# Using SAS® ARRAYs: An Introduction to Easy Recodes

Francis J. Kelley, The University of Georgia, Athens, GA
jkelley@arches.uga.edu

## Introduction.

The ARRAY statement used in the SAS system is often a source of confusion and frustration to new (and not-so-new) 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 programming languages. This paper will briefly review what a SAS ARRAY is (and is not) but the bulk is concentrated on a recoding example.

I shall focus this presentation upon what might be regarded as "introductory" aspects of the ARRAY statement used in SAS. In particular, my examples shall focus upon recoding variables. I should warn the beginner, however, that my large example makes use of a Macro variable.

Often in my work, I am presented with large ungainly blocks of code that are "not working" (this is sometimes the only real explanation I receive). In SAS programs, a typical source of such problems are large numbers of "IF … THEN" statements, usually coded by new SAS programmers. One source in the social sciences might be from 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 no disagree'
4 – "somewhat disagree'
5 – " strongly disagree"

A similar example could be made with "very likely" to "very unlikely." It is common in lengthy surveys to "reverse" the order of these accepted responses. Thus, the first five questions in a survey might be "strongly agree" as above, while the next five might have the order 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 all these "agrees" follow the same order. To do this, a beginning SAS programmer might code something like:

```
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 and specified. A SAS ARRAY allows us to group all variables that are to be treated similarly in one area. A Fortran (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 elements on a single observation. To have access to all the elements 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 (Q6, Q7, Q8):

```
array likert {*} q6 q7 q8 ;
do j = 1 to 3;
    likert{j} = 6 – likert{j};
end ;
```

This little piece of code will reverse the current value of whichever question we are working with (when j = 1, likert{j} = likert{1} = Q6; when j = 2 , it is Q7; and when j = 3 , it is Q8) by subtracting that variable from 6. Missing values will remain missing; note 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." The fact this is often an unwarranted assumption will be overlooked. It is usually a good idea to verify this assumption with procedures such as MEANS/SUMMARY or UNIVARIATE. I would also recommend replacing the "3" in the DO with "DIM(LIKERT)" to avoid (mis)counting the number of elements in the array.

A careful and somewhat tedious development of recoding methods using arrays might seem appropriate here, but discussions on SAS-L, as well as some work on related projects, 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 SAS Library format or it may be in some other form. For the purpose 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 be 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. It could even be in the form of an email document (saved to plain text). 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 become 101, there must be separate entries for "1" and "2" and we do not allow something like "1, 2 = 101" (this could be handled with additional programming but will muddy the example). This program can handle recodes from either numeric or character variables, but cannot change data types. That is, if CODE has a value of 1 (numeric) it cannot be given a value of "A6037" (character). Of course, it can change "1" (character) to "A6037." Additionally, this example will deal with a single variable only. As above, multiple variables can be handled with some additional programming but will muddy the example. Although all the variables may be listed, only those that are to be recoded must be listed.

## A Solution.

Three small DATA steps are needed:

1. Read the RECODES file. This is the file that specifies what each value is to be recoded to. A (very) modest error check will be performed. A count is made of the number of recodes to be made; this count is saved in a macro variable "MAX".
2. Convert the RECODES file into a new file(RECODES2) containing all the old and new values in a single observation.
3. Read the original data (ORIGDATA) 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 has been 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 actual number of recodes described (hence the need to subtract 1 from _N_ and to DELETE the bogus "observation"). The corrected number of observations (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 has been shifted to the left and had trailing blanks removed. Because this example converts character values, a LENGTH of 20 is specified for these to avoid the default of 8.

