PharmaSUG 2023 - Paper AP-252

Program It Forward! – Thinking Downstream when Coding

Frank Canale, SoftwaRx, LLC

ABSTRACT

Within the pharmaceutical industry during these times, we as SAS® users are often faced with short timelines, demands for quick turnarounds, having to create reoccurring or multiple deliverables…all while doing so with decreasing resources. Couple this with ever increasing standards and standardization efforts, and we sometimes find there's less of a need to create programs from scratch. Instead, we try to harvest reusable code – copied from a prior study and modified to run on current study. However, in many instances this process causes unexpected headaches as one tries to understand what the original programmer wanted to do, why, and if it will even work for the next project!

INTRODUCTION

It's not very difficult to work on a project wearing "blinders" – in other words, creating code with the thinking that once you're done with development of the code, it all runs error-free, the desired output is created correctly, and it's validated, that you are done. As most of us know – that's not the case! Most often, we must revisit projects…sometimes several times. Deliverables change, then change multiple times again. Also, and even more common, code is taken from a study or a project and re-implemented on a similar study/project. With the advent of industry-wide CDISC data standards and in-company output standards, reusable code becomes an excellent tool to leverage time savings and efficiencies.

Otherwise very intelligent and very experienced programmers and SAS users know this but may still have the blinders on, as the below examples will show. The need to meet the timeline and deliver the product outweighs good programming practices sometimes. The goal of this paper will be to drive home the suggestion to think about how what you create will affect others and other projects down the road. This paper will describe some of the experiences the author has run into when using other's code with examples, it will explain some of the issues, and it will attempt to guide and remind those of us that taking the time to "program it forward" when creating code (and editing existing code where/when possible) will ensure that the next user (or even ourselves) who must copy and modify/run your program will be able to do so more efficiently and quickly. Using and re-using code is wrought with several problems:

- Lack of comments or lack of adequate comments
- Lack of spacing and punctuation
- Programming best practices not followed or worse, ignored altogether
- Misuse and questionable setup of macros
- Undecipherable/un-understandable code
- Original programmer leaves company or moves on to another project

Because of these – it's very difficult to understand or decipher what the program is doing! Therefore, when creating a program from scratch, or yes – even editing an existing program (if possible and time allows), consider the following when doing so:

- Your code probably WILL be reused by someone else in a future project
- A deliverable could be regenerated for annual updates, new data cuts, a shift in direction of the project, or any number of reasons

- You may have to revisit your code 3/6/9 months, a year, or even 2 years down the road and chances are, you may not remember what it was doing or why!

Much of the following information might seem obvious, and perhaps you as the reader think “I always adequately comment my code or use good programming practice when coding”…and perhaps you do…but there are far too many examples of project production code where forward thinking is not applied.

