Paper 209-27

Comparing Macros to Arrays to Simple Code for Input Statements: A Visual Display for the Novice SAS® Programmer

Lara K. Jungvig, RAND, Santa Monica, CA

ABSTRACT

As a novice SAS programmer, I was thrown for a (%DO) loop by macros. I was not comfortable using macros until the day I was faced with an option: create and read in over 200 new variables by typing out all of their variable names or write several lines of code to do it for me. The decision was quick, and I have embraced macros ever since. In this paper, I present the code that I used for this task, side by side with a simple input statement and an array. The visual display demonstrates different techniques that may be used for the same task. I hope the novice programmer sees that different techniques, although intimidating at first sight, can be a timesaving tools that take some of the drudgery out of writing programs.

BACKGROUND

The code is a sample input statement used to read complex ASCII data in to SAS. The analysis is a medical literature review and meta-analysis. The primary data is abstracted from medical articles using a data collection instrument developed in-house. The data is then entered into an ASCII file and read into SAS. The code included in this presentation is used to input data on the details of treatment interventions given to patients in a study on the use dietary supplements. Data are collected on the following details of the intervention: intervention number, intervention component, dose and unit, route of administration, the prescribed frequency of dose and duration, and whether the drug was taken as needed. In this data collection instrument there are rows for up to a possible seven treatment interventions.

INTRODUCTION

This poster provides a visual illustration meant to encourage the novice programmer to incorporate arrays and macros into his/her programs. By presenting three different approaches to inputting code side by side, the reader can easily see the time saved that would have been spent typing out long lists of variable names. This poster does not explain the how's and why's of arrays and macro code, but simply demonstrates the "aha!" moment that I had that allowed me to be open-minded about using new and different approaches in my programs. Following the code in this text are tools for debugging macros that reader may find helpful.

SAS CODE: SIMPLE, ARRAY, MACRO

Simple input statement

Simple input statements are the obvious "go to" technique for the novice. The variable names are clearly laid out and the program is straightforward. This technique offers efficiency in run time that should be noted. But, there can be copious amounts of typing, so there is a burden in programming time, heightened by potential for programmer error, such as typos, bad syntax, etc. Another prevalent issue is the inflexibility of this technique. If the programmer wants to change variable names, he/she must do that in 7 different places. Examine the following code and making note of the length and repetition in variable names.

```
/**********************************
Using a simple input statement
to read in a flat ASCII file.
 **********************************/
DATA interventions;
INFILE 'C:\myfiles\rawdata\ints01.dat';

       INPUT /
              @1     id             5.
              @10    int_1          $5.
              @15    dose_1         10.
              @25    unit_1         3.
              @28    route_1        $3.
              @31    freq_1         $5.
              @36    frunit_1       $3.
              @39    dur_1          $5.
              @44    dunit_1        $3.
              @47    prn_1          1. @;

       INPUT /
              @1     id             5.
              @10    int_2          $5.
              @15    dose_2         10.
              @25    unit_2         3.
              @28    route_2        $3.
              @31    freq_2         $5.
              @36    frunit_2       $3.
              @39    dur_2          $5.
              @44    dunit_2        $3.
              @47    prn_2          1. @;

       INPUT /
              @1     id             5.
              @10    int_3          $5.
              @15    dose_3         10.
              @25    unit_3         3.
              @28    route_3        $3.
              @31    freq_3         $5.
              @36    frunit_3       $3.
              @39    dur_3          $5.
              @44    dunit_3        $3.
              @47    prn_3          1. @;

       INPUT /
              @1     id             5.
              @10    int_4          $5.
              @15    dose_4         10.
              @25    unit_4         3.
              @28    route_4        $3.
              @31    freq_4         $5.
              @36    frunit_4       $3.
              @39    dur_4          $5.
              @44    dunit_4        $3.
              @47    prn_4          1. @;

       INPUT /
              @1     id             5.
              @10    int_5          $5.
              @15    dose_5         10.
              @25    unit_5         3.
              @28    route_5        $3.
              @31    freq_5         $5.
              @36    frunit_5       $3.
              @39    dur_5          $5.
              @44    dunit_5        $3.
              @47    prn_5          1. @;
       INPUT /
              @1     id             5.
              @10    int_6          $5.
              @15    dose_6         10.
              @25    unit_6         3.
              @28    route_6        $3.
              @31    freq_6         $5.
              @36    frunit_6       $3.
              @39    dur_6          $5.
              @44    dunit_6        $3.
              @47    prn_6          1. @;
```

```
/*Simple input statement continued…*/
        INPUT /
                @1      id              5.
                @10     int_7           $5.
                @15     dose_7          10.
                @25     unit_7          3.
                @28     route_7         $3.
                @31     freq_7          $5.
                @36     frunit_7        $3.
                @39     dur_7           $5.
                @44     dunit_7         $3.
                @47     prn_7           1. ;

 RUN;
```

