PharmaSUG China 2019 – Paper AD-049

Innovative Clinical Programming Methods

Richard Read Allen, Peak Statistical Services, Evergreen, CO, USA

ABSTRACT

Some innovative programming methods for clinical programming will be presented. This paper will show some alternative programming techniques using DOW-loop logic and Hash techniques that will make some of your clinical programming tasks more compact and easier.

INTRODUCTION

The DOW loop is a powerful technique that moves the DATA step SET statement inside of an explicitly-coded DO-loop. This gives the programmer complete control over retention of variable values and the population of the Program Data Vector (PDV) by allowing a natural isolation of DO-loop instructions related to a certain break-event. The DOW isolates actions performed before and after the DO-loop from the instructions inside the loop and eliminates the necessity of retaining or reinitializing variables in most applications. In its most basic and well-known form using a DO UNTIL (LAST.ID) construct, it naturally lends itself to BY-processing of grouped data.

BASICS OF THE DOW LOOP

The basic structure of the DOW loop is as follows [4: Dorfman/Shajenko]:

```
data ... ;
     <stuff done before break-event> ;
     do <index specs> until ( break-event ) ;
          set … ;
          by …;
          <stuff done for each record> ;
     end ;
    <stuff done after break-event... > ;
run ;
```

There are three separate sections of code in this structure where instructions can be grouped for different types of processing, depending on how you need to handle the data.

1. Between the top of the implied data step loop and before the first record in the by-group is read if the action needs to be done before the by-group is processed.
2. Inside the DOW-loop, for each record in the by-group, if the action needs to be done to each record.
3. After the last record in the by-group has been processed and before the bottom of the implied data step loop, if the action such as summarizing needs to be done after the by-group is processed.

Here's a brief description of how the inner loop works:

- The DOW-Loop itself begins with the DO UNTIL statement and takes control from the traditional implicit DATA step loop.
- Because the SET statement is inside of the DOW-loop, the loop is not exited until after the last record for the break-event has been processed.
- Variables populated inside of the loop are retained while all of the records for the break-event are read in.
- A single record for the break-event, containing the filled arrays, is output at the completion of the DOW-loop by the "implied" output at the end of the data step.

The Dorfman/Shajenko paper [4] has some detailed explanations of the inner workings of the DOW loop and its variations, including using multiple DOW loops in the same data step and using the DOW with the data step hash object.

BASICS OF THE DATA STEP HASH OBJECT

The basic pieces of the hash object that we will use are the declare statement and the defineKey, defineData, defineDone, find and replace methods. The four steps for building a hash using these components are

1. **Instantiate the hash object:** Use the DECLARE statement with the keyword HASH followed by the hash name to create the hash object. Options are specified within the parentheses and include DATASET to load the hash object from an existing SAS® dataset and ORDERED to specify how the data is returned in key-value order.
2. **Define the key part:** Use the DEFINEKEY method to create a key of one or more variables. The key must be unique.
3. **Define the data part:** Use the DEFINEDATA method to specify zero or more data variables to be associated with the key variables.
4. **Complete the definition of the hash object:** Use the DEFINEDONE method to conclude the definition of the hash object.

One uses the DATA step Component Interface and object dot notation

ObjectName.Method(Parameters)

to create and manipulate these component objects.

Hash objects read contents of a dataset into memory at the top of the DATA step. Operations on the hash object are run entirely in memory, which is usually faster than disk-based operations like a DATA step merge or PROC SQL.

Some of the methods that are available in the data step hash object are ADD, CHECK, DEFINEDATA, DEFINEDONE, DEFINEKEY, DELETE, FIND, OUTPUT, REMOVE and REPLACE. We will only use DEFINEDATA, DEFINEDONE, DEFINEKEY, FIND and REPLACE in our examples.

APPLICATIONS

We will look and four applications of the above techniques, alone or together, in clinical SAS programming.

