Programming in Style

George Poirier

Background

- In IT business since 1982,
- Independent consultant since 1987,
- Worked for many government and private clients,
- Programming in SAS on-and-off since 1982,
- Use mainly Base SAS and SAS/AF.,
- Built over 30 SAS applications (> 1000 pgms) from scratch, maintained/enhanced many more,
- From simple “reporting” applications to more complicated “code generators” and “specification languages” written in SAS.
- Seen many different styles over the years (good and bad).

Definition

- Style:
  - *“the **distinguishing way** in which something is done, said, written, made, executed, etc.”,*
  - *“the **distinguishing character** of a particular type of writing”,*
  - *“the **way in which a particular** literary **work is expressed”,***
  - *“**conventions followed** by a publisher in using capitals, hyphens, certain spelling, etc.”*

(Webster's Encyclopaedic Dictionary)

Learn from others

- One way to improve your programming style is to borrow (copy) the “best practices” from many other programmers.
- People actually like it when you borrow their code.
- Copying is one of the best ways to learn,
- Many of the good coding practices and styles have many common features.
- Learn to distinguish between the good, the bad, and the ugly
- Principles of style are applicable to all languages, not just SAS

This presentation applies to **production code** only

A sample SAS program *(pg. 1 of 2)*

```
* ---- std-crs pgm;
* 2006-jul-31, jc, fbn const;
%LET Start='01-Sep-2008';
%LET End='31-Dec-2008';
filename Stds 'C:\Students.txt'; filename Tsts 'C:\Tests.txt';filename Scrs
    'C:\TestScores.txt';
filename Crs  'C:\Courses.txt'; filename Tchrs 'C:\Teachers.txt';
data stds; infile stds;
input @1 sid 6. @8 snme $char30. @40 sadrs $char20.
 @60 scit $char20. @90 stel $char12. @103 sstat 1.; if sstat=1 or sstat=1;
data Tests; infile Tsts;
 input @1 cid $char6. @8 tstid $char8. @17 tstnm $char20. @38 tstdt $char11. @50 tstat 1.;
  if tstat=1 & (&start < tstdt < &end);
data tstscrs;infile Scrs;
  input @1 cid $char6. @8 tstid $char8. @17 sid 6. @25 ststscr 5.2;
  if ststscr=. then ststscr=0;
proc sort data=tstscrs;by cid sid;
data crs; infile crs;
input @1 cid $char6. @8 crsnm $char30. @39 crslocn $char10. @49 tid $char6.
@56 crsenrol 4.; if crsenrol>0;
  proc sort data=crs; by tid;
data tchrs; infile tchrs;
  input @1 tid $char6. @8 tnm $char30. @40 offno $char5.
```

A sample SAS program (pg.2 of 2)

```
@46 Offtel $char12. @59 stat 1.;if stat=1;  proc sort data=tchrs; by tid;
proc summary data=tstscrs; by cid sid; var ststscr;
output out=scav mean=scrsav;
data crsts; merge crs (IN=In1) Tchrs (IN=In2);by tid;if in1 & in2;
proc sort data=stds; BY sid;proc sort data=scav; BY SId;
data SA (KEEP=sid cid snme sadrs scit scrsav);
  MERGE stds (IN=In1) Scav (IN=In2); BY sId; IF In1 & In2;
proc sort data=sa;BY cid; proc sort data=crsts;BY cid;
data rcddtls; MERGE sa (IN=In1) crsts (IN=In2); BY cid; if in1 and in2;
  proc sort data=rcddtls;by sid cid;
run;
data _null_; set rcddtls; by sid cid; file print;
if first.sid then do;
  nocrs=0; tmrks=0; savg=0;
  put _page_; put / @15 'STUDENT ID:' @27 sid Z6. / @15 '      NAME:'
  @27 snme $char20. / @15 '   ADDRESS:' @27 sadrs $char20. / @27
  scit $char20. /// @15 'COURSE #' @25 'COURSE NAME' @55 'MARK' @65 'INSTR-NAME'
  / @15 70*'-';end;
nocrs + 1;tmrks+scrsav;
put / @15 cid $char6. @25 crsnm $char29. @55 scrsav 5.2 @65 tnm $char20.;
if last.sid then do;  savg = ROUND(scrsav/nocrs,0.1);
  put / @15 70*'-' / @38 'STUDENT AVERAGE:' @55 savg 4.1 / @15 70*'-';end;run;
```

