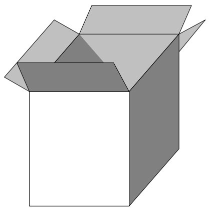
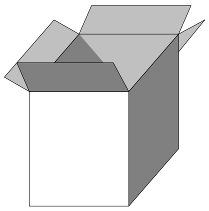
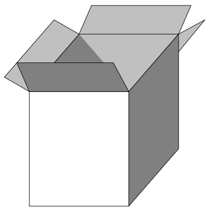
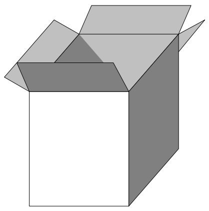
ARRAY: MED_TYPE					
Array Reference	**MED_TYPE[1]**	**MED_TYPE[2]**	**MED_TYPE[3]**	**MED_TYPE[4]**	**MED_TYPE[5]**
Element					
Var	MED_TYP1	MED_TYP2	MED_TYP3	MED_TYP4	MED_TYP5
ID 001	SYNTHROID	MULTIVITAMIN	CALCIUM	ADVAIR	VITAMIN C

ARRAY: MED_TYPE_NEW					
Array Reference	**MED_TYPE_NEW[1]**	**MED_TYPE_NEW[2]**	**MED_TYPE_NEW[3]**	**MED_TYPE_NEW[4]**	**MED_TYPE_NEW[5]**
Element					
New Var	MedTyp01	MedTyp02	MedTyp03	MedTyp04	MedTyp05
ID 001	SYNTHROID	MULTIVITAMIN	CALCIUM	ADVAIR	VITAMIN C

Similarly, you can use one or more simple arrays to recode and categorize data. This example is from a survey of beliefs about doctor patient relationships among dental professionals. We are recoding the variables in array 'original' and creating a set of new variables called 'q'. Notice that instead of specifying the names of each of the variables in the new array, SAS will automatically use the prefix of the array name plus the element number for the names of the variables.

```
data b;  set a;
     array original{27} v01_13-v01_39;
     array q {27};
  do i=1 to 27;
     if original{i} in (1 2) then q{i} = 1;
     if original{i} = 3 then q{i} = 2;
     if original{i} in (4 5) then q{i} = 3;
  end;
run;
```

ARRAY: Original					
Array Reference	**original[1]**	**original[2]**	**original[3]**	**original[4]**	**original[5]**
Element					
VAR	Vol01_13	Vol01_14	Vol01_15	Vol01_16	Vol01_17
ID 001	2	4	3	1	5

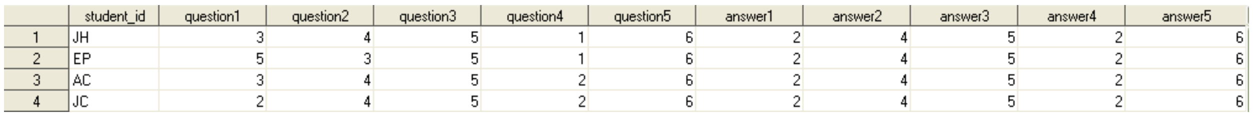
ARRAY: Q					
Array Reference	**q[1]**	**q[2]**	**q[3]**	**q[4]**	**q[5]**
Element					
New Vars	Q1	Q2	Q3	Q4	Q5
ID 001	1	3	2	1	3

COMPARE VALUES ACROSS MULTIPLE ARRAYS

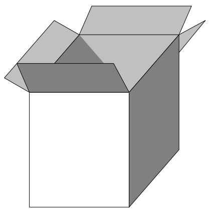
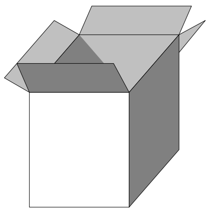
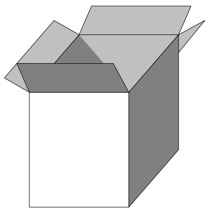
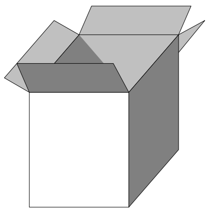
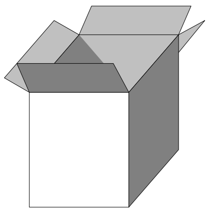
This example demonstrates the use of several of the techniques illustrated above. In addition, we added some PROC SQL and PROC FORMAT. You might want to compare the value of a variable to another variable and then, based on the outcome of that comparison, set the value of a third variable as a flag. You could use this technique to grade papers or to compare longitudinal data values. This example shows how you can use multiple arrays, three in this case, to grade a file of student responses (dataset=responses) by comparing those answers to a key (dataset=answer_key). Based on that comparison you then count the number of errors and assign a final grade to each student. Here two data sets are combined where each has 50 variables (one for each question on the test). Another 50 variables (error flags) are created to flag errors based on the comparison. The total number of errors can then be subtracted from the number of questions. The number of correct answers is then formatted to a letter grade. The keep statements give you a data set that is ready for PROC PRINT and posting for the students.

