Writing SAS® Programs that Work

S. J. Slaughter, Independent Consultant, Davis, CA
Lora L. Delwiche, IT/ANSA, University of California, Davis, CA

ABSTRACT

There are some things you can do to make it more likely that your programs will work and make it easier to figure out what's wrong when they don't work. Make your programs easy to read, test each part of your program before going on to the next, and test your program with simplified or smaller data sets when appropriate.

If your program doesn't work, there are a few steps you can follow that will make it easier to find the source of the problem. Read the SAS log. The SAS log has many messages in the form of notes, warnings, and errors all of which are important in determining the source of the problem. Always start at the beginning of the log and look for simple mistakes first.

INTRODUCTION

Even experienced SAS programmers will tell you it's a surprise when their programs run on the first try—like a golfer getting a hole-in-one. The longer and more complicated the program, the more likely it is to have errors. But don't despair, there are a few guidelines you can follow that can make your programs run correctly sooner, and help you discover errors more easily.

Make programs easy to read

One simple thing you can do, is develop the habit of writing programs in a neat and consistent manner. Programs that are easy to read are easier to debug, and will save you time in the long run. Here, for example, is a program that runs without problems, but is hard to read.

```
DATA demog; INFILE 'demo.dat';
INPUT survnum $ name $ age sex;
location = SUBSTR(survnum,1,1);
PROC SORT;  BY survnum;
DATA ratings; INFILE 'rate.dat';
INPUT survnum $ (song1-song10) (1.);
average = MEAN(OF song1-song10);
PROC SORT; BY survnum;
DATA both; MERGE demog ratings;
BY survnum; RUN;
```

This program reads the two data files in Table 1, sorts them, and merges them. That's not very complicated, but trying to make sense out of the disorganized program above is definitely complicated.

Follow these guidelines for programs that will still make sense six months after you write them:

- Put only one SAS statement on a line. SAS allows you to put as many statements on a line as you wish. This may save you some space in your program, but the saved space is rarely worth the sacrifice in readability.

- Use indentation to show the different parts of the program. Indent all statements except the DATA and PROC statements. This way you can tell at a glance how many DATA and PROC steps there are in a program and which statements belong to which steps. It's also helpful to further indent any statements between a DO statement and its END statement.

- Use comment statements generously to document your programs. This takes some discipline, but is important especially if anyone else is likely to read or use your program. Everyone has a different programming style, and it is often impossible to figure out what someone else's program is doing and why. Comment statements take the mystery out of the program.

Here is the same program with a little organization. Compare this program to the previous one, and consider which program would you prefer to maintain.

```
* Read demographic data;
DATA demog;
   INFILE 'demo.dat';
   INPUT survnum $ name $ age sex;
   location = SUBSTR(survnum,1,1);
* Sort demo data;
PROC SORT;
   BY survnum;

* Read song ratings data;
DATA ratings;
   INFILE 'rate.dat';
   INPUT survnum $ (song1-song10) (1.);
   average = MEAN(OF song1-song10);
* Sort ratings;
PROC SORT;
   BY survnum;

* Combine demo and ratings data;
DATA both;
   MERGE demog ratings;
   BY survnum;
* Print the combined data;
RUN;
```

The second program is longer, but you can tell at a glance that it reads two raw data files, sorts them, and merges them together.

Table 1 Data from the two raw data files containing demographic and ratings data.

DEMO.DAT	RATE.DAT
104 John 45 2	107 2314321423
105 Rich 13 2	108 4354324534
109 Melissa 19 1	206 4343554544
201 Donna 36 1	207 4534543544
101 Joann 13 1	101 5453254154
208 Kevin 32 2	105 4554354354
209 Ashley 25 1	106 4354325435
210 Daniel 26 2	208 5433542345
211 Zack 13 2	209 4344344425
102 Kirk 25 2	102 4325342354
103 Halley 15 1	103 5234131444
202 Jack 41 2	109 5435454254
203 Michael 38 2	201 4532453424
204 Dana 33 1	202 3424352433
106 Amanda 27 1	203 3342553432
107 John 54 2	204 4354543545
108 Judy 31 1	205 5435454535
205 Jane 16 1	104 4543543243
206 Michelle 17 1	210 3245343255
207 Harry 16 2	211 4534254543

Test each part of the program

You can increase your programming efficiency tremendously by making sure each part of the program is working before moving on to write the next part. If you were building a house, you would make sure the foundation was level and square before putting up the walls. You would test the plumbing before finishing the bathroom. You are, by law, required to have each stage of the house inspected before moving on to the next. The same should be done for your SAS programs. But you don't have to wait for the inspector to come out, you can do it yourself.

