CREATING COURSE NOTES FROM AN ANNOTATED SAS PROGRAM

Karen G. Malley, ARC Professional Services Group, Inc.

ABSTRACT

A nonstandard approach to the preparation of course notes for an introductory-level SAS course is described. The use of a thoroughly-commented SAS log provides a mechanism for easy updating as new features of SAS are available, and appeals to students with diverse backgrounds since there can be much to look at and learn from in the SAS log.

INTRODUCTION

The main set of notes for an introductory-level SAS course[1] consists of the output of a long batch SAS job, in which strategically placed comments help guide the students through the material. To improve the readability of the notes, the following guidelines were followed, which could be applied to any material in the SAS system that can be put into a batch job.

On the first day of this class, a much smaller batch job output is also distributed, for students to compare the 8.5x11 format of their notes with the usual 132-column computer paper, to help them understand that their course notes are just another SAS run (just a very long one).

GUIDELINES

1. Output is printed on 3-hole-punched paper (FORMS=999), output is formatted to 78 columns using the OPTIONS statement (80 columns is too wide for this form). Be sure the the DATE and NUMBER system options are turned on. At the first class session, help the students find the start of the SAS log and the print file (tell them the page numbers in the log are in the upper left-hand-side, in the print file on the upper right-hand-side). Use the PAGE statement to start each new topic on a new page. See Figure 1.

```
// EXEC SAS
/*JOBOUT FORMS=999,BURST=Y
//SYSIN DD *
OPTIONS LS=78 DATE NUMBER; PAGE;
*****************************************************
*                                                   *
*           INTRODUCTION TO SAS                     *
*                                                   *
*                                                   *
* (Course dates, instructor name, etc. here)...     *
*                                                   *
.
.
.
*****************************************************;PAGE;
```

Figure 1 Batch job setup

2. Put a RUN statement just before the PAGE statement. This is one of the few occasions for using a RUN statement in batch mode. It forces SAS to print any NOTEs generated by the last example of a topic before going on to the next topic (page). Without RUN, notes such as "DATA SET WORK.WEATHER HAS 6 OBSERVATIONS..." would appear on the next page just before the first DATA or PROC step in that set of examples, totally out of context.

   Since the PAGE statement is not reprinted in the SAS log, put it on the same line as the RUN statement (RUN; PAGE;). Otherwise students wonder why there is a missing line number.

   Defer explanations of RUN and PAGE until later in the course, when students have gotton used to the unusual format, and have a better concept of step boundaries.

```
404         DATA WEATHER;
405           INPUT LATITUDE HIGHTEMP LOWTEMP;
406           CARDS;

NOTE: DATA SET WORK.WEATHER HAS 6 OBSERVATIONS AND 3 VARIABLES.
      1676 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.01 SECONDS AND 712K.

413         RUN;

11     SAS(R) LOG   OS SAS 5.18          MVS/ESA JOB KVM      STEP SAS

414    ************************************************************
415    *                                                          *
416    *  DATA Statement (63-67)                                  *
417    *                                                          *
418    *  - signals to SAS that you want to enter a DATA step and *
419    *    begin building a SAS data set                         *
420    *                                                          *
       (etc.)
```

Figure 2 Each new topic begins on a new page

3. Material at the start of each new topic is enclosed in a box of asterisks to set it apart. Explain that the boxes are really comment statements, especially if the text in the box contains a semicolon. In Figure 3, the semicolon on the line with "form:" actually ends the comment begun on the upper left-hand-side of the box. The next semicolon on the same line (right-hand-side of the box) begins a new comment, which ends at the lower right-hand-side of the box.

```
*****************************************************
*                                                   *
*  LIST statement  (161-162)                        *
*                                                   *
*  - prints the input data lines in the SAS log     *
*                                                   *
*  - form:     LIST;                                *
*                                                   *
*****************************************************;

DATA INSECTS;
  INPUT TRT1 TRT2 TRT3 TRT4 TRT5 TRT6;
  * no LIST statement;
  CARDS;
```