EXAMPLES, ISSUES

CODE WITHOUT COMMENTS OR MINIMUM/INADEQUATE COMMENTS

Example 1a:

```
%let linespp=25;

data _null_;
  seq = strip(&cohort);
  num2 = input(strip(substr(seq, length(seq))), best.);
  num1 = input(strip(substr(seq, 1, 1)), best.);
  call symput('start', trim(left(put(num1, 1.))));
  call symput('ended', trim(left(put(num2, 1.))));
run;

%put cohort=&cohort Start=start ended=&ended;


proc sort data=adam.adeg out=eg;
   by cohort usubjid visitnum visit egdtc egdy doslvlav paramcd;
   where upcase(paramcd) in ('HEGHRMN', 'EGHRMIN', 'EGHRMAX', 'BRHYTHM',
'TOTTM', 'REQQUAL')  and saffl='Y' and dtype^='LVPD' ;
run;

 data checkeg;
   set eg;
   by cohort usubjid visitnum visit egdtc egdy doslvlav  paramcd;
   if (first.paramcd = 0 or last.paramcd = 0) then output ;
run;

proc sort data=eg; by cohort usubjid visitnum visit egdtc egdy; run;

proc transpose data=eg out=eg_avalc (drop=_name_ _label_) prefix=aval_;
   where paramcd in ("BRHYTHM", "REQQUAL");
   by cohort usubjid visitnum visit egdtc egdy doslvlav;
   var avalc;
   id paramcd;
run;

proc transpose data=eg out=eg_avaln (drop=_name_ _label_) prefix=aval_;
   where paramcd ^= "BRHYTHM";
   by cohort usubjid visitnum visit egdtc egdy doslvlav;
```

```
   var avalc;
   id paramcd;
run;

data eg1;
  merge eg_avalc eg_avaln ;
  by cohort usubjid visitnum visit egdtc egdy doslvlav;
run;

data adsl;
  set adam.adsl(keep= cohort usubjid saffl subjid age sex race trtsdtm trtsdt
trtedt trtdurd dosehigh trt01pn eotstt);
  proc sort; by cohort usubjid;
run;

data finalx;
   merge eg1 (in=a) adsl(keep= cohort usubjid saffl subjid age sex race
trtsdt trtedt trtsdtm trtdurd dosehigh trt01pn eotstt);
   by cohort usubjid;
   if a;

   new_tottm = aval_tottm / (3600);

   length asr trtsdtc trtedtc lddtc col1-col9 $100.;

   array col aval_heghrmn aval_eghrmin aval_eghrmax;
   array coly $ col7 col8 col9 ;


   racep = strip(propcase(race));
   if race='BLACK OR AFRICAN AMERICAN' then racep='Black or African
American';
   asr = strip(put(age, 3.)) || "/" || strip(put(sex, $sex.)) || "/" ||
racep;
   *** DETERMINE VALUE FOR LDDT LAST DOSING DATE AND TIME FROM TRTEDTM ***;

   if trtsdt ^= . then trtsdtc = put(trtsdt, E8601DA.);
   if trtedt ^= . then trtedtc = put(trtedt, E8601DA.);

   if trtsdt = . and trtedt = . then lddtc = " ";
   else if eotstt ne '' then lddtc = strip(trtsdtc) ||"/^n"||
strip(trtedtc)||' ('||strip(put(trtdurd, best.))||')';
   else if eotstt eq '' then lddtc = trtsdtc;

   rectime = substr(strip(scan(egdtc, 2, 'T')), 1, 5);

   col1 = cats(subjid,'/^n',asr);
   col2 =
strip(put(trt01pn,trtfmt.))||'/'||strip(put(dosehigh,dshg.))||'/^n'||lddtc;
   col3 = strip(propcase(visit))||'/^n'||strip(scan(egdtc, 1, 'T'))||'
('||strip(put(egdy, best.))||')';

   if doslvlav^=. then col4 = strip(put(doslvlav, dslvl.));

   atime = (input(egdtc, e8601dt.) - trtsdtm)/3600;
   col5 = strip(rectime)|| "/^n" || strip(aval_reqqual) || "/^n"
||strip(put(new_tottm, 6.1));
```

```
   col6 = strip(aval_brhythm);

   do over col;
    coly = strip(put(col, 8.));
   end;
   keep cohort egdy col1 - col9 egdtc;
run;

proc sort data=finalx;
   where &start <= cohort <= &ended;
   by cohort col1 col2 egdy col4 col5;
run;

data final outdata.&pgm;
   length pgbk 8;
   set finalx;
   by cohort col1 col2 egdy col4 col5;
   retain pgbk linesleft;

*** ADD LOGIC FOR PAGE BREAKING ***;
   if _n_ = 1 then do;
      pgbk = 1;
      linesleft = &linespp - 3;
   end;
   else do;
      if first.col1 then numlines = 4;
      else if not first.col1 and first.egdy then numlines = 3;
      else numlines = 1;
      linesleft = linesleft - numlines;
   end;
   if linesleft <= 0 then do;
      pgbk = pgbk + 1;
      linesleft = &linespp - max(4, numlines);
      output;
   end;
   else do;
      output;
   end;
   drop numlines linesleft;
run;

proc sort data=final; by pgbk cohort col1 col2 egdy; run;

================================================================================
```