If you are reading a raw data file, use PROC PRINT to print the SAS data set at least once to make sure it is correct before moving on. Sometimes, even though there are no errors or even notes in your SAS log, the SAS data set is not correct. This could happen because SAS did not read the data the way you imagined (after all it only does what you say not what you are thinking), or because the data has some peculiarities you had not realized. For example, a researcher who received two data files from Taiwan wanted to merge them together by date. She could not figure out why they refused to merge correctly until she printed both data sets and realized one of the files used Taiwanese dates which are offset by 11 years.

It's a good habit to print all the SAS data sets you create in a program at least once to make sure they are correct. As with reading raw data files, sometimes merging, and setting data sets can produce the wrong results even though there were no error messages. So when in doubt, use PROC PRINT.

The program above creates three data sets and so would need three PROC PRINTs.

```
* Read demographic data;
DATA demog;
   INFILE 'demo.dat';
   INPUT survnum $ name $ age sex;
   location = SUBSTR(survnum,1,1);
* Print demo data;
PROC PRINT DATA = demog;
   TITLE1 'Music Survey';
   TITLE2 'Demographic Data';
* Sort demo data;
PROC SORT;
   BY survnum;

* Read song ratings data;
DATA ratings;
   INFILE 'rate.dat';
   INPUT survnum $5. (song1-song10) (1.);
   average = MEAN(OF song1-song10);
* Print ratings;
PROC PRINT DATA = ratings;
   TITLE2 'Ratings for Songs';
* Sort ratings;
PROC SORT;
   BY survnum;

* Combine demo and ratings data;
DATA both;
   MERGE demog ratings;
   BY survnum;
* Print the combined data;
PROC PRINT DATA = both;
   TITLE2 'Combined Data';
RUN;
```

When this program runs, it produces a listing of each data set. An abridged version of that output appears below.

```
                          Music Survey                              1
                        Demographic Data

OBS    SURVNUM    NAME        AGE    SEX    LOCATION

 1       104       John         45     2        1
 2       105       Rich         13     2        1
 3       109       Melissa      19     1        1
 4       201       Donna        36     1        2
 5       101       Joann        13     1        1

                          Music Survey                              2
                        Ratings for Songs

     S                                                        A
     U                                                  S     V
     R        S  S  S  S  S  S  S  S  S  O     E
     V        O  O  O  O  O  O  O  O  O  N     R
O    N        N  N  N  N  N  N  N  N  N  G     A
B    U        G  G  G  G  G  G  G  G  G  1     G
S    M        1  2  3  4  5  6  7  8  9  0     E

1   107 2     3  1  4  3  2  1  4  2  3  1   2.4
2   206 4     3  4  3  5  5  4  5  4  4  2   3.9
3   101 5     4  5  3  2  5  4  1  5  4  1   3.4
4   106 4     3  5  4  3  2  5  4  3  5  2   3.6
5   209 4     3  4  4  3  4  4  4  2  5  1   3.4
```

```
                        Music Survey                              3
                       Combined Data

                                L
     S                          O                                 A
     U                          C                              S  V
     R                          A S S S S S S S S S O  E
     V     N                    T O O O O O O O O O N  R
O    N     A          A S I N N N N N N N N N G  A
B    U     M          G E O G G G G G G G G G 1  G
S    M     E          E X N 1 2 3 4 5 6 7 8 9 0  E

1   101    Joann   13 1 1 . . . . . . . . . .  .
2   101 5          .  .   4 5 3 2 5 4 1 5 4 1  3.4
3   102    Kirk    25 2 1 . . . . . . . . . .  .
4   103    Halley  15 1 1 . . . . . . . . . .  .
5   103 5          .  .   2 3 4 1 3 1 4 4 4 1  2.7
6   104    John    45 2 1 . . . . . . . . . .  .
7   105    Rich    13 2 1 . . . . . . . . . .  .
8   106    Amanda  27 1 1 . . . . . . . . . .  .
9   106 4          .  .   3 5 4 3 2 5 4 3 5 2  3.6
10  107    John    54 2 1 . . . . . . . . . .  .
```

Although this program runs without producing any error messages, the output shows a serious problem. The variable SURVNUM is wrong in the RATINGS data set. The format of $5. for SURVNUM in the INPUT statement of the second DATA step causes SAS to read part of the next variable. Because the values of SURVNUM are wrong, observations from the two data sets cannot be matched. Adding the PROC PRINTs clearly shows where the program went wrong.