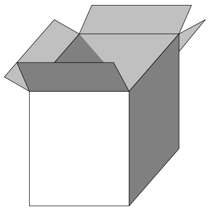
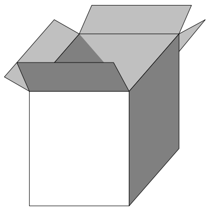
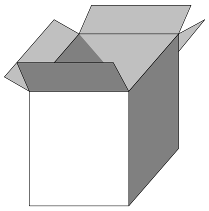
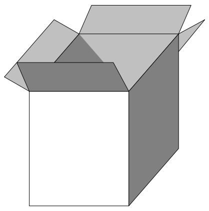
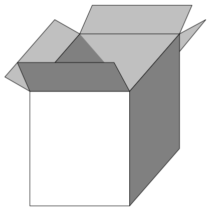
```
* format to create standard A,B,C,D,F scale;;
proc format;
    value af_scale
        46-50="A"
        41-45="B"
        36-40="C"
        31-35="D"
        0-30="F"
    ;
run;

PROC SQL;
create table exams as
select * from work.responses, work.answer_key;
quit;

data graded_exams;
set exams;
    * create 3 arrays;
    array response {50} question1-question50; * the response;
    array answer   {50} answer1-answer50; * the correct answer;
    array error    {50} error1-error50; * new variables used as error flags;
    * do loop to walk through all 50 questions;
    * compare each response to the answer;
    * if the response does not match the answer then flag as an error;
    do i=1 to dim(response);
        if response{i}^=answer{i} then error{i}=1;
        else error{i}=0;
    end;

    * calculate the number correct responses;
    * (subtract the number of errors from the total number of questions);
    n_correct = 50 - sum(of error1-error50);

    * format the number correct to a standard A,B,C,D,F scale;
    grade=put(n_correct,af_scale.);
    * keep just the identifier and the grade;
    keep student_id n_correct grade;
run;
```

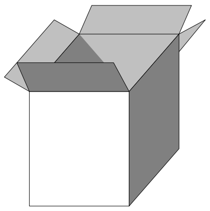
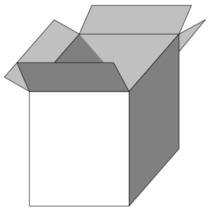
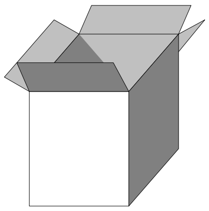
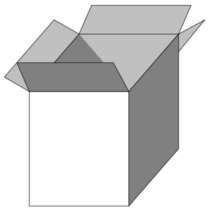
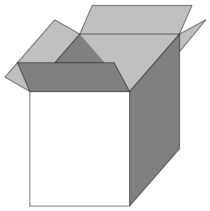
Table Graded exams after PROC SQL



	student_id	question1	question2	question3	question4	question5	answer1	answer2	answer3	answer4	answer5
1	JH	3	4	5	1	6	2	4	5	2	6
2	EP	5	3	5	1	6	2	4	5	2	6
3	AC	3	4	5	2	6	2	4	5	2	6
4	JC	2	4	5	2	6	2	4	5	2	6

ARRAY: RESPONSE					
Array Reference	**Response[1]**	**Response[2]**	**Response[3]**	**Response[4]**	**Response[5]**
Element					
ID JH	3	4	5	1	6

ARRAY: ANSWERS					
Array Reference	**Answers[1]**	**Answers[2]**	**Answers[3]**	**Answers[4]**	**Answers[5]**
Element					
ID JH	2	4	5	2	6

ARRAY: ERRORS					
Array Reference	**Errors[1]**	**Errors[2]**	**Errors[3]**	**Errors[4]**	**Errors[5]**
Element					
ID JH	1	0	0	1	0

RESTRUCTURE A DATASET

Arrays can also be used to restructure a dataset. This is common for graduate students to need to do to perform different types of analyses. The example laboratory dataset has three measurements for WBC for three different dates. We want to restructure this dataset to have one observation for each WBC measurement. To do this, use a simple array for the WBC value and one for the Date values. An output statement in the DO loop creates a new observation for every time the DO loop runs.

```
data work.restructured;
set work.wbc;
ARRAY WBC (3) WBC1-WBC3;
ARRAY Drawdate (3) Date1-Date3;
Do Timepoint= 1 to 3;
      Value=WBC[timepoint];
      Date=Drawdate[timepoint];
      Output;
End;
Keep Id Timepoint  Date;
```

work.WBC

OBS	ID	WBC1	WBC2	WBC3	Date1	Date2	Date3
1	001	6734	7890	5698	23JAN2010	09APR2010	15OCT2010
2	002	4234	6759	4386	15FEB2010	01MAR2010	15MAY2010
3	003	9876	10390	8994	09JUL2010	14AUG2010	01NOV2010