A sample SAS Program

- What’s wrong with this program?
  - Inadequate comments (re. none),
  - Poor naming conventions used,
  - Inconsistent or ‘no’ indentation,
  - insufficient ‘white space’
  - Poor grouping of functionally related code (e.g., sorts and merges)
  - Misleading code (few KEEPs or RUNs, multiple statements per line)
  - Not very readable (good candidate for File 13).

Basic Principles of Style

- Readability
  - Use the 1-hour rule
- Maintainability
  - Keep maintainability in mind at all times
- Standardization
  - All your programs should be setup in a similar fashion

Readability

- Use of comments (style and approach)
- Naming conventions
  - – variable, file, constants, capitalization, etc.
- Code organization and layout conventions
  - – indentation, use of ‘white space’, order of statements,
- Key is readability and comprehension
- The 1-hour rule:
  - – *If you cannot tell what the program does after a 1 hour review, it probably needs to be re-written and/or re-commented.*

Readability - Comments

- **Use a standard header in every program**
  - Program name (with version)
  - System or Application,
  - Purpose, special notes, etc
  - Author (very important)
  - Change history (if no configuration software is being used)
- **Identify every major step in the program**
  - it should explain ‘step-by-step’ what your program does.
- **Identify the end of the program** (code).

Readability - Comments

- Keep comments general.
- Make sure the comments and code agree.
- Don’t comment bad code, re-write it.
- Comment tricky code
  - explain what the code is supposed to do
- Provide examples in the comments (if needed)
- Don’t keep commented code in the program
  - create a new version

Reabability – Naming Conventions

- Prefixes, suffixes, mixed case vs. same case
- Dataset variables, local variables, constants, flags, indicators, counters
- Capitalization of (SAS) keywords, constants, library names, etc.
- File naming conventions
  - – Rename the file if the data changes **(Prices, PricesSrtd, PricesWgtd,..).**

Readabiltiy - Code Layout

VERSION 1: (original code)

```
if eof=0 then do; grp=ing; dist=‘Ont’; rc+1; if flg_not=0
   then gri=‘N’; end; else do; put @1 ‘n =‘ rc; end;
```

VERSION 2: (indentation, better, but still bad)

```
if eof=0 then do;
   grp=ing; dist=‘Ont’; rc+1;
   if flg_not=0 then
   gri=‘N’; end;
else do;
   put @1 ‘n =‘ rc; end;
```

Readabiltiy - Code Layout

VERSION 3: (DO-END lined up, better names, more space)

```
IF eof = 0 THEN
  DO;
    group  = ingrp;
    dstrct = ‘Ont’;
    reccnt + 1;
    IF flag_not = 0 THEN
       groupincl = ‘N’;
  END;
ELSE
  DO;
     put @1 ‘n =‘ reccnt;
  END;
```

Readabiltiy - Code Layout

VERSION 4: Naming (vars., keywords, etc.), cleaner logic, more obvious

```
IF _N_ = 1
THEN
  District = ‘Ont’;        /* -- All recs. are ‘Ont’ -- */

Group = InGroup;
RecordCount + 1;

IF IncludeFlag = EXCLUDED         /* -- EXCLUDED = 0 -- */
THEN
  GroupIncludedInd = ‘N’;

IF EndOfFile
THEN
  PUT @1 ‘Number of Records Read = ‘ RecordCount 7.;
```

Standardization – Code Layout

- Group code together that goes together (e.g., MERGE)

```
/* -------------------------------------------------------------------- */
/* STEP 12: MERGE REVISIONS FILE WITH WEIGHTS FILE                      */
/*          TO APPLY WEIGHT FACTOR TO REVISED PRICES.                   */
/* -------------------------------------------------------------------- */
PROC SORT DATA=Revisions
           OUT=RevisionsSrtd;
  BY QuarterId WeekId;
RUN;

PROC SORT DATA=BasicWeights
           OUT=BasicWeightsSrtd;
  BY QuarterId WeekId;
RUN;

DATA RevisionsWgtd (KEEP=QuarterId WeekId Price WeightFactor PriceWgtd);
  MERGE RevisionsSrtd    (IN=InRevisions)
        BasicWeightsSrtd (IN=InBasicWeights);
     BY QuarterId WeekId;

  IF InRevisions AND InBasicWeights
  THEN
    PriceWgtd = Price * WeightFactor;
RUN;
```

