Paper 65- 27

Don't Be a Slave to Your SAS® Programs

Marje Fecht
Larry Stewart, Senior Director, SAS, Cary, NC

ABSTRACT

Do you spend a lot of time updating your programs to adjust inclusion criteria? Do your programs run forever while you patiently wait? This beginning tutorial focuses on maintenance - free, efficient coding techniques so that you can spend your work time being more productive!

Topics include:
- macro coding techniques
- efficient programming tips
- code reduction tricks
- maintenance-free programming suggestions.

INTRODUCTION

Most programs are written on a tight schedule, using the most accessible knowledge of the programmer. While the tasks are accomplished, the program may not be as efficient as possible, and subsequent submissions may require tedious changes.

This presentation looks at programs that are often written in haste, and then suggests changes to improve the efficiency and maintenance of the programs.

The authors acknowledge that programmers need to strike a balance to:
- minimize your intervention with production jobs
- minimize runtime
- focus your programming time on reusable code (not one-time queries).

REMOVE INEFFICIENT AND "WORDY" CODING

Many SAS programs contain "wordy" coding that can be reduced as you learn more about the SAS language. The wordiness often results in inefficiency as data are processed multiple times, unnecessarily.

This presentation provides better solutions for coding examples that:
- use multiple steps when only one step is needed
- use multiple programs when only one is needed
- take forever to run.

EXAMPLE

Suppose your task is to subset a SAS data set and sort the resulting data set. A common solution to this task is to use a DATA step to subset the data followed by a PROC SORT step to sort the subset.

```
data compare;
   set report;
   if ProductCode in ('XER', 'REF', 'CRS')
      and Date gt '01APR2002'd;
   keep ProductCode Usage Date Location;
run;
proc sort data=compare;
   by ProductCode Date;
run;
```

An alternate solution is to subset the data as you sort it in the PROC SORT step. In this solution, you use a WHERE statement in the PROC SORT step to subset the data, and a KEEP= data set option to subset the columns. The OUT= option enables you to preserve the original table.

```
proc sort data=report
   (keep=ProductCode Usage Date Location)
         out=compare;
   where ProductCode in ('XER','REF','CRS')
         and Date gt '01APR2002'd ;
   by ProductCode Date;
run;
```

REDUCE PROGRAMMER INTERVENTION

When programs are written too specifically to the task, user-intervention is required on subsequent runs to:
- change dates and timeframes
- change titles and inclusion criteria.

Macro variables, SAS functions, and other coding tricks can be used to generalize a program making it more "data-driven". For example,
- macro variables enable you to define values for parameters used throughout a program
- functions provide system date and time information for use in labeling, subsetting, etc.
- Macros provide decision tools that can determine which programming segments to execute based on specified criteria.

EXAMPLE

Suppose your task is to run a daily report that compares month-to-date revenue for the current month to revenue for the previous month. In the solution below, the values for the current month and year, and the values for month and year for the previous month are hard coded in the program. Also, three DATA steps are used to create the desired data set for the report. The first DATA step selects the data for the current month, the second DATA step selects the data for the previous month, and the third DATA step concatenates the two data sets created by the first two DATA steps.

```
data currentmonth;
   set sugi.sales;
   if month(Date) = 4 and year(Date) = 2002;
   MonthYear = " 4/2002";
run;
data lastmonth;
   set sugi.sales;
   if month(Date) = 3 and year(Date) = 2002;
   MonthYear = " 3/2002";
run;
data comparemonths;
    set lastmonth currentmonth;
run;
proc means data=comparemonths;
   class MonthYear;
   var Revenue;
   title "MTD Revenue vs Last Month";
run;
```

The solution above requires you to change the code each month and is inefficient because three DATA steps are used when only one DATA step is needed. An alternate solution is to use DATA step functions to eliminate the need to alter the code each month,

and use the OUTPUT statement to build the final data set in one DATA step.

```
data comparemonths;
   set sugi.sales;
   if month(date)=month(today()) and
      year(date)=year(today()) then do;
         MonthYear=put(today() , mmyyS7.);
         output;
   end;
   else do;
      lastmonth=intnx('MONTH',today(), -1);
      if month(date)=month(lastmonth) and
         year(date)=year(lastmonth) then do;
            MonthYear=put(lastmonth,mmyyS7.);
           output;
       end;
   end;
run;
proc means data=comparemonths;
   class MonthYear;
   var Revenue;
   title "MTD Revenue vs Last Month";
run;
```

TESTING TIPS

When you improve a program using some of the techniques suggested above, it is prudent to compare the results to insure they are correct. Scanning output for similarity is often the approach taken, but it is easy to overlook discrepancies. Testing the equality of tables from the two solutions is a more accurate approach.

PROC COMPARE enables comparison of entire tables, columns, or subsets of data. While many options are available, a very simple comparison of two tables is accomplished using :

```
proc compare data=sales compare=sales2;
run;
```

CONCLUSION

As you learn more about SAS, you will find that there are numerous features that enable you to accomplish tasks easily and efficiently. The simple techniques provided in this presentation should help you identify and correct inefficiencies and wordiness in your coding. As shown in the first two examples above, a common inefficient practice by many beginning SAS users is to process data more times than is necessary. Additional examples showing code reduction techniques are provided in the paper presentation. For jobs that require manual intervention, as in the second example above, our suggestions may reduce or remove the intervention to enable “maintenance-free” jobs.

ACKNOWLEDGMENTS

The authors appreciate the feedback and suggestions provided by Robert Tremblay.

CONTACT INFORMATION

A detailed presentation handout is available at the scheduled presentation, or by contacting the authors.

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

Marje Fecht
Email: fechtmarje@aol.com

Larry Stewart
Email: larry.stewart@sas.com