- There are multiple things being done in the code above…however it contains only 2 comment lines (highlighted above)
- This is just an excerpt of an entire program with little to no comments within the code at all.
- Now, it's basic code…but a programmer who must assume this program and try to figure out what it's doing will have to read through the code (which is recommended anyway) – but there should be comments to introduce each block of code where computations or record selections are done.
- Perhaps adding adequate comments describing the major sections will cut down on the amount of time a subsequent programmer would need to thoroughly read through the code.
- Example:

```
/*****************************************************************
 Set up macro variables for xxxxx and yyyy, and the lines per page
 *****************************************************************/
%let linespp=25;

data _null_;
  seq = strip(&cohort);
  num2 = input(strip(substr(seq, length(seq))), best.);
  num1 = input(strip(substr(seq, 1, 1)), best.);
  call symput('start', trim(left(put(num1, 1.))));
  call symput('ended', trim(left(put(num2, 1.))));
run;

%put cohort=&cohort Start=start ended=&ended;

/*****************************************************************
 Sort ADaM ADEG for the following tests (x1, y1, etc.)
 *****************************************************************/
proc sort data=adam.adeg out=eg;
   by cohort usubjid visitnum visit egdtc egdy doslvlav paramcd;
   where upcase(paramcd) in ('HEGHRMN', 'EGHRMIN', 'EGHRMAX', 'BRHYTHM',
'TOTTM', 'REQQUAL')  and saffl='Y' and dtype^='LVPD' ;
run;

/*****************************************************************
 …other comments describing the action…
 *****************************************************************/
data checkeg;
   set eg;
   by cohort usubjid visitnum visit egdtc egdy doslvlav  paramcd;
   if (first.paramcd = 0 or last.paramcd = 0) then output ;
run;
```

…and so on…

CODE WITHOUT PROPER PUNCTUATION OR SPACING

Example 2a:

```
data rawdm;
  length  studyid $10 domain $2  usubjid $25 subjid $20 siteid $10 brthdtc
$10 sex $1 ethnic $50 race $40;
  set rawdata.dm (rename=(studyid=_studyid siteid=_siteid  sex=_sex
ethnic=_ethnic race=_race));
if ( subject = 'XXXX-001001-028' ) or ( subject = 'XXXX-001018-003' ) or (
subject = 'xXXXX-001001-029' ) then
  do ;
    put "BOOY " subject = _sex = sex_std = ;
  end ;
    usubjid = "XX-" || strip ( subject ) ;
  domain = %upcase ( "&domain" ) ;
  studyid = 'xx-XXXX';
  subjid = substr(subject, 6);
  siteid = substr(sitenumber, 1, 6);
  sex = sex_std; /* per spec, this is good */
%MACRO SKIP ; /* try to figure out sex mess */
if ( subject = 'XXXX-001001-028' ) or ( subject = 'XXXX-001001-029' ) then
  do ;
    put "BOOY " subject = sex = ;
    sex = lowcase ( sex ) ;
  end ;
%MEND SKIP ;
  ethnic=ethnic_std;
  race=race_std;
  if race='BLACK OR AFRICAN AMERICA' then race='BLACK OR AFRICAN AMERICAN';
  brthdtc = strip(yob_raw);
  drop subjectid project: _: env: studysite: siten: siteg: insta: folder:
record: page: datap: sdv: target: min: max: save: sex_std ethnic_std race_std
       yob yob_raw yob_int yob_dd yob_mm /*site*/;
 run;

 %m_ssort ( rawdm , , subject ) ;

%put iwrs is *&iwrs* ;
&dumps proc freq data = rawdata . DrugCo_Study_ub&iwrs ( keep =
subject_cohort ) ;
/*for suppdm*/
data dm2;
   set rawdata.DrugCo_Study_ub&iwrs.;
if ( subject_id = 'XXXX-001001-028' ) or ( subject_id = ' XXXX-001018-003' )
or ( subject_id = 'XXXX-001002-006' ) then
  do ;
    put "BOOY " subject_id = sex = ;
*   sex = lowcase ( sex ) ;
  end ;
    sex = lowcase ( sex ) ;
   length subject $50;
   attrib YOB_RAW label="Year of Birth (Character)" length=$4;
   attrib iwranddt label="Week 2 IWRS Date Randomized" length=8
format=DATETIME22.3;
```

```
   attrib iwrandno label="IWRS Randomization Number" length=8;
   attrib iwpdecod label="IWRS Placebo Randomization Decode" length=$20;

   attrib iwblockn label="IWRS Randomization Block" length=8;
   attrib iwpblock label="IWRS Placebo Randomization Block" length=8;

   if rand_date_sys ne . and rand_time_sys ne . then do;
      iwranddt = input(trim(put(rand_date_sys,date9.)) || ":" ||
trim(put(rand_time_sys,time12.3)),datetime22.3);
   end;

   iwrandno =  randomization_number;
   iwpdecod = placebo_rerand_treat_decode;
   iwblockn = randomization_block;
   iwpblock = placebo_rerand_block;
   cohort = subject_cohort;
   subject = subject_id;
   yob_raw = strip(put(year_of_birth,best.));

   if iwrandno ne . or iwpdecod ne " " or iwblockn ne . or iwpblock ne . or
sex ne " " then
     output ;
   else if ( cohort eq "Cohort 4" ) then
     output ; /* basically, output everyone */
    format sex ;
    informat sex ;
   keep subject sex /*yob_raw */ iwranddt iwrandno iwpdecod visit_code
visit_name iwblockn iwpblock cohort;
&DUMPS PUT "BOOZ " SUBJECT = SEX = ;
run;
```