Test programs with small data sets

Sometimes it's not practical to test your program with your entire data set. If your data files are very large, you may not want to print all the data, or it may take a long time for your program to run. In these cases, you can test your program with a subset of your data.

If you are reading data from a raw data file, you can use the OBS option on the INFILE statement to tell SAS to stop reading when it gets to that line in the file. This way you can read only the first 50 or 100 lines of data, or however many it takes to get a good representation of your data. The following statement will read only the first 100 lines of the data file named DEMO.DAT.

```
INFILE 'demo.dat' OBS = 100;
```

You can also use the FIRSTOBS option to start reading from the middle of the raw data file. So, if the first 100 data lines are not a good representation of your data, but 100 through 200 are, you can use the following statement to read just those lines.

```
INFILE 'demo.dat' FIRSTOBS = 100 OBS = 200;
```

FIRSTOBS and OBS relate to the lines of data in the file, not necessarily observations. If, for example, you are reading two lines of data for each observation then you would need to read 200 lines of data (OBS = 200) to get 100 observations.

If you are reading a SAS data set you can use the OBS and FIRSTOBS data set options in a SET, MERGE, UPDATE, or PROC statement.

```
DATA demog;
   SET demog (FIRSTOBS = 100 OBS = 200);
```

or

```
PROC PRINT DATA = demog (FIRSTOBS = 100
   OBS = 200);
```

Using OBS and FIRSTOBS is an easy way to test your programs, but sometimes it is difficult to get a good representation of your data this way. You may need to create a small test data set by extracting representative parts of the larger data set. Or you may want to make up representative data for testing purposes. Making up data has the advantage that you can simplify the data and make sure you have every possible combination of values to test.

Sometimes you may want to make up data and write a small program just to test one aspect of your larger program. This can be extremely useful for narrowing down possible sources of error in a long or complicated program.

Fixing programs that don't work

In spite of your best efforts, sometimes programs just don't work. More often that not programs don't run the first time. Even with simple programs it is easy to forget a semicolon or misspell a keyword—everyone does it sometime. If your program doesn't work, the source of the problem may be obvious like an error message with the offending part of your program underlined, or not so obvious as when you have no errors but still don't have the expected results. Whatever the problem, here are a few guidelines you can follow to help fix your program.

Read the SAS log

The log contains the program statements you submitted plus messages from SAS about the execution of your program; while the output contains printed results, if any, from SAS procedures you ran. Every time you run a SAS job, you generate a log.

Many SAS programmers ignore the SAS log and go straight to the output. That's understandable, but dangerous. It is possible—and sooner or later it happens to all of us—to get bogus results that look fine in the output. The only way to know they are bad is to check the SAS log. Just because it runs, doesn't mean it's right.

People tend to think of the SAS log as either a rehash of their program or as just a lot of gibberish. OK, we admit, there is some technical trivia in the SAS log, but there is also a wealth of important information.

Here is part of the log generated by the program above:

```
NOTE: Copyright(c) 1989 by SAS Institute
Inc., Cary, NC USA.
NOTE: SAS (r) Proprietary Software Release
6.08 TS404
Licensed to ZXY Corp., Site 0036501

1     * Read demographic data;
2     run;
3     DATA demog;
4        INFILE 'demo.dat';
5        INPUT survnum $ name $ age sex;
6        location = SUBSTR(survnum,1,1);
7     * Print demo data;

NOTE: The infile 'demo.dat' is:
      FILENAME=demo.dat,
      RECFM=V,LRECL=132

NOTE: 20 records were read from the infile
      'demo.dat'.
      The minimum record length was 17.
      The maximum record length was 17.
NOTE: The data set WORK.DEMOG has 20
      observations and 5 variables.
NOTE: The DATA statement used 7.41 seconds.


8     PROC PRINT DATA = demog;
9        TITLE1 'Music Survey';
10       TITLE2 'Demographic Data';
11    * Sort demo data;

NOTE: The PROCEDURE PRINT used 0.6 seconds.


12    PROC SORT;
13       BY survnum;

NOTE: The data set WORK.DEMOG has 20
      observations and 5 variables.
NOTE: The PROCEDURE SORT used 1.76 seconds.
```

