PharmaSUG 2014 – Paper BB02

Understanding and Applying the Logic of the DOW-Loop

Arthur Li, City of Hope National Medical Center, Duarte, CA

ABSTRACT

The DOW-loop is not official terminology that one can find in SAS ® documentation, but it was well-known and widely-used among experienced SAS programmers. The DOW-loop was developed over a decade ago by a few SAS gurus, including Don Henderson, Paul Dorfman, and Ian Whitlock. A common construction of the DOW-loop consists of a DO-UNTIL loop with a SET and a BY statement within the loop. This construction isolates actions that are performed before and after the loop from the action within the loop which results in eliminating the need for retaining or resetting the newly-created variables to missing in the DATA step. In this talk, in addition to understanding the DOW-loop construction, we will review how to apply the DOW-loop to various applications.

BACKGROUND

Consider the following data set, *Example1*. Suppose that you would like to calculate the mean score for each subject. Without using the MEANS procedure, a typical approach is to utilize BY-group processing to calculate the mean score, such as the solution in Program 1.

Example1.sas7bdat

	ID	SCORE
1	A01	3
2	A01	2
3	A02	4
4	A02	2

Program 1:

```
proc sort data=example1;
    by id;
run;

data calMean1 (keep=ID mean_score);
    set example1;
    by id;
    if first.id then do;
        total = 0;
        n = 0;
    end;
    total + score;
    n + 1;
    if last.id then do;
        mean_score = total/n;
        output;
    end;
run;

proc print data=calMean1;
run;
```

Output:

```
                   mean_
 Obs     ID        score


  1      A01        2.5
  2      A02        3.0
```

In Program 1, the ID variable is used as the BY variable. The variable TOTAL is used to accumulate the total score for each subject and N serves as a counter to count the number of observations of each by-group. Both TOTAL and N are initialized to 0 when FIRST.ID equals 1, which is necessary; otherwise, TOTAL and N will accumulate values for all the observations in the data set. During each iteration of the DATA step execution, TOTAL is accumulated with the values from the SCORE variable and N is incremented by 1 by using the SUM statements. The use of the SUM statement is critical because it ensures that both TOTAL and N will retain their values at the beginning of each iteration of the DATA step; otherwise, these two variables would be set to missing at the beginning of the DATA step iteration. Instead of the using the SUM statement, one can also use the RETAIN statement to retain the values of TOTAL and N, along with a simple assignment statement to add values from SCORE to TOTAL and 1 to N. At the end the DATA step, the mean scores are calculated by dividing TOTAL by N and outputting the result when reading the last observation of each subject.

INTRODUCTION TO DOW-LOOP

Program 1 can be rewritten by using the DOW-loop. The construct and idea of the DOW-loop is well presented in Paul Dorfman's paper (The DOW-Loop Unrolled). The structure of a DOW-loop is like the following:

```
data output-data-set;

    <Stuff done before break-event>;                         *1;

    do <index-variable=specification> until (break-event) ;   *2;
       set input-data-set;
       <Stuff done for each record>;
    end;

    <Stuff done after break-event>;                          *3;
run ;
```

This construction separates the DATA step statements into three separate sections:

1. The first segment is between the top of the implicit loop and before the first observation being read in a by-group.
2. A Do-until loop or a modified do-until loop: a certain type of action is performed on each observation within the by-group.
3. The last segment is between after reading and processing the last observation from the by-group and the end of the implicit loop of the DATA step.

The purpose of the construction of the DOW-loop is to isolate the executable statements between two consecutive break-events from the statements that are performed before and after a break-event. For example, Program 2 calculates the mean score for each subject by using the DOW-loop.