==========

- Lack of spacing between data/proc steps and macro calls, as well as lack of indenting, makes the code difficult to read and understand
- Like a book with no paragraph breaks and lack of proper spacing and punctuation, not many would want to try to read code like this.
- SKIP macro in the middle of the code doesn't help matters…perhaps a comment block to render the code section unusable, with comments why, would be better
- Example:

```
data rawdm;

  length studyid $10 domain $2  usubjid $25 subjid $20 siteid $10
         brthdtc $10 sex $1 ethnic $50 race $40;

  set rawdata.dm (rename=(studyid=_studyid siteid=_siteid  sex=_sex
                          ethnic=_ethnic race=_race));

   if (subject = 'XXXX-001001-028') or (subject = 'XXXX-001018-003') or
      (subject = 'XXXX-001001-029') then do;
       put subject = _sex = sex_std = ;
   end ;

   usubjid = "XX-" || strip (subject);
```

```
   domain = %upcase ("&domain") ;
   studyid = 'xx-XXXX';
   subjid = substr(subject, 6);
   siteid = substr(sitenumber,1,6);
   sex = sex_std;

   /*************************************************
   Per email 2023-Feb-02, following assignments not needed…FC…
   if (subject = 'XXXX-001001-028') or (subject = 'XXXX-001001-029')
      then do ;
       sex = lowcase ( sex ) ;
   end;
   ************************************************************/

   ethnic=ethnic_std;
   race=race_std;

   if race='BLACK OR AFRICAN AMERICA' then race='BLACK OR AFRICAN AMERICAN';

   brthdtc = strip(yob_raw);

   drop subjectid project: _: env: studysite: siten: siteg: insta: folder:
        record: page: datap: sdv: target: min: max: save: sex_std ethnic_std
        race_std yob yob_raw yob_int yob_dd yob_mm;
run;
```

…and so on…

LACK OF ADHERENCE TO PROGRAMMING BEST PRACTICES

Figure 3a:

```
data adsl;
  set adam.adsl;
  where scrnfl ^= "Y";     *** REMOVE SCREEN FAILURE ****;
  proc sort; by usubjid;
run;
```

The above code should be written as follows:

```
data adsl;
  set adam.adsl;
  where scrnfl ^= "Y";     *** REMOVE SCREEN FAILURE ****;
run;

proc sort;
   by usubjid;
run;
```

…Or better yet…

```
proc sql;
   create table adsl as
      select *
         from adam.adsl(where=(scrnfl ne "Y"))
         order by usubjid;
quit;
```

Example 3b:

```
data ae;
  set adam.adae;
  where tretemfl="Y";
run;


data ae2;
  set ae;

  if length(strip(aestdtc)) = 10 then astdt = input(aestdtc,yymmdd10.);
            -- more statements --
run;
```

The above code should be condensed into one DATA step:

```
data ae2;
```

```
  set adam.adae(where=(trtemfl="Y"));

  if length(strip(aestdtc)) = 10 then astdt = input(aestdtc,yymmdd10.);
            -- more statements --
run;
```

Example 3c:

```
data co_lb_new;
merge  s_co_lbseq(in=a) s_lb_test_code(in=b);
by usubjid lbrefid test_code;
if  a  and  b
then  do;
            idvar='LBSEQ';
            idvarval=strip(put(lbseq,best.));
            output;
        end;
else  if  a
       then  put 'WAR' 'NING: No match in LB ' usubjid= lbrefid= test_code=;
run;
```