Figure 3 New topics enclosed in comment boxes

4. Use the LIST statement in DATA steps wherever possible, and follow each DATA step with a PROC PRINT of the new SAS data set (see Figure 4). Then students can see both the input and the output of the DATA step. Note that students will confuse the LIST statement with "list" style INPUT unless you warn them. It is also helpful to show them a different job output that reads an external file with a long record length, for them to learn to use the RULE markings that wrap onto the next line.

```
 (SAS log)
680         DATA ALLERGY;
681           INPUT CITY $ CLINIC SERUM $ QUANTITY;
682           LIST;
683           CARDS;

RULE:       ----+----1----+----2----+----3----+----4
684         ATLANTA   3  A  204.7
685         DENVER    2  A   11.0
686         HONOLULU  1  A   87.5
687         JUNEAU    1  A   14.7
688         NEWARK    2  A   73.8
689         OMAHA     3  A   24.5
690         TRENTON   2  A   66.0
691         MIAMI     1  A   65.2
```

```
  (Print file)          SAS DATA SET ALLERGY

              OBS   CITY        CLINIC   SERUM   QUANTITY

              1     ATLANTA     3        A       204.7
              2     DENVER      2        A        11.0
              3     HONOLULU    1        A        87.5
              4     JUNEAU      1        A        14.7
              5     NEWARK      2        A        73.8
              6     OMAHA       3        A        24.5
              7     TRENTON     2        A        66.0
              8     MIAMI       1        A        65.2
```

Figure 4 Students see both input and output of DATA step

5. Comments inside DATA or PROC steps explain the example currently being covered in class (Figure 5). Put at least one comment into each DATA or PROC step, even if you think the purpose of the example should be obvious. Whenever possible, put in extra examples that you usually would not have time to cover, in case the class moves more quickly than usual, or to give more advanced students something to look at while you help slower students with questions.

```
PROC PRINT DATA=COMPANY;
  TITLE 'SAS DATA SET COMPANY';
  FORMAT NETWORTH 15.2;
  * without the FORMAT statement, PROC PRINT rounded the
    values to integers;
DATA SCI;
  * scientific notation is acceptable numeric data;
  * 5E-7 means 0.0000005;
  INPUT BIOTIN MOBILITY;
  LIST;
  CARDS;
0     1415
5E-7 1354
1E-6 1311
(etc.)
```

Figure 5 Add many SAS comment statements to examples

6. The fact that the LIST statement dumps the contents of the input buffer into the SAS log can be used to illustrate the difference between / and #n in the INPUT statement. Using #n, the input buffer in n lines long, and all input lines are reprinted in the log. Using /, only the last line of the observation from the data file is reprinted. See Figure 6.

```
909         DATA WIND;
910           INPUT MONTH $ N NE E SE #2 S SW W NW;
911           LIST;
912           CARDS;

RULE:       ----+----1----+----2----+----3----+----4----+----5

913         JAN 3 6 5 7
914         1 1 0 2
915         FEB 1 7 4 12
916         1 2 0 0
917         MAR 1 9 7 10
918         2 1 0 0
919         APR 1 5 5 11
920         3 3 1 1
            (etc.)

939         DATA WIND2;
940           INPUT MONTH $ N NE E SE / S SW W NW;
941           * with / SAS reads the lines one at a time.
942            When the LIST stmt is executed, only the last
943             line of the two is available for listing;
944           LIST;
945           CARDS;

RULE:       ----+----1----+----2----+----3----+----4----+----5

947         1 1 0 2
949         1 2 0 0
951         2 1 0 0
953         3 3 1 1
            (etc.)
```

Figure 6 The log helps understanding of / vs. #

7. To explain SAS date values, print the input data lines, the unformatted SAS date value, and the formatted SAS date value. See Figure 7.

```
(program)

DATA RAYLEIGH;
  * date values handled;
  INPUT TESTDATE DATE9. +1 ORIGIN $6. +1 PURIFIER $15.
    +1 WEIGHT 7.;
  LIST;
  CARDS;
29NOV1893 NO      HOT IRON        2.30143
 5DEC1893 NO      HOT IRON        2.29816
 6DEC1893 NO      HOT IRON        2.30182
(etc.)
PROC PRINT;
  TITLE 'SAS DATA SET RAYLEIGH';
  TITLE2 'TESTDATE IS UNFORMATTED FOR PRINTING';
PROC PRINT;
  TITLE2 'TESTDATE IS FORMATTED FOR PRINTING';
  FORMAT TESTDATE DATE9.;
```

```
(Print file)                SAS DATA SET RAYLEIGH
                      TESTDATE IS UNFORMATTED FOR PRINTING

    OBS    TESTDATE    ORIGIN    PURIFIER          WEIGHT

      1      -24138    NO        HOT IRON         2.30143
      2      -24132    NO        HOT IRON         2.29816
      3      -24131    NO        HOT IRON         2.30182
    (etc.)

                          SAS DATA SET RAYLEIGH
                      TESTDATE IS FORMATTED FOR PRINTING

    OBS    TESTDATE    ORIGIN    PURIFIER          WEIGHT

      1    11/29/93    NO        HOT IRON         2.30143
      2    12/05/93    NO        HOT IRON         2.29816
      3    12/06/93    NO        HOT IRON         2.30182
    (etc.)
```

