Paper 220-25

Making the Transition from College to Industry

Brian Varney, Trilogy Consulting, Kalamazoo, Michigan

ABSTRACT

Most people who end up as SAS® programmers do so coming from a statistics or computer science program. Many times neither does a good enough job at teaching the fundamentals of SAS programming. Computer science programs typically teach lower level languages such as C++ and statistics programs focus more on statistical theory and analysis. The introduction of SAS programming gets left up to industry. Over the past few years more and more colleges have been implementing SAS programming courses which is helping tremendously.

INTRODUCTION

This paper will try to outline areas to which people lack exposure when starting as an entry level SAS programmer.

A BIG PICTURE UNDERSTANDING OF SAS

SAS has grown to be much more than just a statistical package. Knowing how SAS compares to other software packages and what components make up SAS can help one become a better programmer or consultant. Understanding the different functions of SAS and the competitors in those areas can be beneficial.

AN EXPLANATION OF HOW A SAS PROGRAM COMPILES AND EXECUTES

This is a topic that really helps one understand how SAS is different from lower level languages. Realizing that SAS is an interpreted language compiled and executed in small chunks is a fundamental concept in SAS. Those chunks consist of global statements, data steps, or procedures. Explaining how SAS programs work with and without macro is especially helpful. There are some good diagrams in the SAS Macro Language Reference manual.

Consider the following code.

```
data _null_;
   today=today();
   call symput('tdate',
               put(today,mmddyy10.));

title1 "Report For &tdate.";
```

Without the run statement ending the data step, the value gets loaded into the macro variable tdate too late to be used in the title statement.

DATA STEP CONCEPTS

Understanding how a data step works is key to SAS. Knowing what happens at compile time versus execution time helps us to write efficient and correct data steps.

At compile time, the data step is proofread for syntax errors and the program data vector is prepared. It is during compile time that SAS finds out the variables and variable attributes it is going to have to work with before any data values are read.

For instance, consider the problem in the following data step.

```
data one;
   set two(keep=var1 var2);
   if var1 < 5 then
      newvar='HIGH';
   else
      newvar='HIGHER';
run;
```

Understanding that the length of newvar is set at compile time according to the first reference to the variable in the data step code is important. Many new programmers run into the problem of character variable truncation. In this case, the variable newvar would always have the value 'HIGH' and the error may never be caught until it is too late.

Another problem for beginning programmers is trying to maintain others code which appears to be complicated. Consider the following data step.

```
data _null_;
   if 1=2 then
      set two nobs=nobs;
      call symput('numobs',
                  trim(left(put(nobs,12.))));
   stop;
run;
```

The previous data step code has many issues requiring a solid understanding of compile time versus execution time to fully comprehend. If you have never seen this chunk of code before you might be wondering "What the $%#@$ is this person trying to do"? I have run across code similar to this in many different companies so let's discuss the concepts.

The purpose of the data step is figure out the number of observations in the SAS data set two. Then load that value into a macro variable called NUMOBS.

This data step is taking advantage of the fact that the header of the data set is examined at compile time. The number of observations is stored in an internal variable called nobs. We do not have to execute the set statement because we only want to see how many observations are on the data step not read the data itself.

The set statement is seen at compile time but never executed because 1=2 is always false.

Even though there are methods to do this now that are easier to read, you never know when you need to maintain programs using this method.

USING CHARACTER VARIABLE FUNCTIONS

During most courses, the numeric variable functions are typically the only ones addressed. In industry, however, we frequently need to work with character strings. Consider the following character variables and sample values:

First_Name="Brian";
Last_Name="Varney";

A common operation is combining the variables Last_Name and First_Name into one variable containing both pieces of information such as the following.

LF_Name=”Varney, Brian”;

This is accomplished by the following line of data step code.

LF_Name=trim(left(Last_Name))||’, ‘||left(First_Name);

The above used the character variable functions trim and left. Some other commonly used character variable functions are:

- index
- scan
- substr
- put
- reverse
- length
- tranwrd
- compress
- upcase
- lowcase

There are many more character variable functions which can make your life much easier if you are aware of them. A complete list of these can be found in the online help within SAS.

FLOATING POINT ERROR EXPOSURE

It is usually quite a surprise when one first encounters floating point error. It is very confusing for a new programmer when the formatted value in the data appears as a 110 but is really 109.9999999999999. Consider the following data step code.

```
data test;
   do I=1 to 100;
      x + 1.1;
   end;
   if x=110 then
      put ‘true’;
   else
      put ‘false’;
run;
```

This looks innocent enough but floating point error is going to interfere here. Even though 1.1 added to itself 100 times is equal to 110, the floating point error causes the actual value of x to end up being 109.99999999999. It is helpful to know to use the round function in this case. See the following example.

```
if round(x,.001)=110 then …
```

The fuzz function can be used in this situation as well.

```
if fuzz(x)=110 then …
```

It is also useful to discuss base 2 versus base 10 arithmetic as well as the dangers of shortening the storage length of numeric variables. It is best to leave numeric variables to their default length of 8 bytes to avoid value truncation.

UNDERSTANDING DATE, TIME, AND DATETIME VARIABLES

There are only two types of variables in SAS, character and numeric. Dates, times, and datetimes are special cases of numeric variables.

A SAS Date is the number of days since January 1, 1960
For example: a value of 1=January 2, 1960

A SAS Time is the number of seconds since midnight
For example: a value of 1=00:00:01

A SAS Datetime is the number of seconds since midnight of January 1, 1960.
For example: a value of 1=January 1, 1960 00:00:01

In many industries, working with dates is a necessity. For instance in clinical trials, dates are sometimes stored as character because they can be incomplete. A strong knowledge of the functions and formats is a necessity in these situations. An example follows.