The above code should be written as follows, using spacing and punctuation:

```
data co_lb_new;
   merge  s_co_lbseq(in=a)
          s_lb_test_code(in=b);
   by usubjid lbrefid test_code;

   if a and b then do;
      idvar='LBSEQ';
      idvarval=strip(put(lbseq,best.));
      output;
   end;
   else if a then
      put 'WAR' 'NING: No match in LB ' usubjid= lbrefid= test_code=;
run;
```

Example 3d:

```
data inform1;
  merge rawdm rawex ic01 rawae rawenrol1;
  by subject;
  subject = strip(subject);
  length ageu $10 country $3;
  if yob_yyyy ^= . and icdt ^= . then do;
*    if rescrdt_yyyy=" " then age = year(datepart(icdt))-yob_yyyy;
```

```
    if ( rescrdt_yyyy eq . ) then
      age = year ( datepart ( icdt )) - yob_yyyy ;
     else age = rescrdt_yyyy - yob_yyyy;
    ageu = 'YEARS';
  end;
if index ( subject , '1010-001' ) then put "BOOD " subject = age = icdt =
yob_yyyy = rescrdt_yyyy = ;
    /* looks like both main and QC are using the wrong variable per spec -
should be siteid per spec, main uses sitegroup */
  if substr(sitenumber,1,3)='001' then country='USA';
  if substr(sitenumber,1,3)='003' then country='ITA';
  if ( substr ( sitenumber , 1 , 3 ) eq "004" ) then
    country = "NLD" ;
  if substr(sitenumber,1,3)='005' then country='ESP';
  drop /*icdt*/ yob_yyyy RESCRDT_YYYY;
run ;

 %m_deletex ( rawdm rawex ic01 rawae rawenrol1 ) ;
 %m_ssort ( inform1 , , subject ) ; /* uhhh */

***** GET RFPENDTC ******;
%MACRO SKIP ; /* rewrite to order of spec */
*** actigraphy data***;
/* apparently no longer needed - was from rawdata act_epochsummary */

*** AE DATA ***;
proc sort data=rawdata.ae out=outae; where AETERM^=" ";  by subject aestdt
aeendt;
run;

data aex;
  set outae;
  by subject subject aestdt aeendt;
    if ( n ( aestdt , aeendt )) then
  maxdate = max(aestdt, aeendt);
  proc sort; by subject maxdate;
run;
.
.
.
. Many many statements follow…
.
.
.
data alldt1;
  set alldt;
  by subject chkdate;
  if last.subject;
  rfpendtc = put(chkdate, e8601da.);
  proc sort; by subject;
run;

%m_deletex ( alldt ) ;
%MEND SKIP ;
```

The above code is an excerpt of a program where a rather large chunk of the code was placed within a macro called “SKIP”. In addition, there were at least 3 other large chunks of code that were handled the exact same way. The “SKIP” macro is never called.

This works fine - but it makes readability and understanding of the code very difficult. There were no comments saying why the code is SKIPPED, when, who performed the skip, etc. It would be better practice to save the original program to an archived copy, then completely removing the skipped code or commenting it out. Naturally, comments describing why the code is “skipped’ or commented out are recommended.

Example 3e:

```
data rawae;
  set rawae;
  by subject;
  length dthdtc $20 dthfl $1;
    /* the change... */
.
.
. More statements…
.
.
.
/*
                                          Cumulative    Cumulative
TREATMENT_CODE    Frequency     Percent     Frequency      Percent
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
             1          42       75.00            42        75.00
             2          14       25.00            56       100.00
                      Frequency Missing = 1
                                            Cumulative    Cumulative
TREATMENT_DECODE    Frequency     Percent     Frequency      Percent
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
DrugName + SoC            42       75.00            42        75.00
Placebo + SoC             14       25.00            56       100.00
                        Frequency Missing = 1
*/
run ;
```

Having procedure output in the middle of a program is not good programming practice…programs are programs, output is output…we should not be mixing the two types of SAS files. If the output needs to be saved, it could be written out to an RTF/PDF file and archived, or placed into a “readme” file for future reference.