The SAS log above is a blow-by-blow account of how SAS ran the program. It starts with notes about the version of SAS, the type of computer, and the SAS site number. It contains the original program statements with line numbers added on the left. If there are any error messages, they will appear here indicating where SAS got confused and what action it took. You may also find warnings and notes which sometimes indicate errors and other times just provide useful information.

DATA steps are always followed by a note with the name of the SAS data set created (WORK.DEMOG in the log above), and the number of observations (20) and variables (5). Even though this type of note is normal, it is a good idea to glance at it to make sure that SAS did not "loose" any observations, or accidentally create a lot of unwanted variables. Here, for example, is part of a log indicating that SAS read 20 lines from a raw data file, but created only 10 observations.

```
NOTE: 20 records were read from the infile
      'rate.dat'.
      The minimum record length was 14.
      The maximum record length was 14.
NOTE: SAS went to a new line when INPUT
      statement reached past the end of a
      line.
NOTE: The data set WORK.RATINGS has 10
      observations and 13 variables.
```

If you have two lines of raw data for every observation then this is fine. If you have one line of raw data for each observation, then SAS ran into a problem reading your data.

The SAS log has three types of messages about your program: notes, warnings, and errors.

Errors are hard to ignore. Not only do they come up in red on your screen, but your program will not run with errors. Usually errors are some kind of syntax or spelling mistake. The following shows the error message when you accidentally add a slash between the PROC MEANS and DATA= keywords. SAS underlines the problem (the slash) and tells you it doesn't recognize the symbol.

```
1    PROC MEANS /DATA=one;
                 -
                 200
ERROR 200-322: The symbol is not recognized.
```

The location of the error is easy to find, because it is usually underlined, but the source of the error can sometimes be tricky. Sometimes what is wrong is not what is underlined, but something else earlier in the program.

Warnings are less serious than errors, because your program will run with warnings. But beware, a warning may mean that SAS has done something you have not intended. For example, SAS will attempt to correct your spelling of certain keywords. If you misspelled INPUT as IMPUT you will get the following message in your log:

```
WARNING 1-322: Assuming the symbol INPUT was
   misspelled as IMPUT.
```

Usually you would think, "SAS is so smart—it knows what I meant to say," but occasionally that may not be what you meant at all. Make sure that you know what all the warnings are about and that you agree with them.

Notes are less straightforward than either warnings or errors. Sometimes notes just give you information like telling you the execution time of each step in your program, but sometimes notes can indicate a problem. If, for example, you have the following note in your SAS log:

```
NOTE: SAS went to a new line when INPUT
   statement reached past the end of a line.
```

this could mean that SAS did exactly what you wanted, or it could indicate a problem with either your program or your data. Make sure that you know what each note means and why it is there.

Start at the beginning

Whenever you read the SAS log, start at the beginning. This may seem like a ridiculous statement—why wouldn't you start at the beginning? Well, if you are using SAS display manager, the SAS log rolls by in the window and when the program is finished, you are left looking at the end of the log. If you happen to see an error at the end of the log, it is natural to try to fix that error first—the first one you see. Avoid this temptation. Often errors at the end of the log are caused by earlier ones and if you fix the first error, often most or all of the other errors will disappear. If your lawnmower is out of gas and won't start, it's probably better to add gas before trying to figure out why it won't start. The same logic applies to debugging SAS programs, fixing one problem will often fix the others.