Program 2:

```
data calMean2 (keep=ID mean_score);
    do n = 1 by 1 until (last.id);
        set example1;
        by id;
        total = sum(total, score);
    end;
    mean_score = total/n;
run;
```

To better understand the construction of the DOW-loop, one should understand what happens in the PDV during the DATA step execution, which is illustrated in Figure 1 and Figure 2. In Program 2, variable N serves as a counting variable for counting the number of observations within each by-group. Variable The variable TOTAL does not need to be retained in the DATA step. There are a couple of reasons for not retaining TOTAL. First of all, TOTAL is assigned to a missing value at the beginning of each iteration of the DATA step execution. Within each iteration of the DATA step, SAS processes one entire by-group, which consists of multiple observations. Within each iteration of the DATA step execution, TOTAL is accumulated by using the explicit DO loop. After processing all the observations for a by-group, MEAN_SCORE is then calculated.

FIRST ITERATION OF DATA STEP:

```
data calMean2 (keep=ID mean_score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	1	1		•	•	•	•

EXPLANATION: _N_ is initialized to 1. Both FIRST.ID and LAST.ID are set to1. The rest of the variables are set to missing. (Note: _ERROR_ is not shown for simplicity purposes)

First iteration of the DO loop:

```
do n = 1 by 1 until (last.id);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	1	1		•	1	•	•

EXPLANATION: N is assigned to 1.

```
set example1;  by id;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	1	0	A01	3	1	•	•

EXPLANATION: The SET statement copies the first observation from EXAMPLE1 to the PDV. FIRST.ID is assigned to 1, and LAST.ID is assigned to 0.

```
total = sum(total, score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	1	0	A01	3	1	3	•

EXPLANATION: TOTAL is calculated.

```
end;
```

EXPLANATION: SAS reaches the end of the loop. Since LAST.ID does not equal 1, the loop continues.

Second iteration of the DO loop:

```
do n = 1 by 1 until (last.id);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	1	0	A01	3	2	3	•

EXPLANATION: N is incremented to 2

```
set example1;  by id;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	0	1	A01	2	2	3	•

EXPLANATION: The SET statement copies the second observation from EXAMPLE1 to the PDV. FIRST.ID is assigned to 0, and LAST.ID is assigned to 1.

```
total = sum(total, score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	0	1	A01	2	2	5	•

EXPLANATION: TOTAL is calculated.

```
end;
```

EXPLANATION: Since LAST.ID equals 1, the loop ends.

```
mean_score = total/n;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
1	0	1	A01	2	2	5	2.5

EXPLANATION: MEAN_SCORE is calculated

```
run;
```

EXPLANATION: SAS reaches the end of the DATA step. The implicit OUTPUT statement executes. The first observation is generated. Then, SAS returns to the beginning of the DATA step to start the next iteration.

Figure 1. The first iteration of DATA step of Program2 .

SECOND ITERATION OF DATA STEP:

```
data calMean2 (keep=ID mean_score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	0	1	A01	2	•	•	•

EXPLANATION: At the beginning of the 2nd iteration, _N_ is incremented to 2. ID and SCORE retained their values because these variables are read from the input data. N, TOTAL, and MEAN_SCORE are set to missing because these are the variables that are created in the DATA step.

First iteration of the DO loop:

```
do n = 1 by 1 until (last.id);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	0	1	A01	2	1	•	•

EXPLANATION: N is assigned to 1.

```
set example1;  by id;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	1	0	A02	4	1	•	•

EXPLANATION: The SET statement copies the third observation from EXAMPLE1 to the PDV. FIRST.ID is assigned to 1, and LAST.ID is assigned to 0.

```
total = sum(total, score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	1	0	A02	4	1	4	•

EXPLANATION: TOTAL is calculated.

```
end;
```

EXPLANATION: SAS reaches the end of the loop. Since LAST.ID does not equal 1, the loop continues.

Second iteration of the DO loop:

```
do n = 1 by 1 until (last.id);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	1	0	A02	4	2	4	•

EXPLANATION: N is incremented to 2

```
set example1;  by id;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	0	1	A02	2	2	4	•

EXPLANATION: The SET statement copies the fourth observation from EXAMPLE1 to the PDV. FIRST.ID is assigned to 0, and LAST.ID is assigned to 1.

```
total = sum(total, score);
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	0	1	A02	2	2	6	•

EXPLANATION: TOTAL is calculated.