QUESTIONABLE OR UNNECESSARY SET UP AND CALLING OF MACROS

Example 4a:

```
%macro gendata;

*** DETERMINE DENOMINATORS ***;
   %if &cohort = 1 %then %do;
      %let start = 1;
      %let stop = 1;
   %end;

.
.
. More Statements
.
.

data adcv;
   set adam.adcv;
   where cohort in (1 2);
.
.
. More Statements
.
.
run;

%macro genstats;

*** GENERATE STATISTICS FOR OUTPUT FOR TYPE=UN ***;
proc mixed data=adcv method=reml maxiter=1000 maxfunc=5000;
   class trt(ref="Placebo") avisitn usubjid;
   model chg = trt avisitn trt*avisitn base /ddfm=kenwardroger;
   repeated avisitn / subject = usubjid type = un;
   lsmeans trt*avisitn/cl alpha = 0.1;
   estimate "Week 2 difference" trt 1 -1
            trt*avisitn 1 0 0 0 -1 0 0 0/
            cl alpha = 0.1;
   estimate "Week 4 difference" trt 1 -1
            trt*avisitn 0 1 0 0 0 -1 0 0/
            cl alpha = 0.1;
   estimate "Week 6 difference" trt 1 -1
            trt*avisitn 0 0 1 0 0 0 -1 0/
            cl alpha = 0.1;
   estimate "Week 10 difference" trt 1 -1
            trt*avisitn 0 0 0 1 0 0 0 -1/
            cl alpha = 0.1;
   ods output estimates=est lsmeans=lsmeans convergencestatus=constat;
run;

.
.
. More Statements
.
```

```
.
%mend genstats;

%genstats

*** EXP THE ESTIMATES FROM THE LSMEANS DATASET ***;
data lsmeans;
   set lsmeans;
   cohort = &a;
*** KEEP ONLY WEEKS 2 4 6 10 ***;
   if avisitn in(2,4,6,10);

   if trt =: "CK" then trt = 2;
   else trt = 1;

*** CREATE VARIABLE MEANCI90 TO CONCATENATE THE VARIABLES CREATED ABOVE ***;
   length meanci90 $40;
   meanci90 = strip(put(estimate, 8.1)) || " (" || strip(put(stderr, 8.1)) ||
")" || " [" || strip(put(lower, 8.1)) || ", " || strip(put(upper, 8.1)) ||
"]";
run;

*** SORT DATA FOR TRANSPOSING ***;
proc sort data=lsmeans;
   by cohort avisitn trt;
run;
.
.
. More Statements
.
.
%mend gendata;

%gendata;
```

In the above code, there is a macro created called GENDATA…and within this macro another macro is set up called GENSTATS. So, the program calls %GENDATA and within this macro %GENSTATS is also called, using input data created somewhere within the GENDATA macro. While this is perfectly acceptable practice, it would be better if the two macros were set up independently of one another. This way, changes to one macro do not affect the other macro, and vice-versa. This also promotes code readability, understanding, and debugging.

Additionally, the macro code in this program was only generated once. In cases like this, there is no need to create a macro in the first place! It's much better to reserve use of macros for multi-use or repetitive tasks, and in cases like this, and simply set up the program for single use.