Standardization - Code Layout

- Setup each program in your group (team, application, project, division, etc.) in the same fashion.
  - Standardize common approaches,
  - Standard record layouts can be %Included,
  - Read and edit input files first,
  - Place outputs in proper order (e.g., Error rpt. before Final rpt.)
- Place all ‘global’ %INCLUDEs in the same place
  - e.g., included macros
- Setup each data step in a similar fashion and order
  - INFILE, FILE, SET, INPUT, RETAIN, etc.

Maintainability

- Use consistent indentation of code
  - lineup DO; and END; statements
- Format your program to make it easier to read and understand.
  - Even for simple maintenance tasks, re-format the program so it will be easier to maintain in the future.
- Break complicated equations into simpler steps,
- Use standard SAS functions, don’t write your own
  - e.g, VERIFY, INDEX

Maintainability

- Write out input parameters
  - to the log or to a more permanent file
- Edit your input file for unexpected values
  - Don’t let your programs run with garbage.
- Check for developer errors
  - Invalid values of internal codes, etc.
- Use the simplest, most appropriate language feature for the task
  - SELECT vs. IF-THEN-ELSE
- Print summary statistics for each major data step

Maintainability

- Use %INCLUDEs to re-use standard pieces of code
  - Record layouts, report headers, etc.
  - Like LEGO
- Use Macros to turbo charge your programs
  - Allows others to reuse complicated code that they may not understand,
  - Use them when appropriate
- Use LINK/RETURN like subroutines
  - Don’t use it without the Return
- Ensure Macros and subroutines only do 1 thing,
- Write code to trap division by zero errors, missing values, etc.

Maintainability

- Many coding problems start with the format of the data
  - Input and output files
- Change the format of the data if you can
  - Don’t write complicated code to process badly designed data files,
  - see if the files can be changed first
- Sometimes it’s better to reformat the data on input to make the program simpler
  - Depends on the size and purpose of the program.

A sample SAS Program (Improved)

```
/* -------------------------------------------------------------------------- */
/*                                                                            */
/* PROGRAM: SCHL_RPTS_REPORT_CARDS_V2_1.SAS                                   */
/*                                                                            */
/* SYSTEM : STUDENT MARKS AND REPORTS APPLICATION                             */
/*                                                                            */
/* PURPOSE: TO PRINT STUDENT REPORT CARDS FOR A SPECIFIC SEMESTER             */
/*                                                                            */
/* NOTES  : - ONLY CURRENT F/T AND P/T STUDENTS ARE SELECTED                  */
/*          - MISSED TESTS ARE GIVEN A MARK OF ZERO                           */
/*                                                                            */
/* USAGE  : YOU MUST PROVIDE THE CORRECT SEMESTER START AND END DATES TO      */
/*          ENSURE THE CORRECT TESTS SCORES ARE SELECTED.                     */
/*                                                                            */
/* AUTHOR : Joseph Consultant                                                 */
/*                                                                            */
/* -------------------- MODIFICATIONS --------------------------------------- */
/* JC  - Joseph Consultant, Fly-by-Night Software Inc.                        */
/* GLP - George L. Poirier, Autumn Group Inc.                                 */
/*                                                                            */
/*                                                                            */
/* YY-MM-DD INIT VER ---------------- DESCRIPTION --------------------------- */
/* 06-07-31 JC   1.0 Created.                                                 */
/* 08-11-09 GLP  2.0 Updated and Reformatted to make it easier to read        */
/* 08-11-10 GLP  2.1 Fixed bug with Student Avg. not displaying correct value */
/*                                                                            */
/* -------------------------------------------------------------------------- */
```