Here is a log that contains two errors, three warnings, and three notes indicating a problem.

```
1     * Read demographic data;
2     DATA demog
3        INFILE 'demo.dat';
                ------------
                200
4        INPUT survnum $ name $ age sex;
5        location = SUBSTR(survnum,1,1);
6     * Print demo data;

 ERROR 200-322: The symbol is not
   recognized.
ERROR: No CARDS or INFILE statement.
NOTE: The SAS System stopped processing
   this step because of errors.
WARNING: The data set WORK.DEMOG may be
   incomplete.  When this step was stopped
   there were 0 observations and 5 variables.
WARNING: The data set WORK.INFILE may be
   incomplete.  When this step was stopped
   there were 0 observations and 5 variables.
NOTE: The DATA statement used 6.54 seconds.

7     PROC PRINT DATA = demog;
8        TITLE1 'Music Survey';
9        TITLE2 'Demographic Data';
10    * Sort demo data;

NOTE: No observations in data set
   WORK.DEMOG.
NOTE: The PROCEDURE PRINT used 0.6 seconds.

11    PROC SORT DATA = demog;
12       BY survnum;
13    RUN;

WARNING: Input data set is empty.
NOTE: The data set WORK.DEMOG has 0
   observations and 5 variables.
NOTE: The PROCEDURE SORT used 1.59 seconds.
```

It looks like this program is in big trouble, but there is really just one problem, a missing semicolon at the end of the DATA statement. Once that semicolon is in place, all the messages about problems will disappear.

Look for common mistakes first

More often that not, there is a simple reason why your program doesn't work. Look for a simple reason before trying to find something more complicated.

Missing semicolons are the most common source of errors in SAS programs. This is unfortunate because, while it is easy to forget the semicolon, it is not always easy to find the missing semicolon. SAS reads statements from one semicolon to the next without regard to the layout of the program. When you leave off a semicolon, you, in effect, concatenate two SAS statements together. When this happens, you can get error messages that just don't make much sense. For example, you may get an error message saying the INPUT statement is not valid. This is odd because INPUT is a valid SAS statement. In such cases, look for a missing semicolon.

If you have errors, SAS will underline the location in your program where it encountered the error. This helps, but can be misleading. Sometimes, the actual mistake is not what is underlined, but something before that. Be sure to look at the underlined item and the previous few statements.

In our experience, a small number of mistakes comprise the majority of errors received by SAS programmers. Here is a list of the ten most common problems encountered in SAS programs and the major causes of each.

NOTE: INPUT statement reached past the end of the line

- Blank lines in your raw data file
- Using list style input without a space between every data value
- Some data lines shorter than the rest

NOTE: Lost card

- Missing or duplicate data lines

NOTE: Invalid data

- Forgetting to specify that a variable is character (SAS assumes it is numeric).
- Incorrect column specifications producing embedded spaces in numeric data.
- List style data with two periods in a row and no space in between.
- Failing to mark a missing value with a period in list style input, causing SAS to read the data for the next variable.
- Using the letter O instead of the number zero.
- Special characters such as carriage-return-line-feed and page-feed.
- Invalid dates (such as September 31) read with a date informat.

NOTE: Missing values were generated

- A variable cannot be computed because of existing missing values in your data

NOTE: Numeric values have been converted to character (and vice versa)

- Setting a variable equal to another variable of a different type.
- Using a variable with the wrong type of function.
- Using a character variable in an arithmetic expression.

ERROR: Invalid option, Error: The option is not recognized, or Error: Statement is not valid

- A missing semicolon
- A misspelled keyword
- A DATA step statement in a PROC step (or vice versa)
- The correct option with the wrong statement
- An unmatched quote

NOTE: Variable is uninitialized or Error: Variable not found

- A misspelled variable name
- Using a variable that has been dropped
- Using the wrong data set
- A logic error, such as using a variable before it is created

DATA step produces wrong results, but no error message

- A logic error, such as using a SET statement instead of a MERGE.

SAS truncates a character variable

- The length of the variable is too short

SAS stops in the middle of a job

- An unmatched quote
- An unmatched comment
- No RUN statement at end of program in interactive SAS
- Ran out of time in batch mode
- /* in the first column under MVS

CONCLUSIONS

You can make your programs run correctly sooner, and discover errors more easily, by following the guidelines in this paper. Make your programs easy to read, test each part of your program before going on to the next, and test your program with simplified or smaller data sets when appropriate. When you do get an error, read the SAS log, start at the beginning, and look for simple mistakes first.

For a more extensive discussion of debugging, you may want to look at *The Little SAS Book*, on which this article is based.

REFERENCES

Delwiche, L. L., and Slaughter, S.J. (in press), *The Little SAS Book*, Cary, NC: SAS Institute, Inc.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries.

The authors may be contacted at:

Susan J. Slaughter (916) 756-8434

Lora L. Delwiche (916) 752-6285
lldelwiche@ucdavis.edu