1. Transposing a supplemental SDTM domain for merging with main domain (DOW)
2. Creating Y/N flags for a variety of conditions or medical history (DOW)
3. Creating a subset of a larger dataset (HASH)
4. Creating a baseline flag in an SDTM domain. (DOW and HASH)

APPLICATION #1 – TRANSPOSING A SUPPLEMENTAL DOMAIN

A good application of the DOW in clinical programming is to transpose a supplemental domain so that it can be merged with the parent domain by USUBJID and **SEQ. This of course can also be done via PROC TRANSPOSE but an extra data step would be required to convert IDVARVAL to numeric for merging with **SEQ. Using the DOW, we can do this in one step.

The code below transposes SUPPCM and keeps the ATC Level 2 Text (ATC02) and ATC Level 3 Text (ATC03) for merging with CM. After the loop is left after each IDVARVAL, the CMSEQ is derived as the numeric value of IDVARVAL. We can now merge this with CM by USUBJID and CMSEQ very easily.

```
data SUPPCM(keep=usubjid cmseq ATC:);
  do until(last.idvarval);
    set sdtmdata.SUPPCM;
    by usubjid idvarval;
    if qnam='ATC02' then ATC02=qval;
    if qnam='ATC02ID' then ATC02CD=qval;
  end;
  cmseq=input(idvarval,best.);
run;
```

This method is really useful when you need to do some derivations on the values in the supplemental domain. These could be done either inside or outside the loop. For example, QVAL is always character in the supplemental domain. If you need to convert it to numeric, this could be done inside the loop when you define that variable.

For example, let's say our SUPPCM dataset stores imputed start and end dates in records where QNAM='CMSTDTCI' or QNAM='CMENDTCI' and we need to convert these to numeric dates to do some calculations after merging with CM. We can add these calculations to the DOW loop as in the following example.

Example #1

A partial SUPPCM dataset is shown below:

USUBJID	IDVARVAL	QNAM	QLABEL	QVAL
101	1	ATC01	ATC Level 1 Text	ALIMENTARY TRACT AND METABOLISM
101	1	ATC01ID	ATC Level 1 Code	A
101	1	ATC02	ATC Level 2 Text	DRUGS FOR CONSTIPATION
101	1	ATC02ID	ATC Level 2 Code	A06
101	1	ATC04	ATC Level 4 Text	OSMOTICALLY ACTING LAXATIVES
101	1	ATC04ID	ATC Level 4 Code	A06AD
101	1	CMENDTCI	CM End Date - Imputed	2017-10-13
101	1	CMSTDTCI	CM Start Date - Imputed	2016-09-01
101	2	ATC01	ATC Level 1 Text	BLOOD AND BLOOD FORMING ORGANS
101	2	ATC01ID	ATC Level 1 Code	B
101	2	ATC02	ATC Level 2 Text	ANTIANEMIC PREPARATIONS
101	2	ATC02ID	ATC Level 2 Code	B03

101	2	ATC04	ATC Level 4 Text	IRON BIVALENT, ORAL PREPARATIONS
101	2	ATC04ID	ATC Level 4 Code	B03AA
101	2	CMENDTCI	CM End Date - Imputed	2017-10-13
101	2	CMSTDTCI	CM Start Date - Imputed	2017-08-01
102	1	ATC01	ATC Level 1 Text	ALIMENTARY TRACT AND METABOLISM
102	1	ATC01ID	ATC Level 1 Code	A
102	1	ATC02	ATC Level 2 Text	STOMATOLOGICAL PREPARATIONS
102	1	ATC02ID	ATC Level 2 Code	A01
102	1	ATC04	ATC Level 4 Text	OTHER AGENTS FOR LOCAL ORAL TREATMENT
102	1	ATC04ID	ATC Level 4 Code	A01AD
102	1	CMENDTCI	CM End Date - Imputed	2017-09-22
102	1	CMSTDTCI	CM Start Date - Imputed	2017-09-21

If we run the following code against this sample dataset above

```
data SUPPCM(keep=usubjid cmseq ATC: CM:);
  do until(last.idvarval);
    set sdtmdata.SUPPCM;
    by usubjid idvarval;
    if qnam='ATC02' then ATC02=qval;
    if qnam='ATC02ID' then ATC02CD=qval;
    if qnam='CMSTDTCI' then CMSTDTCI=input(qval,yymmdd10.);
    if qnam='CMENDTCI' then CMENDTCI=input(qval,yymmdd10.);
  end;
  cmseq=input(idvarval,best.);
run;
```