Improved pgm. (cont’d)

```
/* ------ SEMESTER START AND END DATES (FRMT = DD-MMM-YYYY) ----------------- */
%LET SemesterStartDt ='01-Sep-2008';
%LET SemesterEndDt   ='31-Dec-2008';


/* -------------------------------------------------------------------------- */
/* EXTERNAL FILE NAME DECLARATIONS                                            */
/*                                                                            */
/* -------------------------------------------------------------------------- */
FILENAME Students 'C:\Students.txt';
FILENAME Tests    'C:\Tests.txt';
FILENAME Scores   'C:\TestScores.txt';
FILENAME Courses  'C:\Courses.txt';
FILENAME Teachers 'C:\Teachers.txt';
```

Improved pgm. (cont’d)

```
/* -------------------------------------------------------------------------- */
/* STEP 1: READ STUDENTS FILE                                                 */
/*         RETRIEVE NAME AND ADDRESS                                          */
/*                                                                            */
/*         VALID VALUES OF StudentStatus:                                     */
/*                       1 = Current FT,                                      */
/*                       2 = Current PT,                                      */
/*                       3 = Dropped Out,                                     */
/*                       4 = Graduated                                        */
/* -------------------------------------------------------------------------- */
DATA Students (KEEP=StudentId StudentName StudentAddress StudentCity);

  INFILE Students;

  INPUT @1   StudentId             6.
        @8   StudentName     $CHAR30.
        @40  StudentAddress  $CHAR20.
        @60  StudentCity     $CHAR20.
        @90  StudentPhone    $CHAR12.
        @103 StudentStatus         1.;

  IF StudentStatus IN(1 2);
RUN;
```

Improved pgm. (cont’d)

```
/* -------------------------------------------------------------------------- */
/* STEP 6: CALCULATE THE STUDENTS AVERAGE MARK PER COURSE                     */
/*         (THE TEST SCORES ALL HAVE EQUAL WEIGHT)                            */
/* -------------------------------------------------------------------------- */
PROC SORT DATA=TestScores
           OUT=TestScoresSrtd;
  BY CourseId StudentId;
RUN;

PROC SUMMARY DATA=TestScoresSrtd;
  BY CourseId StudentId;
  VAR StudentTestScore;
  OUTPUT OUT=StudentCourseAvgs
        MEAN=StudentCourseAvg;
RUN;
```

Improved pgm. (cont’d)

```
/* -------------------------------------------------------------------------- */
/* STEP 7: GET NAME OF TEACHER FOR EACH COURSE                                */
/*                                                                            */
/* -------------------------------------------------------------------------- */
PROC SORT DATA=Courses
           OUT=CoursesSrtd;
  BY TeacherId;
RUN;

PROC SORT DATA=Teachers
           OUT=TeachersSrtd;
  BY TeacherId;
RUN;

DATA CourseTeachers (KEEP=CourseId CourseName TeacherName OfficePhone);

  MERGE CoursesSrtd  (IN=InCourses)
        TeachersSrtd (IN=InTeachers);
     BY TeacherId;

  IF InCourses AND InTeachers;

RUN;
```

Improved pgm. (cont’d)

```
/* -------------------------------------------------------------------------- */
/* STEP 8: MATCH STUDENT NAMES TO COURSE MARKS                                */
/*                                                                            */
/* -------------------------------------------------------------------------- */
PROC SORT DATA=Students
           OUT=StudentsSrtd;
  BY StudentId;
RUN;

PROC SORT DATA=StudentCourseAvgs
           OUT=StudentCourseAvgsSrtd;
  BY StudentId;
RUN;

DATA StudentAverages (KEEP=StudentId      CourseId    StudentName
                           StudentAddress StudentCity StudentCourseAvg);

  MERGE StudentsSrtd          (IN=InStudents)
        StudentCourseAvgsSrtd (IN=InCourseAvgs);
     BY StudentId;

  IF InStudents AND InCourseAvgs;

RUN;
```