A patient in a clinical trial is asked when he had a headache last. He remembers that back in January of 1999 he had severe headaches but he can not remember what day it started and stopped. The data may be stored as 01/00/1999. When the data is analyzed a rule may be applied which says to change the missing day to the first day of the month, the middle of the month, or the last day of the month. Data manipulation of this nature requires a solid understanding of character variable functions as well as the date functions and formats.

HOW SAS HANDLES MISSING VALUES

There can be differences in how missing values are treated in expressions, functions, and procedures. For example, consider the following differences in the calculations of weeksum.

```
data calchrs;
   mon=8;
   tue=10;
   wed=9;
   thu=.;
   fri=9;
   sat=4;
   sun=0;
   weekadd=mon+tue+wed+thu+fri+sat+sun;
   weeksum=sum(mon,tue,wed,thu,fri,sat,sun);
run;
```

The variable weekadd will result in a missing value. The variable weeksum will result in 40 because missing values are ignored by the functions.

A SAS procedure typically ignores missing values unless there is an option which tells it not to ignore them.

CREATING CUSTOM OUTPUT REPORTS

In classroom exercises, custom output reports generally are not a concern. In industry, custom output reports are the typical endpoint that one is trying to end up with. The following SAS tools are good to review and experiment with while in college.

- Non-Graphical
  PROC Tabulate
  PROC Print
  PROC Report
  Data Step w/ put statements

- Graphical
  PROC GPLOT
  PROC GCHART

BASIC UNDERSTANDING OF RELATIONAL DATABASES

In college classes, the data for a study is typically contained in one file with all of the necessary variables present. Most companies store their data in a relational database requiring programmers to go to multiple files to get the information that is necessary. One function that is necessary to perform is the merge or join. Almost any SAS programming position is going to require the merging of SAS data sets. To perform a merge successfully there are a few things that a programmer needs to identify and remember.

The first thing is that whenever information is stored in a file there are things called key variables. These variables allow us to be able to drill down to one piece of information in the file.

For example, let's say a medical insurance company keeps information on its members. They would have a file holding demographic information requiring one record per claimant. The key variables for this file may be the social security number of the employee and the social security number of the dependent. Most likely a file of this nature would contain many other variables such as first and last name of employee, first and last name of dependent(s), employee's company which the insurance is through, gender, birth date as well as many others.

Claimant Demographic Table Example

Employee SSN	Status	Dependent SSN	Date of First Coverage
123-45-6789	Employee	123-45-6789	01/01/1999
123-45-6789	Spouse	898-12-2345	02/03/1998
123-45-6789	Child	788-55-5555	02/23/1989
456-78-9111	Employee	456-78-9111	03/12/1978
567-89-9123	Employee	567-89-9123	04/04/1998

Let's say there is a second table for prescription activity as shown below. How would these tables fit together?

Claimant Prescription Activity Table Example

Claimant SSN	Date of Claim	Claim Code	Medication Name
123-45-6789	01/31/1999	4RF567HU8	Amoxicillan
123-45-6789	02/09/1999	3ERT6Y7UI	Cephalasporin
788-55-5555	03/01/1999	99EFR5432	Procepia
456-78-9111	04/01/1979	4RT56YUU	Valium
456-78-9111	04/07/1979	3E4R5T6Y7	Tylenol VIII

Understanding the data and its associated rules are very important before any kind of analysis or reporting can be done. For instance, what would happen if a person switched insurance companies and then switched back a few years later. How would that situation be represented in this file?

This company would also have many other database tables containing information on medical claims, dental claims, medical providers, etc for the employees and their dependents. Understanding how all of the files relate to each other is very important.

Once this is understood, programmers would need to know how to use data step merges and/or SAS SQL joins.

EXPOSURE TO GOOD PROGRAMMING STANDARDS

Most companies have programming standards. Getting people in the habit of following good programming standards will save them and the people who maintain their SAS code much pain in the long run.

Below are a few things that are typically included in SAS programming standards. Please note that this list is not exhaustive.

- program header standards
- comment block standards
- statement indentation rules
- case sensitivity of code rules
- if then else indentation rules
- naming standards for programs, variables, and data sets
- macro usage rules
- program version control rules
- program validation checklist/rules

WHERE TO GO FOR SAS HELP

Knowing where to go for information when working with SAS can save a lot of time. An understanding of how to use the following sources of SAS help can help someone be more resourceful later on.

- SAS Online Help
- SAS Manuals
- SAS Phone Tech Support
- SAS Web Tech Support

HOW TO VALIDATE A PROGRAM

How to validate a program may seem like common sense but is something that is commonly rushed and overlooked by beginning SAS programmers. Making a checklist for the cosmetic and content parts of the output from a SAS program is key in getting accurate results that look as specified.

A few checklist items are included below as an example of the cosmetic validation.

- Titles
- Footnotes
- Variable labels
- Decimal precision

Typically to check the content parts of SAS output a separate program is written to make a "not so pretty" listing containing the summarized data so numbers can be cross checked.

CONCLUSION

This paper attempts to summarize concepts that typically need to be addressed when someone enters industry from a college setting. Excellent guides for learning some of these concepts are the following manuals.

- SAS Language and Procedures Usage
- Getting Started with the SQL Procedure
- SAS Macro Language Reference

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author may be contacted at:

Brian Varney
Trilogy Consulting
5148 Lovers Lane
Kalamazoo, MI 49002
Work Phone: 616-344-4100 x110
Fax: 616-344-1887
Email: BKVarney@TrilogyUSA.com