Figure 7 SAS date values illustrated

8. To illustrate system-dependent commands, such as the JCL needed for a batch SAS job (Figure 8), enclose the explanatory text and the example completely in comments.

```
*************************************************************
*                                                           *
*  INFILE Statement (99-128)                                *
*                                                           *
*  - use when data is on an external file, not in the       *
*    job stream                                             *
*                                                           *
*  - tells SAS which JCL DD card to use to find the         *
*    raw data                                               *
*                                                           *
*  - form:       INFILE fileref;                            *
*                                                           *
*    . fileref matches the fileref on the JCL DD            *
*      statement                                            *
*                                                           *
*  - WYLBUR's "show dsname" command may be used to help     *
*    identify a disk data set's storage format              *
*                                                           *
*  - options may be specified after the fileref             *
*                                                           *
*  EXAMPLE 1:  Data on a Wylbur disk in fixed block         *
*              format:                                      *
*                                                           *
*     //III JOB (AAAA,NNN,A),MALLEY                         *
*     //PROCLIB DD DSN=ZABCRUN.PROCLIB,DISP=SHR             *
*     // EXEC SAS                                           *
*     //IN DD UNIT=FILE,DISP=SHR,VOL=SER=FILE44,           *
*     //      DSN=AAAAIII.MOUSE.STUDY.DATA                 *
*     //SYSIN DD *                                         *
*     DATA;                                                *
*       INFILE IN;                                         *
*       INPUT MOUSE PREDRUG POSTDRUG;                      *
*     PROC PRINT;                                          *
*     PROC MEANS;                                          *
*                                                          *
(etc.)
```

Figure 8 Enclose JCL in comments

9. Developing examples for SAS procedures is particularly easy. Figure 9 shows part of the PROC PRINT section of the course.

```
PROC PRINT;
  TITLE 'PROC PRINT';
  TITLE2 'WITH NO PROCEDURE INFORMATION STATEMENTS';
PROC PRINT;
  ID SUBJ;
  TITLE2 'WITH AN ID STATEMENT';
PROC PRINT;
  VAR SUBJ SEX X1-X11;
  FORMAT X1-X11 6.2;
  TITLE2 'WITH A VAR AND FORMAT STATEMENT';
RUN;PAGE;
```

Figure 9 PROC step examples are easy to create

10. It is important to intentionally show warning messages and notes indicating potential problems (see Figure 10). However, the main set of course notes should not contain errors (intentional or otherwise), because students wonder if they can rely on the notes at all when they see "ERRORS ON PAGE n" at the end of the SAS log. Show real error messages in a separate job. The course notes job is a good place to explain _N_, _ERROR_, and how to read and make use of messages about missing values being generated: "2 at 2404:5." Many people use SAS for years without understanding some of these messages.

