

Using SAS® ARRAYS: An Introduction to Easy Recodes

Francis J. Kelley, The University of Georgia, Athens, GA 30602-1911

The ARRAY statement used in SAS is often a source of confusion and frustration to new (and not-so-new) SAS programmers. Possibly this is caused, at least in part, by the programmers familiarity (or lack of familiarity) with the use of arrays in third generation languages. This paper will briefly review what a SAS ARRAY is (and is not), and provide descriptions of SAS ARRAY statements. Programming examples will begin with simple recoding statements and move on to more complicated variations.

I shall focus in this presentation upon what might be regarded as "introductory" aspects of the ARRAY statement used in SAS. In particular, my examples will concentrate upon recoding variables.

Often in my work, I am presented with large ungainly blocks of code that are "not working" sometimes this is the only explanation I get. In SAS programs, a typical source of such problem areas is large numbers of "IF ... THEN" statements, usually coded by new SAS programmers. A typical cause in the social sciences might be the need to "reverse a Likert scale." A Likert scale typically consists of five responses (including a neutral central point) that encompass all reasonable responses to a question or statement and is common in surveys; for example, from "strongly agree" to "strongly disagree."

A "Five-Point" Likert could be:

- 1 "strongly agree"
- 2 "somewhat agree"
- 3 "neither agree nor disagree"
- 4 "somewhat disagree"
- 5 "strongly disagree"

It is common on lengthy surveys to "reverse" the order of such responses. For example in the first five questions of a questionnaire, the first response offered might be "strongly agree" as above; in the next five the order might be reversed with "strongly disagree" being the first choice. When it comes time to perform the analysis, however, it is usually easier for the researchers if the data all follow the same order. A beginning SAS programmer might code something such as:

```
if q6 = 1 then q6 = 5 ;
else if q6 = 2 then q6 = 4 ;
...and so on...
```

This kind of coding is incredibly error-prone. It requires that each value of each question to be reversed be specified. A SAS ARRAY allows us to group all variables that are to be treated similarly in one area. A Fortran (or C, or almost any other third generation language) programmer would readily recognize this, but SAS arrays are not quite the same as arrays in those languages. They constitute "arrays within arrays," as a SAS library file is itself a two-dimensional array, being composed of "rows" (observations) and "columns" (variables). A SAS array may contain only the values available on a single

observation. To have access to all the observations in a library file, it is necessary to use SAS/IML®. Here is one way to use a SAS array to reverse the Likert scoring on three questions:

```
array likert {*} q6 q7 q8;
do i = 1 to 3 ;
  likert{i} = 6 - likert{i} ;
end ;
drop i ;
```

This little piece of code will reverse a five point likert value by subtracting the current value of whichever question we are working with (when $i = 1$, $\text{likert}\{i\}$ is $\text{likert}\{1\}$ which is Q6; when $i = 2$, it is Q7 and when $i = 3$, it is Q8) from 6. Missing values will remain missing; note that this does not address the question of out-of-range values, but that is an altogether different issue - we shall assume our data is "clean." It is usually a good idea to verify this using procedures such as SUMMARY or UNIVARIATE.

I had intended at this point to go through a series of refinements on the theme of using SAS arrays to aid in recoding variables; but a discussion on SAS-L (or comp.soft-sys.sas) provided an interesting view of how recoding might be accomplished using arrays.

The Problem:

Let us suppose we must perform an extensive series of recodes upon a file. This file may be in a SAS Library format, or it may be in some other form; for the purposes of this example, it will be simple raw data. The recodes could be almost anything, but let us say that a set of classification categories, perhaps hundreds of them, are to be changed. Let us say also that some categories will remain unchanged, while others will be merged together.

Having established the problem, let us now say that we have a file that describes both the original values and what they are to be after the recode. This is not an unreasonable condition, particularly if numerous codes are to be changed. Our recodes file will be in a rather simple form:

```
oldvalue = newvalue
```

This will be restricted to a single value on either side of the "="; thus, if categories 1 and 2 are to be recoded into a category 101, there will be an entry for "1" and another entry for "2" and we will not allow "1,2 = 101" (this could be handled with additional programming, but will muddy the example.) In addition, though our example will deal with numeric values, this will work equally well with character values.

A Solution:

Three small DATA steps are needed:

1. Read the RECODES file. This file specifies each oldvalue is to be transformed into one observation per recode value. One Macro variable must be created for the total number of values to be recoded.
2. Convert the RECODES file into a new file (RECODES2) containing all the old and new values, of a single observation.
3. Read the ORIGDATA file and perform the recode.