Work.restructured

OBS	ID	Timepoint	VALUE	Date
1	001	1	6734	23JAN2010
2	001	2	7890	09APR2010
3	001	3	5698	15OCT2010
4	002	1	4234	15FEB2010
5	002	2	6759	01MAR2010
6	002	3	4386	15MAY2010
7	003	1	9876	09JUL2010
8	003	2	10390	14AUG2010
9	003	3	8994	01NOV2010

You can also use arrays to restructure the data back to the original format. However, this code is more complicated and requires the use of multi-dimensional arrays. For more information on this, refer to Ron Cody's Paper on Transforming SAS data sets using arrays.[10]

5. CONCLUSIONS AND WHERE TO FIND MORE

This paper is the fourth in the 'Grad Student How-To' series[1,2,3] and gives graduate students a useful tool for writing more efficient code. This tool, which can save beginning programmers, such as graduate students, hours of effort and pages of code is a simple array. Arrays are especially high-impact when you have a large group of similar variables and you want to 'do the same thing' to all of the variables. This paper described simple array syntax, how arrays work, and gave a variety of examples from the authors' experiences.

There are many references available to students who are interested in learning more about arrays. First, start at the support.sas.com website to locate papers and documentation. The SAS Language reference for version 9.2 is online at http://support.sas.com/documentation/cdl_main/index.html and allows you to search for a keyword such as 'ARRAY' or '_TEMPORARY_. In addition, you can get to sample code and notes at http://support.sas.com/notes/index.html. There are many useful code samples in this repository. Finally, you can search http://support.sas.com/events/sasglobalforum/previous/online.html for many useful papers from previous SAS Global Users Group conferences. We have listed useful ARRAY papers in the reference section, including basic papers[7-9] and some more advanced and specialized applications of arrays.[10-13]

REFERENCES

1) Priest EL. Keep it Organized- SAS tips for a research project. South Central SAS Users Group 2006 Conference, Irving, TX, October 15-18, 2006.

2) Priest EL, Adams B, Fischbach L. Easier Exploratory Analysis for Epidemiology: A Grad Student How To Paper. SAS Global Forum 2009. Paper 241-2009.

3)Priest EL, Blankenship D. A Grad Student 'How-To' Paper on Grant Preparation: Preliminary Data, Power Analysis, and Presentation. SAS Global Forum 2010. Paper 274-2010.

4) SAS. ARRAY Statement. SAS® 9.2 Language Reference: Dictionary, Third Edition. http://support.sas.com/documentation/cdl/en/lrdict/63026/HTML/default/viewer.htm#a000201956.htm

5) SAS. DATA Step Processing. SAS® 9.2 Language Reference: Concepts,Second Edition. http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a001281588.htm

6)SAS. DO Statement. SAS® 9.2 Language Reference: Dictionary, Third Edition.

http://support.sas.com/documentation/cdl/en/lrdict/63026/HTML/default/viewer.htm#a000201951.htm

Basic Array papers

7)Waller, J. How to use ARRAYs and DO Loops: Do I DO OVER or Do I DO i? SAS Global Forum 2010. Paper 158-2010.

8)First, S. and Schudrowitz, T. Arrays Made Easy: An Introduction to Arrays and Array Processing. SUGI 30. Paper 242-30.

9)Keelan, S. Off and Running with Arrays in SAS. SUGI 27. Paper 66-27.

Intermediate/Advanced Application of arrays

10)Cody, R. Transforming SAS Data Sets Using Arrays. SUGI 25. Paper 1-25..

11)Scerbo, M. Everyone Needs a Raise (Arrays). SAS Global Forum 2007. Paper 216-2007.

12)MacDougall, M. Comparing Pairs: The Array's the Thing. SUGI 26. Paper 106-26.

13)Leslie, R.S. Using Arrays to Calculate Medication Utilization. SAS Global Forum 2007. Paper 043-2007.

SAS. Usage note 1780. http://support.sas.com/kb/1/780.html Implicit ARRAY support in version 7 and beyond.

ACKNOWLEDGMENTS

Thanks to the following individuals at Baylor Health Care System: Dr. Sunni Barnes for the dental survey example, Dr. Derek Blankenship for the microarray example, and Dr. Liz Burton for her laboratory data project. In addition, thanks to Dr. Tim Church at Pennington Biomedical Research Center for the medication recode example. All data used for these examples is simulated data.

CONTACT INFORMATION

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

Elisa L. Priest, MPH
elisapriest@hotmail.com