Our resultant dataset will be

Obs	USUBJID	ATC02	ATC02CD	CMSTDTCI	CMENDTCI	cmseq
1	101	DRUGS FOR CONSTIPATION	A06	20698	21105	1
2	101	ANTIANEMIC PREPARATIONS	B03	21032	21105	2
3	102	STOMATOLOGICAL PREPARATIONS	A01	21083	21084	1

APPLICATION #2 – CREATING Y/N FLAGS FOR CONDITIONS AND HISTORY

Another good use of the DOW is in creating flags for certain conditions or events. The following code reads through MH and CM to create flags for the following:

- Diabetes
- IV drug use
- Secondary infection
- Peripheral vascular or arterial disease – PVD/PAD

```
data med_hx(keep=usubjid diabfl ivblfl secinffl pvdpadfl);
  do until(last.usubjid);
     set sdtmdata.mh
         sdtmdata_cm;
     by usubjid;
     diab=max(diab,(index(upcase(mhhlgt),'DIABE')>0 |
                    index(upcase(mhterm),'DIABE')>0 |
                    index(upcase(atc02),'DIABETES')>0 |
                    index(upcase(atc03),'INSULIN')>0
                    ));

     iv=max(iv,(((index(upcase(mhterm),'IV')>0 |
                  index(upcase(mhterm),'INTRA')>0 |
                  index(upcase(mhterm),'IM')>0
                  ) &
                 upcase(mhdecod) in('DRUG ABUSE','SUBSTANCE USE')
                 )
                ));

     secinf=max(secinf,(mhcat=:'Relevant' &
                        index(upcase(mhterm),'SECOND')>0));

     pvdpad=max(pvdpad,(index(upcase(mhterm),'VASCULAR')>0 |
                        index(upcase(mhterm),'ARTERIAL')>0 |
                        index(upcase(mhterm),'PVD')>0 |
                        index(upcase(mhterm),'PAD')>0 |
                        index(upcase(mhdecod),'PERIPHERAL VENOUS')>0
                        ));
  end;
  length diabfl ivblfl secinffl pvdpadfl $1;
  diabfl=ifc(diab,'Y','N');
  ivblfl=ifc(ivbl,'Y','N');
  secinffl=ifc(secinf,'Y','N');
  pvdpadfl=ifc(pvdpad,'Y','N');
run;
```

Inside the loop we are creating binary 0/1 variables for each of the conditions by searching for certain terms contained in variables in MH or CM. The definitions and terms to search for were agreed upon with the sponsors. When we've completed the loop for each USUBJID, these four binary variables will indicate whether or not the subject has the condition. These are converted to Y or N outside the loop for each USUBJID.

Note that MH and CM were interleaved by USUBJID so that we could loop through all records in both datasets for a subject. This allows us to search terms in both datasets for any of the conditions.

APPLICATION #3 – EXTRACTING A SUBSET OF A LARGER DATASET

A simple illustration of the data step hash object is to extract certain records from a larger dataset. One nice feature of the hash object is that the larger dataset does not need to be sorted in the same order as the smaller one in order to do such an extraction. This saves a time-consuming sort (and possible re-sort), especially when the dataset is very large.

In the sample code below, the object is to extract only those parameters that are graded for toxicity for further processing from an ADLB dataset.

The first step is to use PROC SQL to identify the PARAMCD that are graded and create a dataset containing only these PARAMCD.

```
proc sql;
  create table graded_parms as
  select distinct paramcd
  from ADLB(where=(missing(atoxgrn)=0));
quit;
```

The second step is a data step where we set up the hash object containing the PARAMCD we need to extract at the top of the data step before the first observation in the larger dataset (ADLB) is read. This resides in memory and allows us to quickly search the ADLB for any record where PARAMCD is the same as one of the codes stored in memory by the DEFINEKEY statement and only keep those records.

```
data GRADED;
  if _n_=1 then do;
    declare hash g(dataset:'graded_parms');
    g.defineKey('paramcd');
    g.defineDone();
  end;
  set ADLB;
  if g.find()=0;
run;
```

The ADLB dataset was sorted by USUBJID, PARCAT1, PARAMCD, ADTM. It did not need to be sorted for the hash object to match the PARAMCD to the list of the ones we wanted to extract. The GRADED dataset will have the same sort order as ADLB but will only contain those PARAMCD that have a toxicity grade.