```
end;
```

EXPLANATION: Since LAST.ID equals 1, the loop ends.

```
mean_score = total/n;
```

_N_ (D)	FIRST.ID (D)	LAST.ID (ID)	ID (K)	SCORE(D)	N(D)	TOTAL(D)	MEAN_SCORE(K)
2	0	1	A02	2	2	6	3

EXPLANATION: MEAN_SCORE is calculated

```
run;
```

EXPLANATION: SAS reaches the end of the DATA step. The implicit OUTPUT statement executes. The first observation is generated. Then, SAS returns to the beginning of the DATA step to start the next iteration

Figure 2. The second iteration of DATA step of Program2 .

When you apply a DOW-loop for an application, the data set is often grouped by values of one or more variables. A break-event can be identified when reading the last observation within a by-group, like in the previous example.

In some situations, you can also use the DOW-loop for the data sets that are not grouped by values of a variable. For example, based on the MISS data set, suppose you would like to modify the SCORE variable. If SCORE is missing for the current observation, you would like to use the SCORE value from the previous observation. The solution in Program 3 does not use the DOW-loop. The variable FOO is used to hold the current non-missing value, which needs to be carried forward to the next iteration of the DATA step. Thus, variable FOO needs to be retained in the DATA step by using the RETAIN statement; otherwise, FOO will be set to missing at the beginning of the DATA step iteration.

MISS.sas7bdat

	ID	SCORE
1	A	3
2	B	4
3	C	.
4	D	5
5	E	.
6	F	.

Program 3:

```
data p3 (drop=foo);
    set miss;
    retain foo;
    if not missing(score) then foo = score;
    else if missing(score) then score=foo;
run;

proc print data=p3;
run;
```

Output:

```
 Obs     ID     score

   1     A        3
   2     B        4
   3     C        4
   4     D        5
   5     E        5
   6     F        5
```

To modify Program 3 by using the DOW-loop, one needs to identify the break-event. Since the data set MISS is not grouped by the value of a variable, you can treat the entire data set as one single group and process the last observation as the break event. To know when to read the last observation from the input data, you can use the END= option in the SET statement. The variable that is specified in the END= option is a temporary variable that can be used to indicate when reading the last observation of the input data.

Program 4 is a modified version of Program 3 by using the DOW-loop. In this program, there is only one DATA step iteration. Within the DATA step, each observation from the input data is read into the explicit DO UNTIL loop. The variable The FOO variable does not need to be retained since there is only one DATA step iteration. FOO will hold its value throughout the DATA step execution.

Program 4:

```
data p4(drop= foo);
    do until (last);
        set miss end=last;
        if not missing(score) then foo = score;
        else if missing(score) then score=foo;
        output;
    end;
run;
```

DOUBLE DOW-LOOP

One of the techniques that Paul Dorfman presented in his paper is the use of more than one DOW-loop in the DATA step. For example, Program 2 calculated the mean score for each subject. Suppose that you would like to merge the mean score with the original data. This can be easily done by using the MERGE statement in the DATA step by merging the original data set with the data set that contains the calculated mean scores. This approach requires two DATA step executions. You can use one single DATA step that contains two DOW-loops to accomplish this task (See Program 5).

Program 5:

```
data p5 (drop=n total);
    do n = 1 by 1 until (last.id);
        set example1;
        by id;
        total = sum(total, score);
    end;
    mean_score = total/n;
    do until (last.id);
        set example1;
        by id;
        output;
    end;
run;

proc print data=p5;
run;
```

Output:

```
                              mean_
 Obs     ID       score       score

  1      A01        3          2.5
  2      A01        2          2.5
  3      A02        4          3.0
  4      A02        2          3.0
```

After the mean score is calculated in the PDV in Program 5, the second DOW loop reads each observation within the same by-group. The break-event of the second DOW-loop is the same as the first one, which is reading the last observation of a by-group. Dorfman explains that having the input data set EXAMPLE1 twice in the DATA step is the same as if there are two independent virtual data sets EXAMPLE1A and EXAMPLE1B which are identical to the EXAMPLE1 data set. Readers can refer to his paper for more detailed technicalities.

Instead of using LAST.ID as the break-event, you can modify the second DOW-loop by letting the loop iterate N times where N is the counter variable that resulted from the first DOW-loop (See Program 6). Before entering the second DOW-loop, the upper index of the loop is set with a value from N which is accumulated from the first DOW-loop. Then N is set to 1 and starts to iterate until the loop reaches the number of times that is specified in the upper index.

Program 6:

```
data p6 (drop=n total);
    do n = 1 by 1 until (last.id);
        set example1;
        by id;
        total = sum(total, score);
    end;
    mean_score = total/n;
    do n = 1 to n;
        set example1;
        output;
    end;
run;
```

LOCF (LAST OBSERVATION CARRIED FORWARD) EXAMPLE

One can use the DOW-loop to simplify the coding of a LOCF-type application. You can also find the reference for this technique from Richard Allen and Venky Chakravarthy papers.

For example, the data set PATIENT contains the triglyceride (TGL) measurement and smoking status (SMOKE) for patients for different time periods. Notice that some patients only have one measurement whereas others were measured more than once in different years. Suppose that you would like to create a data set that contains the most recent non-missing data. The resulting data set will have three variables: PATID (patient ID), TGL_NEW (the most recent non-missing TGL) and SMOKE_NEW (the most recent non-missing SMOKE). A couple of issues need to be considered for solving this problem. First of all, the most recent non-missing data for TGL and SMOKE occur at different time points. For instance, for patient A01, the most recent non-missing TGL is 150 in 2007 but the most recent non-missing SMOKE is “Y” in 2005. The second issue is that some patients might only have missing values for either TGL or SMOKE. In this situation, you only need to use the missing value in the resulting data set for this variable. For instance, the TGL measurement is missing for A03.

Patients.sas7bdat

	PATID	VISIT	TGL	SMOKE
1	A01	2005	.	Y
2	A01	2007	150	
3	A02	2004	.	
4	A02	2005	200	N
5	A02	2006	210	N
6	A03	2005	.	Y
7	A04	2002	164	
8	A04	2004	170	Y
9	A04	2006	190	
10	A04	2007	.	N
11	A05	2005	189	

Program 7 illustrates an example without using the DOW-loop. Since the TGL_NEW and SMOKE_NEW variables are used to hold the most non-missing recent values for each subject, these two variables need to be retained by using the RETAIN statement. Furthermore, these two variables need to be set to missing when reading the first observation for each patient to prevent them from carrying the values from the previous patients. During each iteration of the DATA step, the values from the TGL and SMOKE variables are assigned to TGL_NEW and SMOKE_NEW (respectively) provided that TGL and SMOKE are not missing. The subsetting-IF statement is used to control generating the output data when reading the last observation of each subject.

Program 7:

```
proc sort data=patients out=patients_sort;
    by patid visit;
run;

data patients_single (drop= visit tgl smoke);
    set patients_sort;
    by patid;
    retain tgl_new smoke_new;
    if first.patid then do;
        tgl_new = .;
        smoke_new = " ";
    end;
    if not missing(tgl) then tgl_new=tgl;
    if not missing(smoke) then smoke_new=smoke;
    if last.patid;
run;

proc print data=patients_single;
run;
```

Output:

```
                                     smoke_
 Obs     PATID     tgl_new      new