The RECODES file.

The RECODES file contains the recodes to be performed. These do not have to be one per line; the example reads data until none is left (note the "===" at the end of the INPUT statement). The INFILE option EOF= is used to allow a branch to be performed once the last of the data is read. The EOF branch (in this case to the statement labeled as LAST) takes place after the last of the data has been read, so the automatic variable _N_ is one more than the number of actual observations read (hence the need to DELETE the bogus "observation"). The value K is saved as the macro variable MAX. The TRIM and LEFT functions are used to assure that the character output of the PUT function is shifted to the left and has trailing blanks removed.

```
data recodes ;
  infile cards eof=last;
  input old equals $ new === ;
  return;
  last k = _N_ - 1 ;
  call symput ('max',trim(left(put(k,3)))) ;
  delete ;
  cards ;
  1 = 101 2 = 102 3 = 25 4 = 5 5 = 10
  6 = 100 8 = 200 9 = 10 10 = 100 11 = 100
```

The RECODES2 File.

The RECODES file is an N by M file, where N is the number of observations and M is the number of variables. This must be converted into a file that contains only one record with all the necessary variables. The macro variable MAX is used here, both in the ARRAY statements where it sets the dimension of the arrays, and in the RETAIN and KEEP statements where it is used to specify the variable names. In a SAS array, if no names are specified for the variables, they are assigned array_name1, array_name2, ... array_nameN - where N is the dimension of the array. The SET option END= was used to allow us to output RECODES2 only when the arrays OLDVAL and NEWVAL are complete. The KEEP statement assures that only the variables we want are written.

```
data recodes2 ;
  set recodes end=done;
  array oldval {&max} ;
  array newval {&max} ;
  retain oldval1-oldval&max newval1-newval&max;
  keep oldval1-oldval&max newval1-newval&max;
  oldval[_N_] = old ;
```

```
newval[_N_] = new ;
if done then output ;
```

The ORIGDATA file - and the REVDATA file.

In our examples, the data is in-stream; this is not how we would find it in actuality. Assume our original data is referenced by "infile ORIGDATA;". Here is what happens: the old and new values are explicitly RETAINED - again, we see how handy Macro variables can be. On the first iteration of the DATA step (_N_ = 1) the RECODES2 data set is read and its contents stored in the ARRAYs OLD and NEW. The old variable ORIGINAL is read from the ORIGDATA file. As some values are not to be changed by our recoding, we assign REVISED = ORIGINAL before making the comparisons. The final step requires that our program compares the value of ORIGINAL with each value on the OLD array. If a match is made, the variable REVISED is assigned the corresponding value from the NEW array. To limit the number of comparisons, a WHILE condition was coded on the iterative DO that loops through the array. When a match is made, the WHILE-condition will be met and the loop exited.

```
data revdata ;
  infile origdata ;
  retain oldval1-oldval&max newval1-newval&max ;
  if _N_ = 1 then set recodes2 ;
  array old {*} oldval1-oldval&max ;
  array new {*} newval1-newval&max ;
  input original ;
  revised = original ;
  out = 0 ;
  do i = 1 to &max while (out < 1) ;
    if original = oldval{i} then do ;
      revised = new{i} ;
      out = 1 ;
    end ;
  end ;
  keep original revised ;
run;
proc print data=revdata ;
  title 'Dataset: REVDATA';
run;
```

Using the following as the ORIGDATA file, see the Proc PRINT results are below:

origdata.dat

```
1
2
3
4
5
6
7
8
9
10
11
12
```

Dataset: REVDATA

OBS	ORIGINAL	REVISED
1	1	101
2	2	102
3	3	25
4	4	5
5	5	10
6	6	100
7	7	7
8	8	200
9	9	10
10	10	100
11	11	100
12	12	12

To conclude, I hope others will see that SAS arrays provide rather easy and straight forward solutions to many of what I tend to view as the tedious, dull and - inevitably - error-prone aspects of programming.

Reference:

SAS Institute Inc. (1990), *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS and SAS/IML software are registered trademarks or trademarks of SAS Institute Inc in the USA and other countries. ® Indicates USA registration.

Acknowledgement

All writers can benefit from the work of a good editor, and I am indebted to Jeannie McElhannon for all her efforts on this.

Author Contact Information

Francis Joseph Kelley
UCNS Client Services
Computer Services Annex
Athens, GA 30602-1911
jkelly@uga.cc.uga.edu
706-542-5359