Even simpler, this whole process can all done in one data step with the hash object as follows:

```
data GRADED;
  if _n_ = 1 then do ;
    dcl hash g(dataset:"ADLB(where=(missing(atoxgrn)=0))") ;
    g.definekey("paramcd") ;
    g.definedone() ;
  end ;
  set ADLB;
  if g.find()=0 ;
run ;
```

APPLICATION #4 – FLAGGING BASELINE OBSERVATIONS

This last example shows how to use both the DOW and a data step hash object in the same data step. It was inspired by a solution posted by Paul Dorfman on SAS-L back in January 2019 solving a similar problem. The object is to choose the last non-missing observation prior to treatment start and label it as baseline by setting a flag variable to 'Y'.

In the following example, we will look at deriving the baseline flag in a lab dataset.

Example #2

Below is a portion of a sample lab dataset used during the creation of the SDTM LB dataset.

USUBJID	LBTESTCD	VISIT	LBDTC	RFSTDTC	LBSTRESN
1	ALB	1	2018-09-15T11:26	2018-10-15T12:32	40
1	ALB	2	2018-09-30T10:30	2018-10-15T12:32	38
1	ALB	3	2018-10-15T12:30	2018-10-15T12:32	38
1	ALB	4	2018-10-29T11:25	2018-10-15T12:32	42
1	ALB	5	2018-11-13T10:45	2018-10-15T12:32	40
1	ALB	6	2018-11-29T11:11	2018-10-15T12:32	42
1	ALB	7	2018-12-15T11:45	2018-10-15T12:32	40
1	ALB	8	2018-12-30T12:10	2018-10-15T12:32	41
1	ALB	9	2019-01-13T13:32	2018-10-15T12:32	38
2	ALP	1	2018-09-01T10:15	2018-09-30T10:35	122
2	ALP	2	2018-09-15T11:15	2018-09-30T10:35	125
2	ALP	3	2018-09-30T10:25	2018-09-30T10:35	.
2	ALP	4	2018-10-14T11:08	2018-09-30T10:35	130
2	ALP	5	2018-10-30T13:28	2018-09-30T10:35	128
2	ALP	6	2018-11-13T11:59	2018-09-30T10:35	126
2	ALP	7	2018-11-28T09:30	2018-09-30T10:35	128
2	ALP	8	2018-12-14T10:15	2018-09-30T10:35	.
2	ALP	9	2018-12-29T11:35	2018-09-30T10:35	124
2	HGB	1	2018-09-01T10:30	2018-09-30T10:35	.
2	HGB	2	2018-09-15T11:30	2018-09-30T10:35	.
2	HGB	3	2018-09-30T10:40	2018-09-30T10:35	31
2	HGB	4	2018-10-14T11:23	2018-09-30T10:35	30
2	HGB	5	2018-10-30T13:43	2018-09-30T10:35	28
2	HGB	6	2018-11-13T12:14	2018-09-30T10:35	26
2	HGB	7	2018-11-28T09:45	2018-09-30T10:35	28
2	HGB	8	2018-12-14T10:30	2018-09-30T10:35	.
2	HGB	9	2018-12-29T11:50	2018-09-30T10:35	24

The identification and labelling of the baseline observations can be done in one data step. Here is the code to accomplish this. It is explained in more detail below.

```
data BL_Flag;
  if _n_=1 then do;
    declare hash h();
    h.defineKey('usubjid','lbtestcd');
    h.defineData('_min');
    h.defineDone();
    do until(EOF);
      set Labs end=EOF;
      if h.find() then call missing(_min);
      if input(rfstdtc,b8601dt.)<input(lbdtc,b8601dt.) then do;
        h.replace(); /* See NOTE below */
        continue;
      end;
      if missing(lbstresn)=0 then do;
        if h.find() then _min=input(rfstdtc,b8601dt.)-input(lbdtc,b8601dt.);
        else _min=min(_min,input(rfstdtc,b8601dt.)-input(lbdtc,b8601dt.));
      end;
      h.replace();
    end;
  end;
  set Labs;
  delta_time=ifn(missing(lbstresn)=0 & rfstdtc>=lbdtc,
                 input(rfstdtc,b8601dt.)-input(lbdtc,b8601dt.),
                 constant("bigint")
                 );
  h.find();  /* fills in minimum for each key */
  LBBLFL=char(" Y",(delta_time=_min)+1);
run;
```