   1       A01        150          Y
   2       A02        210          N
   3       A03          .          Y
   4       A04        190          N
   5       A05        189
```

Program 8 is an alternative solution by using the DOW-loop. In Program 8, SAS reads all the observations for each subject in the DOW-loop within one single iteration of the DATA step. This construction eliminates the necessity for retaining both the TGL_NEW and SMOKE_NEW variables since these variables will retain their values while processing all the records within the same by-group. Furthermore, the TGL_NEW and SMOKE_NEW variables do not need to be set to missing explicitly because these two variables are set to missing at the beginning of the DATA step iteration, which is the same time when reading the first observation of each subject.

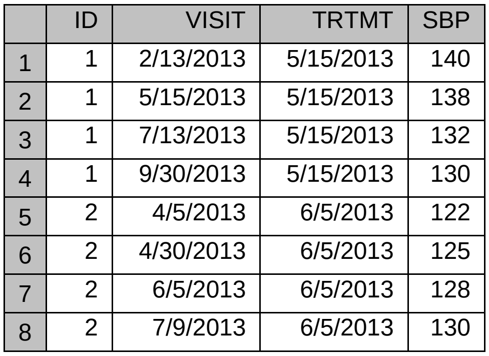
Program 8:

```
data patients_single1 (drop= visit tgl smoke);
    do until (last.patid);
        set patients_sort;
        by patid;
        if not missing(tgl) then tgl_new=tgl;
        if not missing(smoke) then smoke_new=smoke;
    end;
run;
```

CHANGE FROM BASELINE EXAMPLE

Using the DOW-loop can also simplify the application of the change-from-baseline calculation. The following example is adapted from the example in Allen's paper. For example, in the BASE data set, the variable VISIT is the visiting time and TRTMT is the treatment time or the baseline measurement time. SBP is the systolic blood pressures that are measured at each visiting time. Based on this data set, we would like to create a variable (B_SBP) which contains the SBP value at the treatment time. For SBP that is measured before the treatment time, B_SBP will be set to missing. Furthermore, we would also like to create a variable (C_SBP) which is the difference between the current SBP measurement and the baseline SBP measurement. For SBP that was measured before treatment date or on the treatment date, C_SBP will be set to missing.

Base.sas7bdat



	ID	VISIT	TRTMT	SBP
1	1	2/13/2013	5/15/2013	140
2	1	5/15/2013	5/15/2013	138
3	1	7/13/2013	5/15/2013	132
4	1	9/30/2013	5/15/2013	130
5	2	4/5/2013	6/5/2013	122
6	2	4/30/2013	6/5/2013	125
7	2	6/5/2013	6/5/2013	128
8	2	7/9/2013	6/5/2013	130

The DOW-loop is not used in Program 9. Both ID and VISIT are sorted first. Since the variable B_SBP needs to be carried forward to the next observation, the RETAIN statement is then used to retain its value. There are three conditions in the IF/THEN-ELSE statement to create the variables B_SBP and C_SBP. When the visiting time is less than the treatment time (VISIT<TRTMT), B_SBP is set to missing; this step prevents the baseline SBP from the current patient to carry forward to the next patient. When the visiting time equals the treatment time (VISIT=TRTMT), the baseline SBP (B_SBP) is assigned with the value from SBP. When the visiting time is greater than the treatment time (VISIT>TRTMT), C_SBP is calculated.

Program 9:

```
proc sort data=base;
    by ID visit;
run;

data p9;
    set base;
    retain b_sbp;
    if visit<trtmt then b_sbp = .;
    else if visit=trtmt then b_sbp = sbp;
    else if visit>trtmt then c_sbp=sbp-b_sbp;
run;

proc print data=p9;
run;
```

Output:

