# ITERATIVE PROCESSING OF A DATA SET USING TWO-DIMENSIONAL ARRAYS

Charles A. DePascale and Margaret R. Rost
Advanced Systems in Measurement and Evaluation

## INTRODUCTION/PROBLEM

In the last year we encountered two tasks which required multiple passes through the same data set to process the data. That is, within a single data step we wished to accomplish two objectives:

> Read each record two or more times
>
> Alter the value of variables within a record based on conditions found in subsequent records.

For example, the first case involved scored student responses to a test. The data file had to be modified so that there were no students with zero or perfect scores, and no items which 0% or 100% of the students answered correctly.

We were unable to find any procedures or statements which would allow us to repeatedly return to the first record in a data set and process the data again until our conditions had been met. Therefore, we decided to try to simulate this process. The solution presented in this paper is a technique which involves the use of two-dimensional arrays and multiple DO LOOPS.

## CREATING THE DATA ARRAY

The key to this technique is using a two-dimensional array to read the entire data file, or at least the relevant variables, into memory. The first dimension of the array represents the number of records in the data file and the second dimension represents the number of variables. A data file with 50 records and 5 variables would require the array statement: ARRAY SIMREC{50,5}.

As each record is read in the data step the values are assigned to this array and retained across records. All records except for the final record in the file are deleted. The result is instead of a data set which contains 50 records and 5 variables we are now working with a data set which contains 1 record and 250 variables. In essence, each set of 5 elements in the array contains the information from an original single record. A data set containing the variables S1-S5 would be processed in the following manner:

```
ARRAY S{5};
ARRAY SIMREC{50,5};
RECNO=_N_;
DO I=1 TO 5;
  SIMREC{RECNO,I}=S{I};
END;
RETAIN SIMREC1-SIMREC250;
IF RECNO<50 THEN DELETE;
```

## PROCESSING THE DATA

Now that the data are contained in an array, they can be 'read' and altered as many times as necessary within a single data step. We accomplished this through the use of multiple DO LOOPS.

An initial DO LOOP is used to read through the array once and tracking variables can be created to monitor key values (i.e. maximum scores, sums of 'variables' across 'records', etc.). After this loop is completed the values of the tracking variables are examined to determine whether the data must be modified. If the data must be modified a second DO LOOP is used to perform the modifications and initialize the tracking variables, if necessary. After the data have been modified, we use a GOTO statement to return to the first DO LOOP and begin the process again. This process will continue until the values of the tracking variables at the end of the first DO LOOP indicate that no further processing of the data are necessary. In short, the structure we used for handling these types of tasks was

```
BEGIN:    {FIRST DO LOOP}

          CHECK TRACKING VARIABLES,
          STOP? (Y/N)

          {SECOND DO LOOP}

          GOTO BEGIN
```

## CONCLUSIONS/LIMITATIONS

Using this technique we were able to successfully accomplish both of our tasks. There are, however, some serious limitations to its usefulness.

Most importantly, reading all of the data into a two-dimensional array puts quite a strain on computer memory. In processing our last task, which required storing twenty variables, we were only able to handle 60 records per unit of analysis. That is, any attempts to execute the program with more than 60 records resulted in an 'out of memory' message.

Secondly, reading all of the data into a two-dimensional array puts quite a strain on our memory. Programming for the tracking variables can become fairly complicated and it is hard to remember which array elements represent which original variables.

In short, the technique works and served us well, but we will happily welcome other solutions to the problem of making multiple passes through records within a single data step.