11. This format of course notes is well suited to showing the efficiency introduced by the use of the LENGTH statement to decrease the number of bytes used to store integers. Students can compare the number of observations that fit on a track with and without a LENGTH statement. See Figure 11.

```
2399      DATA COMPUTE;
2400      * this step shows what happens when calcu-
2401        lations are performed on missing values;
2402        INPUT A B;
2403        LIST;
2404        C=A+B;
2405        D=A/B*100;
2406        E=SQRT(D);
2407        CARDS;

RULE:     ----+----1----+----2----+----3----+----4----+

2408      4 9
2409      . 8
2410      12 15
2411      14 .
NOTE: MISSING VALUES WERE GENERATED AS A RESULT OF
      PERFORMING AN OPERATION ON MISSING VALUES.
      EACH PLACE IS GIVEN BY: (NUMBER OF TIMES) AT
      (LINE):(COLUMN).
      2 AT 2404:5   2 AT 2405:5   2 AT 2406:5

NOTE: DATA SET WORK.COMPUTE HAS 4 OBSERVATIONS AND 5
      VARIABLES. 1066 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.02 SECONDS AND 720K.

2412        PROC PRINT;
2413          TITLE 'SAS DATA SET COMPUTE';
NOTE: THE PROCEDURE PRINT USED 0.03 SECONDS AND 784K
      AND PRINTED PAGE 115.


2414        DATA COMPUTE2;
2415          * here we attempt to divide by zero;
2416          INPUT A B;
2417          LIST;
2418          D=A/B*100;
2419          CARDS;

RULE:     ----+----1----+----2----+----3----+----4----+----5

2420      4 9
2421      12 15
NOTE: DIVISION BY ZERO AT LINE 2416 COLUMN 6.
2422      17 0
A=17 B=0 D=. _ERROR_=1 _N_=3
2423      3 6
2424      15 11
NOTE: MATHEMATICAL OPERATIONS COULD NOT BE PERFORMED AT THE
      FOLLOWING PLACES. THE RESULT OF THESE OPERATIONS HAVE
      BEEN SET TO MISSING VALUES.
      EACH PLACE IS GIVEN BY: (NUMBER OF TIMES) AT
      (LINE):(COLUMN).
      1 AT 2416:5
NOTE: DATA SET WORK.COMPUTE2 HAS 5 OBSERVATIONS AND 3
      VARIABLES. 1676 OBS/TRK.
NOTE: THE DATA STATEMENT USED 0.02 SECONDS AND 712K.
```

Figure 10 Force warning messages

```
Without the LENGTH statement:

DATA SET WORK.LESIONS HAS 10 OBSERVATIONS AND 5 VARIABLES.
1066 OBS/TRK.


With the LENGTH statement:

DATA SET WORK.LESIONS HAS 10 OBSERVATIONS AND 5 VARIABLES.
3352 OBS/TRK.
```

Figure 11 Log shows effect of LENGTH statement

12. One DATA step can illustrate several concepts. The example below shows the usefulness of the END= option when using

the sum statement (TOTAL + COUNT) to accumulate totals. This is the first time the PUT statement is seen by the students as well. Students know that if they do not understand the form of the PUT statement here, they'll get it later when that topic is discussed in depth, and they can concentrate on the function of the sum statement. Later on, when the PUT statement is covered, some students already have a sense of its use.

By carefully adding additional features into examples, or by adding advanced examples that you cover in class only when students request it, you can give the more experienced students something to look at and learn about, while working with beginners on elementary concepts. See Figure 12.

```
DATA HELP;
  INFILE CARDS END=X;
  INPUT MONTH $ COUNT;
  TOTAL + COUNT;
  IF X=1 THEN PUT 'TOTAL REQUESTS FOR PROGRAMMING'
     'ASSISTANCE YEAR-TO-DATE IS '  TOTAL;
  CARDS;
JAN 402
FEB 507
MAR 603
APR 450
(etc.)
```

Figure 12 Illustrating several concepts at once

13. Until you can tell that the students understand the way in which the notes were produced, keep repeating that it is a real SAS run, just a very long one with lots of DATA and PROC steps. After they get used to the idea, they find additional things in the log to ask you about, such as why does SAS keep putting the word WORK in front of each data set name. Also, there are many chances for them to understand a concept if they missed it the first time around, because the concept keeps showing up in later examples, such as the @@ in an INPUT statement.

14. One logistical problem for the students is how to look at the SAS log and print file simultaneously (to compare the input and output of a DATA step or PROC step). If there is enough desk space in the classroom, they can set up two piles, and leaf through them separately. Students report that this completely solves the problem. If there isn't enough desk space, they are forced to spend time and energy flipping back and forth between the two.

CONCLUSIONS

Annotated output provides real-life SAS runs for students to learn from. New features in SAS are easily added in examples, and when the job is rerun to generate a new set of notes, the notes are automatically updated by the SAS system itself, rather than the course author trying to reproduce SAS output using a text editor. The SAS log also has plenty of "white space" on which the students can write notes.

AUTHOR

Karen G. Malley
ARC Professional Services Group, Inc.
1301 Piccard Dr., Rm. 202
Rockville, MD 20850
(301)258-6826

[1] "Introduction to SAS at NIH", by Karen G. Malley at the Laboratory of Statistical and Mathematical Methodology, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland with contributions by Ray Danner, Laura Brachman, and Elvira Agron.