Example 4b:

```
data qc ;
     retain &allvars ;
     set qc ;
     by cohort paramcd param avisitn avisit ;
     length c1 $200 header $200 ; /* 200 and 200 are what readrtf likes */
     c1 = scan ( q_q , 2 , "\" ) ;
     header = avisit ;
     %m_missto0 ( pla_ , from = . , to = sum ( pla_00 , 0 )) ;
     %m_missto0 ( act_ , from = . , to = sum ( act_00 , act_10 , act_20 ,
act_30 , act_05 , act_15 , 0 )) ;
     length c2 - c8 $75 ; /* leave c1 for label */
     if ( first . avisit ) then
       do ;
         /* set missing columns to zeros - only if at least one missing
column */
&dumps if _n_ lt 30 and avisit ne "Baseline" then put "BOO1a " pla_00 =
act_00 = act_10 = act_20 = act_30 = act_05 = act_15 = avisit = ;
         if ( sum ( pla_00 , act_00 , act_10 , act_20 , act_30 , act_05 ,
act_15 , 0 ) gt 0 ) then
           do ;
             if ( nmiss ( pla_00 , act_00 , act_10 , act_20 , act_30 , act_05
, act_15 )) then
               _fill = 1 ; /* fill in with zeros only if have at least one
non-zero value in the row of `N`s */
             %m_missto0 ( pla_00 , from = . , to = 0 ) ;
             %m_missto0 ( act_00 , from = . , to = 0 ) ;
             %m_missto0 ( act_10 , from = . , to = 0 ) ;
             %m_missto0 ( act_20 , from = . , to = 0 ) ;
&dumps if _n_ lt 30 then put "BOO1b " pla_00 = act_00 = act_10 = act_20 =
act_30 = act_05 = act_15 = ;
             %m_missto0 ( act_30 , from = . , to = 0 ) ;
&dumps if _n_ lt 30 then put "BOO1c " pla_00 = act_00 = act_10 = act_20 =
act_30 = act_05 = act_15 = _n_ = / ;
             %m_missto0 ( act_05 , from = . , to = 0 ) ;
             %m_missto0 ( act_15 , from = . , to = 0 ) ;
           end ;
         %macro setfirst ( var ) ;
          retain &var.t ;
          &var.t = &var ;
         %mend setfirst ;
         %setfirst ( pla_ ) ;
         %setfirst ( act_ ) ;
         retain tot_t ;
         tot_t = pla_t + act_t ;
         %setfirst ( pla_00 ) ;
         %setfirst ( act_00 ) ;
         %setfirst ( act_10 ) ;
         %setfirst ( act_20 ) ;
         %setfirst ( act_30 ) ;
&dumps if _n_ lt 30 then put "BOO1d " pla_00 = act_00 = act_10 = act_20 =
act_30 = act_05 = act_15 = _n_ = act_30t = / ;
         %setfirst ( act_05 ) ;
         %setfirst ( act_15 ) ;
         %setfirst ( tot_t ) ;
```

```
          %macro setx ( col , val ) ;
           if ( &val ne 0 ) or ( _fill ) then
             &col = strip ( put ( &val , 31. )) ;
          %mend setx ;
          /* first is just N, rest are n(%) */
          if ( cohort eq 2 ) then
            do ; /* pla, ck, ckpla, ck10, ck20, ck30, over */
              %setx ( c2 , pla_ ) ;
              %setx ( c3 , act_ ) ;
              %setx ( c4 , act_00 ) ;
              %setx ( c5 , act_10 ) ;
              %setx ( c6 , act_20 ) ;
              %setx ( c7 , act_30 ) ;
&dumps put "BOO1e " ( _all_ ) ( = ) / _n_ = / ;
              %setx ( c8 , tot_t ) ;
            end ;
          else if ( cohort eq 3 ) or ( cohort eq 4 ) then
            do ; /* ck, ck5, ck10, ck15 */
              %setx ( c2 , act_ ) ;
              %setx ( c3 , act_05 ) ;
              %setx ( c4 , act_10 ) ;
              %setx ( c5 , act_15 ) ;
            end ;
          else
            abort abend ;
        end ;
      else
        do ; /* these are n and % */
          %macro setxp ( col , val , den = ) ; * act_t ) ;
           %if ( &den eq ) %then
             %do ;
               %let den = &val.t ;
             %end ;
           if ( &val le 0 ) and ( &den gt 0 ) then
             &col = "0" ;
           else if ( &val eq . ) then
             do ;
             end ;
           else
             do ;
               if ( &val eq &den ) then
                 &col = strip ( put ( &val , 31. )) || " (" || strip ( put (
100 * &val / &den , 31.0 )) || ")" ;
               else
                 &col = strip ( put ( &val , 31. )) || " (" || strip ( put (
100 * &val / &den , 31.1 )) || ")" ;
             end ;
          %mend setxp ;
          if ( cohort eq 2 ) then
            do ; /* pla, ck, ckpla, ck10, ck20, ck30, over */
              %setxp ( c2 , pla_   ) ;
              %setxp ( c3 , act_   ) ;
              %setxp ( c4 , act_00 ) ;
              %setxp ( c5 , act_10 ) ;
              %setxp ( c6 , act_20 ) ;
              %setxp ( c7 , act_30 ) ;
```

```
            if ( n ( pla_ , act_ , act_00 /*, act_10 , act_20 , act_30*/ ))
then
              act_X = sum ( pla_ , act_ /*, act_00 , act_10 , act_20 ,
act_30*/ ) ;
            %setxp ( c8 , act_X , den = tot_t  ) ;
          end ;
        else if ( cohort eq 3 ) or ( cohort eq 4 ) then
          do ; /* ck, ck5, ck10, ck15 */
            %setxp ( c2 , act_ ) ;
            %setxp ( c3 , act_05 ) ;
            %setxp ( c4 , act_10 ) ;
            %setxp ( c5 , act_15 ) ;
          end ;
        else
          abort abend ;
      end ;
run ;
```