```
data recodes ;
  infile cards   eof=last ;
  length old new $ 20 ;
  input old $ equals $ new $ @@ ;
  if equals ne '=' then put _ALL_ ;
  drop equals ;
  return ;
  last: k = _N_ - 1 ;
    call symput
    ('max',trim(left(put(k,8.))) );
    delete ;
cards ;
A_64 = A_10_1064  A_65 = A_10_1085
       A_73 = A_12_1073
A_25 = B_5x_1025  A_26 = B_5x_1025
       A_10 = C_204_Z10
A_5 = A_100_1005 A_890 = C_10_2890
       B_24 = AB_Z1_1024
A_60 = A_12_1060  A_68 = A_10_2168
;
```

### The *RECODES2* File.

The RECODES file is an n×m file; n is the number of observations and m is the number of variables (in this case, m=2). This must be converted to a 1 × (n×m) file, one record containing all observations and variables. The macro variable MAX is used here in both ARRAY statements, where it sets the dimension of the arrays and in the RETAIN and KEEP statements, 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. Note the length of 20 specified for the (character) arrays OLDVAL and NEWVAL. This is where to remove the blanks from MAX (&max) is shown. The KEEP statement assures that only those variables we want are written.

```
data recodes2 ;
   set recodes   end=done ;
  array oldval {&max} $ 20 ;
  array newval {&max} $ 20 ;

  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* and *REVDATA* Files.

The example below shows the data file as being in-stream. This is not how we would find it in any real environment. However, for this example in-stream data (INFILE CARDS) is a file. Normally, we would want to refer to an ORIGDATA file.

As in RECODES2, the OLDVALs and NEWVALs are explicitly RETAINEd – again, we see how handy Macro variables can be. On the first iteration of the DATA step (_N_=1) the RECODES2 dataset is opened and its contents stored in the arrays OLD and NEW files. We will not open this dataset again; however, because of the RETAIN the values will be kept for the remainder of this step. The original code (ORIGINAL) is read from our data. As some values are not to be changed by this recoding, we assign REVISED = ORIGINAL before making the comparisons. The final step requires that our program compare the values of ORIGINAL with each value in 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 condition is set and the loop is exited.

```
data revdata;
  infile cards ;
  if _N_ = 1 then set recodes2 ;
  length  original  revised $ 20 ;
  retain oldval1-oldval&max
       newval1-newval&max ;
  array old {*} $ oldval1-oldval&max;
  array new {*} $ newval1-newval&max;
  input resp_id $ original $ var ;
  revised = original ;
  out = 0 ;
  do j = 1 to &max while (out < 1) ;
    if original = old{j} then do ;
      revised = new{j} ;
      out = 1 ;
    end ;
  end ;
  keep resp_id original revised var ;
cards ;
123  A_1      23
2390 A_25     15
291  A_68     20
2991 A_26     74
1090 B_20     68
887  B_24     46
271  A_64    344
501  A_5     401
8901 A_10     38
;
```

Here is the result of a Proc PRINT on the dataset REVDATA:

```
                The SAS System

OBS ORIGINAL REVISED   RESP_ID VAR
```

```
1   A_1     A_1         123     23
2   A_25    B_5x_1025   2390    15
3   A_68    A_10_2168   291     20
4   A_26    B_5x_1025   2991    74
5   B_20    B_20        1090    68
6   B_24    AB_Z1_1024  887     46
7   A_64    A_10_1064   271     344
8   A_5     A_100_1005  501     401
9   A_10    C_204_Z10   8901    38
```

## Conclusion.

Most SAS programmers will recognize this is not the only way a recode problem can be solved. There are alternatives: Proc FORMAT might be used to create special user-defined formats to handle change in the values displayed. SORT/MERGE might be used to replace the old values with the new ones. Proc SQL could be used in place of SORT/MERGE to accomplish the same thing. The example above is just that: an example. It illustrates some of the things you can do with an array and how you might use it. It also shows one way of avoiding having programs with lengthy, all but unmanageable blocks of error-prone code. The ARRAY statement can be a great help.

## Reference.

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



## Acknowledgement.

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

## Author Contact Information.

Francis Joseph Kelley
UCNS Client Services
Computer Services Annex
University of Georgia
Athens, GA 30602-1911
706.542.5359
jkelley@arches.uga.edu