At the top of the data step, a hash object is set up to contain the key variables USUBJID and LBTESTCD, as well as a data variable for the minimum positive difference between the treatment start (RFSTDTC) and the datetime for an observed record (LBDTC) with non-missing LBSTRESN called _MIN.

Also at the top of the data step, a DOW is set up to loop through all of the observations in the lab dataset and store the minimum difference between the RFSTDTC and LBDTC for each of the key variables in DEFINEKEY in memory. Inside the DOW the following happens:

- The variable _MIN is set to missing at the top of the loop when the keys are first found using the FIND method.
- Next we check for whether the current observation is after treatment start. If it is, then the _MIN value is replaced with the current minimum value in the hash for the current key variables and we continue to the next observation.
- If the current observation is before treatment start, then we check that LBSTRESN is not missing. If it exists, then
    - we assign _MIN as the difference between treatment start and the observation time for the first time the key variables are encountered.
    - Otherwise, _MIN is the minimum of the current value in _MIN and the difference between treatment start and the observation time.
- The current value of _MIN is then replaced in the hash for the current key variables and we go to the next observation.

When we reach the end of the lab dataset, all combinations of key variables will have the minimum positive time between the treatment start and the observed time of the record stored in memory in the hash object.

Next we read in the lab dataset and calculate the difference (DELTA_TIME) between treatment start and the observed time of each record. If the observed time is after treatment start or LBSTRESN is missing, DELTA_TIME is set to a large number so that it can't be considered for the minimum. We use the FIND method to populate _MIN on each of the records for each key combination and when the DELTA_TIME=_MIN the observation is flagged with LBBLFL='Y'.

NOTE: This REPLACE method covers cases where no observation is prior to treatment start - Otherwise key not found below when the FIND method is used to populate _MIN

The results of running this code against the example dataset above is as follows:

USUBJID	LBTESTCD	VISIT	LBDTC	RFSTDTC	LBSTRESN	_MIN	DELTA_TIME	LBBLFL
1	ALB	1	2018-09-15T11:26	2018-10-15T12:32	40	120	2595960	
1	ALB	2	2018-09-30T10:30	2018-10-15T12:32	38	120	1303320	
1	ALB	3	2018-10-15T12:30	2018-10-15T12:32	38	120	120	Y
1	ALB	4	2018-10-29T11:25	2018-10-15T12:32	42	120	1.7977E308	
1	ALB	5	2018-11-13T10:45	2018-10-15T12:32	40	120	1.7977E308	
1	ALB	6	2018-11-29T11:11	2018-10-15T12:32	42	120	1.7977E308	
1	ALB	7	2018-12-15T11:45	2018-10-15T12:32	40	120	1.7977E308	
1	ALB	8	2018-12-30T12:10	2018-10-15T12:32	41	120	1.7977E308	
1	ALB	9	2019-01-13T13:32	2018-10-15T12:32	38	120	1.7977E308	
2	ALP	1	2018-09-01T10:15	2018-09-30T10:35	122	1293600	2506800	
2	ALP	2	2018-09-15T11:15	2018-09-30T10:35	125	1293600	1293600	Y
2	ALP	3	2018-09-30T10:25	2018-09-30T10:35	.	1293600	1.7977E308	
2	ALP	4	2018-10-14T11:08	2018-09-30T10:35	130	1293600	1.7977E308	
2	ALP	5	2018-10-30T13:28	2018-09-30T10:35	128	1293600	1.7977E308	
2	ALP	6	2018-11-13T11:59	2018-09-30T10:35	126	1293600	1.7977E308	
2	ALP	7	2018-11-28T09:30	2018-09-30T10:35	128	1293600	1.7977E308	
2	ALP	8	2018-12-14T10:15	2018-09-30T10:35	.	1293600	1.7977E308	
2	ALP	9	2018-12-29T11:35	2018-09-30T10:35	124	1293600	1.7977E308	
2	HGB	1	2018-09-01T10:30	2018-09-30T10:35	.	.	1.7977E308	
2	HGB	2	2018-09-15T11:30	2018-09-30T10:35	.	.	1.7977E308	
2	HGB	3	2018-09-30T10:40	2018-09-30T10:35	31	.	1.7977E308	
2	HGB	4	2018-10-14T11:23	2018-09-30T10:35	30	.	1.7977E308	
2	HGB	5	2018-10-30T13:43	2018-09-30T10:35	28	.	1.7977E308	
2	HGB	6	2018-11-13T12:14	2018-09-30T10:35	26	.	1.7977E308	
2	HGB	7	2018-11-28T09:45	2018-09-30T10:35	28	.	1.7977E308	
2	HGB	8	2018-12-14T10:30	2018-09-30T10:35	.	.	1.7977E308	
2	HGB	9	2018-12-29T11:50	2018-09-30T10:35	24	.	1.7977E308	