In the above code, there are multiple macro use issues. First, the program has multiple calls to external macros (see blue highlighted text), but there are no comments anywhere in the program describing what the external macros do. Secondly and more problematic, there are 3 macros created within one DATA STEP, as shown in the yellow highlighted sections. Macros within a program should be created outside/independent of a DATA step.

Macro creation and usage

Macros, while very powerful, should be created and used judiciously. Not every situation calls for a macro to be created. Generally, the following should be considered before creating a macro:

- If there's a need across a group or organization to perform a certain task in a standard way (i.e., standard Adverse Event tables, certain parts of CDISC or ADaM dataset creation, annual Safety reporting), creation of a universal, standard macro would ensure quality and efficiency.

- If there's a need within a program to perform the same operation on several variables (i.e., calculation and formatting of descriptive statistics for several variables within a table, generation of data collection and reporting for separate cohorts), an inter-program macro should be created and called to cut down on the need to copy/modify code.

- However, if an operation is performed only once within a program, a macro is not needed, and creation of one could hamper the ability for another user to decipher what the program is doing and debug it if necessary. In this case, basic SAS code is perfectly fine.

- Macros should be declared within their own blocks, never within DATA steps, and descriptive comments should be used to describe the operation as clearly and succinctly as possible.

INDECIPHERABLE/UNREADABLE/UN-UNDERSTANDABLE CODE

In the interest of not being redundant, I will not copy the above code examples here. However, it stands to reason that many, if not all, of the above issues and examples help to render a program very difficult (sometimes impossible) to read, understand, and to try to decipher what the program is supposed to do. As stated above, these are real-world examples in my current assignment. And in this experience, after trying to read the code and figure out what it is supposed to do, I was lucky enough to be able to simply run the code and retrieve the desired result. Of course, we can't always be so fortunate. If the program had not generated the desired results, or generated log errors/issues following the run, I as the responsible programmer on the current study would have had to debug the code as best as I could.

In the case of an erroneous run or receiving undesirable results, and trying to determine the cause of the issues and what the program is supposed to do, it probably would be worthwhile to spend time adding the necessary comments, punctuation and spacing, moving macro creation out of data steps or in a separate portion of the code, and such. This effort to "clean-up" the code while debugging and deciphering it is worth the extra time for the next time someone must reuse or rerun the program in question.

SUGGESTIONS – THE POINT

The following are some suggestions on how to create code using best practices and keeping in mind the possibility of reusability down the road:

- COMMENT COMMENT and COMMENT some more!
- Use Base SAS whenever possible and keep it SIMPLE, readable, and understandable
- Adhere to best programming practices
    - Separate DATA and PROC steps
    - Use "run;" at the end of each DATA/PROC step
    - Indent – use spaces, not tabs
    - Avoid having "run-on sentences" of code…end a line with the semi-colon and begin the next statement on a separate line
    - Use spaces between lines to separate blocks of code within a step
- Try to avoid macros that call other macros if possible
- Be judicious in use of macros
- If a program contains processing that is single use, don't make it a macro – keep it basic.
- Macros should be separate from data processing code and vice-versa

CONCLUSION

In the fast-paced, constantly changing realm of clinical/statistical programming, maximizing and increasing efficiencies and decreasing development time are key goals to meet when attempting to produce quality SAS-based output. As stated above, many of us are faced with constant and multiple deliverables, and the ability to turn items around quickly is paramount. Taking advantage of standards across studies within a project by reusing code is good, but sometimes difficult and time consuming if the original code is tough to understand or decipher.

Therefore, it's better to create code that not only works the first time but is reusable down the road. That's why it's best to Program It Forward – consider the potential future usage of your code when programming!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Frank Canale, SoftwaRx, LLC
E-mail: frankocb@yahoo.com