```
Obs     ID          visit           trtmt     sbp     b_sbp     c_sbp

  1      1     02/13/2013     05/15/2013     140         .         .
  2      1     05/15/2013     05/15/2013     138       138         .
  3      1     07/13/2013     05/15/2013     132       138        -6
  4      1     09/30/2013     05/15/2013     130       138        -8
  5      2     04/05/2013     06/05/2013     122         .         .
  6      2     04/30/2013     06/05/2013     125         .         .
  7      2     06/05/2013     06/05/2013     128       128         .
  8      2     07/09/2013     06/05/2013     130       128         2
```

Program 10 uses a DOW-loop to accomplish the same task. Again, both ID and VISIT need to be sorted first, but only the ID variable is used as the by-variable in the DATA step. Reading the last observation for each patient served as the break-event for the DOW-loop. Notice that there are two conditions in the IF/THEN-ELSE statement that are needed. Program 10 didn't specify the condition when the visiting time is less than the treatment time; this is the situation where B_SBP needs to be assigned with the missing values. Since B_SBP is initialized to missing at the beginning of the DATA step and the data is sorted by ID and then by the visiting time, there is a need to take care of the condition when the visiting time is less than the treatment time.

Program 10:

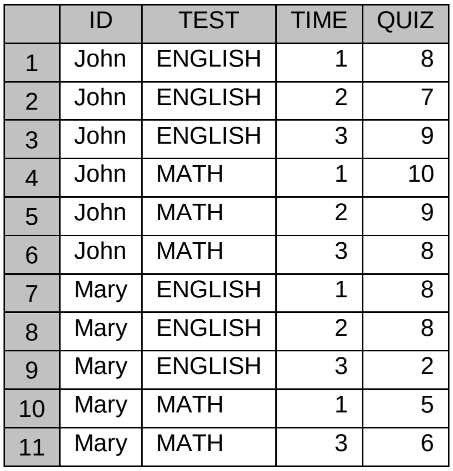
```
proc sort data=base;
    by ID visit;
run;

data p10;
    do until(last.ID);
        set base;
        by ID;
        if visit=trtmt then b_sbp = sbp;
        else if visit>trtmt then c_sbp=sbp-b_sbp;
        output;
    end;
run;
```

TRANSPOSING DATA USING DOW-LOOP

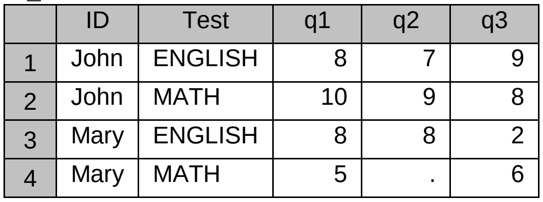
You can also utilize the DOW-loop to transpose data. More complicated examples of transposing data by using the DOW-loop can be found in Richard Allen and Nancy Brucken's paper. For example, the data set, QUIZ contains the 3 quiz scores for English and MATH for John and Mary. Notice that Mary is missing the second quiz score for math. The transposed data will look like the QUIZ_TRANS data set.

Quiz.sas7bdat



	ID	TEST	TIME	QUIZ
1	John	ENGLISH	1	8
2	John	ENGLISH	2	7
3	John	ENGLISH	3	9
4	John	MATH	1	10
5	John	MATH	2	9
6	John	MATH	3	8
7	Mary	ENGLISH	1	8
8	Mary	ENGLISH	2	8
9	Mary	ENGLISH	3	2
10	Mary	MATH	1	5
11	Mary	MATH	3	6

Quiz_trans.sas7bdat



	ID	Test	q1	q2	q3
1	John	ENGLISH	8	7	9
2	John	MATH	10	9	8
3	Mary	ENGLISH	8	8	2
4	Mary	MATH	5	.	6

Program 11 presents a solution without using the DOW-loop. In this program, Q1-Q3 variables are created by using the ARRAY statement. During DATA step iteration and depending upon the value of TIME at each iteration, Q1, Q2, or Q3 are assigned with the value from the QUIZ variable. Q1-Q3 need to be retained in the DATA step because you need to hold their values and output to the output data set until you finish reading all the quizzes for each subject. Furthermore, Q1-Q3 need to be set to missing values when starting to read the first quiz of each subject to prevent the current quiz score from being carried forward to the next observation if the next quiz score is missing.

Program 11:

```
proc sort data=quiz;
    by ID test;
run;

data p11 (drop=time quiz index);
    set Quiz;
    by id test;
    array q q1-q3;
    retain q;
    if first.test then do;
        do index = 1 to 3;
        q[index]=.;
        end;
    end;
    q[time]=Quiz;
    if last.test;
run;
```

Program 12 simplifies Program 11 by using the DOW loop. Q1-Q3 do not need to be retained in the DATA step because these three variables are populated in the DOW-loop within one single iteration of the DATA step. Since these three variables are set to missing at the beginning of the DATA step, there's no need for setting these three variables to missing explicitly.

Program 12:

```
data p12 (drop=time quiz);
     array q q1-q3;
     do until (last.test);
         set quiz;
         by id test;
         q[time]=quiz;
     end;
run;
```

COMPARING EFFICIENCY

Most of the examples that were presented in this paper use two approaches: one uses the DOW-loop, and the other one does not. To compare which program is more efficient, four sets of data were simulated and tested on a laptop with an Intel Core i5 processor and 8 GB of RAM. The real time and CPU time of each program is recorded in Table 1.

For example, to compare Program 1 and Program 2, 100 million observations are simulated. Program 2, which used the DOW loop to compute the means for each subject, seems to be slightly faster than Program 1. Program 8 and Program 12, which used DOW-loops, are also slightly faster than Programs 7 and 11 respectively. However, Program 9, which did not use the DOW loop, is faster than Program 8. In conclusion, there is no clear advantage which approach is better than the other (if one uses efficiency for comparison).

Table 1: Efficiency comparison between programs with the DOW-loop and programs without using the DOW-loop.

Number of observations in the simulated data	Programs	Using DOW-loop	Real Time	CPU Time
100,000,000	Program 1	No	11.20 sec	11.06 sec
	Program 2	Yes	11.17 sec	10.76 sec
60,002,649	Program 7	No	9.57 sec	9.40 sec
	Program 8	Yes	9.40 sec	8.92 sec
100,000,000	Program 9	No	1:28.84 min	29.84 sec
	Program 10	Yes	1:44.72 min	36.79 sec
108,000,000	Program 11	No	30.19 sec	24.96 sec
	Program 12	Yes	20.32 sec	20.32 sec

CONCLUSION

Using a DOW-loop can be used as an alternative approach to solving many real-world applications. Utilizing a DOW-loop eliminates the necessity of retaining newly-created variables that need to be carried forward explicitly in the DATA step, and/or avoids setting certain variables to missing values when reading the first observation within each by-group; consequently, using a DOW-loop simplifies the coding for some applications in the DATA step. However, there is no real advantage of using a DOW-loop for achieving an efficient program based on the programs that were presented in this paper. Furthermore, regardless of whether or not one utilizes the DOW-loop, one needs to grasp the essence of DATA step programming, such as how the data were processed during the compilation and execution phases; otherwise, programming errors are easily/unexpectedly generated.

REFERENCES

Allen, Richard Read. Practical Uses of the DOW Loop. WUSS 2009 Proceedings.
Brucken, Nancy. 2 PROC TRANSPOSE = 1 DATA Step DOW-Loop. PharmaSUG 207 Proceedings.
Chakravarthy, Venky. The DOW (not that DOW!!!) and the LOCF in Clinical Trials. SUGI 28 Proceedings.
Dorfman, Paul M. The DOW-Loop Unrolled. SESUG 2010 Proceedings
Li, Arthur. 2013. Handbook of SAS® DATA Step Programming. Chapman and Hall/CRC.

CONTACT INFORMATION

Arthur Li
City of Hope National Medical Center
Division of Information Science
1500 East Duarte Road
Duarte, CA 91010 - 3000
Work Phone: (626) 256-4673 ext. 65121
Fax: (626) 471-7106
E-mail: arthurli@coh.org