Note that the HGB for USUBJID=2, does not have any non-missing LBSTRESN prior to treatment start so _MIN is missing and LBBLFL is not populated for this subject.

CONCLUSION

The DOW-Loop and data step hash object are extremely powerful programming techniques Though these methods may at first seem peculiar, especially to the novice programmer, once they have been understood and mastered they can provide very simple solutions for programming a variety of clinical SAS programming tasks.

REFERENCES

DOW:

1. *2 PROC TRANSPOSEs = 1 DATA Step DOW-Loop*, Nancy Brucken, PharmaSUG 2007 Proceedings
2. *One-Step Change from Baseline Calculations*, Nancy Brucken, PharmaSUG 2008 Proceedings
3. *The DOW (not that DOW!!!) and the LOCF in Clinical Trials*, Venky Chakravarthy, SUGI 28 Proceedings
4. *The DOW loop unrolled,* Paul Dorfman & Lessia Shajenko, PharmaSUG 2008 Proceedings
5. *Practical Uses of the DOW Loop in Pharmaceutical Programming,* Richard R Allen, PhUSE 2009 Proceedings.

HASH:

1. Getting Started with the DATA Step Hash Object, Jason Secosky & Janice Bloom, PharmaSUG 2007 Proceedings
2. Getting Started with the DATA Step Hash Iterator, Janice Bloom & Jason Secosky, support.sas.com
3. Hashing: Generations, Paul Dorfman & Gregg Snell, SUGI 28 Proceedings
4. DATA Step Hash Objects as Programming Tools, Paul Dorfman & Koen Vyverman, SUGI 30 Proceedings
5. Think FAST! Use Memory Tables (Hashing) for Faster Merging, Greg Snell, SUGI 31 Proceedings
6. A Hash Alternative to the PROC SQL Left Join, Ken Borowiak, NESUG 2005 Proceedings
7. Dorfman, Paul M and Shajenko, Lessia S (2011). “Hash + Point = Key.” NESUG 2011 Proceedings.
8. *The SAS® Hash Object: It’s Time To .find() Your Way Around*, Peter Eberhardt, SAS Global Forum 2011 Proceedings
9. Merge vs. Join vs. Hash Objects: A Comparison using "Big" Medical Device Data, James Johnson, PharmaSUG 2012 Proceedings
10. Dorfman, Paul and Don Henderson. 2018. *Data Management Solutions Using SAS Hash Table Operations: A business Intelligence Case Study*, Cary NC; SAS Institute Inc.

ACKNOWLEDGMENTS

Thanks to Nancy Brucken for bringing the DOW technique to my attention and to Paul Dorfman for the inspiration for application #4, as well as furthering my knowledge in both areas.

RECOMMENDED READING

Data Management Solutions Using SAS Hash Table Operations: A business Intelligence Case Study by Paul Dorfman and Don Henderson.

CONTACT INFORMATION

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

Richard Read Allen
Peak Statistical Services
303-670-5386
rrallen@peakstat.com
www.peakstat.com