Improved pgm. (cont'd)

```
/* -------------------------------------------------------------------------- */
/* STEP 9: ADD COURSE AND TEACHER INFORMATION TO STUDENT COURSE MARKS         */
/*         THIS FILE WILL CONTAIN ALL DATA REQUIRED FOR THE REPORT CARD       */
/* -------------------------------------------------------------------------- */
PROC SORT DATA=StudentAverages
           OUT=StudentAvgsSrtd;
  BY CourseId;
RUN;

PROC SORT DATA=CourseTeachers
           OUT=CourseTeachersSrtd;
  BY CourseId;
RUN;

DATA ReportCardDetails (KEEP=StudentId StudentName StudentAddress StudentCity
                             CourseId CourseName TeacherName StudentCourseAvg);

  MERGE StudentAvgsSrtd    (IN=InStudentAvgs)
        CourseTeachersSrtd (IN=InCourseTeachers);
     BY CourseId;

  IF InStudentAvgs AND InCourseTeachers;

RUN;
```

Improved pgm. (cont’d)

```
/* -------------------------------------------------------------------------- */
/* STEP 10: PRINT REPORT CARDS                                                */
/*                                                                            */
/* -------------------------------------------------------------------------- */
PROC SORT DATA=ReportCardDetails
           OUT=ReportCardsSrtd;
  BY StudentId CourseId;
RUN;
```

Improved pgm. (cont’d)

```
DATA _NULL_;
  SET ReportCardsSrtd;
   BY StudentId CourseId;

  RETAIN NumCourses
         TotalMarks 0;

  /* ----- INITIAL PROCESSING (FOR EACH NEW STUDENT) ----- */
  IF FIRST.StudentId
  THEN
    DO;
      NumCourses = 0;
      TotalMarks = 0;
      StudentAvg = 0;

      PUT _PAGE_;
      PUT / @15 'STUDENT ID:' StudentId           Z6.
          / @15 '      NAME:' StudentName    $CHAR20.
          / @15 '   ADDRESS:' StudentAddress $CHAR20.
          / @27               StudentCity    $CHAR20.
        /// @15 'COURSE #'
            @25 'COURSE NAME'
            @55 'MARK'
            @65 'INSTR-NAME'
          / @15 70*'-';
    END;
```

Improved pgm. (cont’d)

```
 /* ----- MAIN PROCESSING (EVERY COURSE RECORD) --------- */
  NumCourses + 1;
  TotalMarks + StudentCourseAvg;

  PUT / @15 CourseId          $CHAR6.
        @25 CourseName       $CHAR29.
        @55 StudentCourseAvg       5.2
        @65 TeacherName      $CHAR20.;


  /* ----- FINAL PROCESSING (PRINT OVERALL AVERAGE) ----- */
  IF LAST.StudentId
  THEN
    DO;
      StudentAvg = ROUND(TotalMarks / NumCourses, 0.1);
      PUT / @15 70*'-'
          / @38 'STUDENT AVERAGE:‘
            @55 StudentAvg   4.1
          / @15 70*'-';
    END;

RUN;

/* ------------------- END OF PROGRAM --------------------------------------- */
```

Implementation

- **Team Effort;**
  - Cannot determine a style by yourself (unless you work alone).
  - What is readable to you is junk to someone else
  - No one style is ideal for everyone.
  - Each group should adopt their own style and enforce it.
  - Don’t let the ‘lowest common denominator’ prevail
- **Only way to get better is to conduct code/style reviews**
  - You want other people’s input
- **Most coding problems are discovered in reviews**
  - Many are design based
- **Don’t stop with your first attempt**
  - It’s an on-going process

Implementation

- **Benefits;**
  - – Simpler and faster coding,
  - – Easier testing and maintenance,
  - – Easier Impact assessments,
  - – Easier Estimating
  - – Better documentation
    - can write a “code scanner” for comments
  - – Raises everyone’s skill level
  - – Allows better use of programming resources
  - – Increased productivity
  - – Less expensive development

Additional Reading

The Elements of Programming Style

B. W. Kernighan & P. J. Plauger, Addison-Wesley, 1978

The Elements of Style,

W. Strunk, Jr. & E. B. White, MacMillan, 1972

The Psychology of Computer Programming,

G. M. Weinberg, Van Nostrand Reinhold, 1971

The Mythical Man-Month

F. P. Brooks, Jr., Addison-Wesley, 1975

How to Communicate Technical Information,

J. Price & H. Korman, Benjamin/Cummings Publishing Co. Inc. 1993.

Programming in Style

The End