Using arrays to read in raw data

There are pluses and minuses that exist when using arrays. The ease of debugging and run time efficiency are the positives that pop up immediately. The task of creating arrays and do loops is particularly bothersome. Also, if your character variables are not initialized, they must be defined in the array, or an error will occur. Another issue to mention is that arrays act like simple code in that they do not have the flexibility of macros. For instance, in this case there is a potential for up to seven interventions on the data collection instrument. If the number of interventions is lower, say five, you have to scroll back through your code to find the do loop in order to change the upper limit. Examine the following code set up in an array with the previous discussion in mind:

```
/*****************************************
Using arrays to read in a flat ASCII file.
*****************************************/

DATA interventions;
INFILE C:\myfiles\rawdata\ints01.dat';

/*Making arrays*/
ARRAY a_ints {*}$       int_1 - int_7;

ARRAY a_dose {*}        dose_1 - dose_7;

ARRAY a_unit {*}        unit_1 - unit_7;

ARRAY a_route {*}$      route_1 - route_7;

ARRAY a_freq {*}$       freq_1 - freq_7;

ARRAY a_frunit {*}$     frunit_1 - frunit_7;

ARRAY a_dur {*}$        dur_1 - dur_7;

ARRAY a_dunit {*}$      dunit_1 - dunit_7;

ARRAY a_prn {*}         prn_1 - prn_7;

/*Do loop reads in interventions*/

DO i=1 to 7;
        INPUT /
                @1      id              5.
                @10     a_ints[i]       $5.
                @15     a_dose[i]       10.
                @25     a_unit[i]       3.
                @28     a_route[i]      $3.
                @31     a_freq[i]       $5.
                @36     a_frunit[i]     $3.
                @39     a_dur[i]        $5.
                @44     a_dunit[i]      $3.
                @47     a_prn[i]        1. @;

END;  /*ends do loop interventions*/
RUN;
```

Using macro code to read in raw data

Macro language saves programming time. Typing time is cut by volumes in some cases and the potential for making typos and programmer error is decreased. The flexibility that is built in to macros is demonstrated in the following code. If there are fewer interventions than the seven possible on the data collection form, a macro variable may be included at the top of the code and its value can easily be changed, cutting back on the number of completely missing variables. As a side note, macros may be used throughout your programs, and not just in the data step, adding to its flexibility. There is some run time inefficiency, however. Run time can be decreased by 5-10% when using macros. If you have huge datasets then this issue is a consideration in your choice of coding techniques. Examine the following code noting the use of macro variables for flexibility and length of code.

```
/*****************************************
Using macros to read in a flat ASCII file.
*****************************************/

DATA interventions;
INFILE 'C:\myfiles\rawdata\ints01.dat';

%MACRO readin;
%LET nints=5   /*macro variable*/

/*Do loop to read in interventions*/
     %DO ic=1 %TO &nints;
          INPUT /
               @1      id              5.
               @10     int_&ic         5.
               @15     dose_&ic        10.
               @25     unit_&ic        3.
               @28     route_&ic       $3.
               @31     freq_&ic        $5.
               @36     frunit_&ic      $3.
               @39     dur_&ic         5.
               @44     dunit_&ic       $3.
               @47     prn_&ic         1. @;

     %END;  /* ends do loop*/
%MEND;      /* ends macro 'readin' */
%readin     /* calls macro for execution */

RUN;
```

A word about debugging macros

As macro code can be difficult to debug, be aware that debugging can eat up the time saved in typing. However, SAS provides tools that can greatly reduce your debugging time. The MPRINT option prints statements generated by macro execution to the log file, so you can view resolved macros. You can direct MPRINT output to an external file by using the MFILE option and assigning the fileref MPRINT to the output file's fileref statement. These options print out the resolved macro and display error messages. MLOGIC is an option that traces the macro processor and writes to the log. For more information on macro debugging, refer to the special chapter on this topic in SAS Macro Language: Reference documentation called, "Macro Facility Error Messages and Debugging."

CONCLUSION

In summary, the novice programmer should not shy away from using advanced techniques. Once I understood that macros were programs that write programs, and tried this new approach in my input statement, I became less intimidated by macros. I was pleased not to have to write as much code and began to concentrate on debugging issues and to investigate issues of run time efficiency. However, I also learned that debugging macro code can be time consuming. The options MPRINT, MFILE, MLOGIC are invaluable debugging tools. Arrays provide a little

better run time efficiency. Creating arrays increases programming time and arrays are less flexible than macro code. Each technique presented here has advantages and disadvantages. I encourage the novice programmer to try different approaches and to incorporate arrays and macros into his/her programs.

REFERENCES

1. SAS® Institute Inc. (1999). "SAS OnlineDoc® Version 8," SAS® Institute Inc., Cary, NC.
2. Delwiche, Lora D. and Susan J. Slaughter (1996). "The Little SAS® Book," SAS® Institute, Inc., Cary, NC.
3. Carpenter, Art (1998). "Carpenter's Complete Guide to the SAS® System Macro Language," SAS® Institute Inc., Cary, NC.

ACKNOWLEDGEMENTS

I would like to thank Beth Roth, Mark Totten, Louis Ramirez and Ian Hilton for their help on this poster.

CONTACT INFORMATION

Lara K. Jungvig
Associate Programmer/Analyst
RAND Corporation
1700 Main Street
Santa Monica, CA 90407
Phone: (310) 393-0411
Fax: (310) 451-7059
E-mail: Jungvig@rand.org