Paper HOW-08-2014

Understanding How the DATA Step Works

ABSTRACT

To be a successful DATA Step programmer, one needs to grasp how the DATA step processes data via the program data vector (PDV) during the compilation and execution phases. Understanding how and why each of the automatic or user-defined variables is initialized and retained in the PDV is essential for writing an accurate program. Among these variables, the following variables deserve special attention, including variables that are created in the DATA step by using the RETAIN or the SUM statements, and via by-group processing (FIRST.VARIABLE and LAST.VARIABLE). In this paper, you will be exposed to what happens in the PDV and how these variables are retained from various applications.

INTRODUCTION

A common befuddlement often facing beginning SAS® programmers is that the SAS data set that they create is not what they intended to create; i.e. there are more or less observations than intended or the value of the newly-created variable is not retained correctly. These types of mistakes occur because new programmers often focus exclusively on language syntax but fail to understand how the DATA step actually works. The purpose of this paper is to guide you through how DATA step programming operates, step by step, by way of providing various examples.

OVERVIEW OF DATA STEP PROCESSING

A DATA step is processed sequentially via the *compilation* and *execution* phases. In the compilation phase, each statement is scanned for syntax errors. If an error is found, SAS will stop processing. The execution phase only begins after the compilation phase ends.

In the execution phase, the DATA step works like a loop, repetitively executing statements to read data values and create observations one at a time. Each loop is called an iteration. We can refer to this type of loop as the implicit loop.

Not all SAS statements in the DATA step are executed during the execution phase. Instead, statements in the DATA step can be categorized as *executable* or *declarative*. The declarative statements are used to provide information to SAS and only take effect during the compilation phase. The declarative statements can be placed in any order within the DATA step. Here are a few examples of declarative statements: LENGTH, FORMAT, LABEL, DROP, and KEEP statements.

In contrast to declarative statements, the order in which executable statements appear in the DATA step matters greatly. For example, to read an external text file, you need to start with the INFILE statement, followed by the INPUT statement. The INFILE statement is used to identify the location of the external file and the INPUT statement instructs SAS how to read each observation. Thus, you must place the INFILE statement before the INPUT statement because SAS needs to know where to find the external file *before* it can read it. Program 1 illustrates how DATA step processing works. This program reads raw data from a text file, EXAMPLE_1.TXT, and creates one variable, BMI.

EXAMPLE_1.TXT contains two observations and three variables, NAME (column 1 – 7), HEIGHT (column 9 – 10), and WEIGHT (column 12 – 14). Notice that the WEIGHT variable for the first observation is entered as “12D”, which is a data entry error. Since each variable is occupied in a fixed field and the values for these variables are standard character or numerical values, the column input method is best used to read the raw data set.

EXAMPLE_1.TXT:

```
Barbara 61 12D
John    62 175
```

Program 1:

```
data ex1;
    infile 'W:\SAS Book\dat\example_1.txt';
    input name $ 1-7 height 9-10 weight 12-14;
    BMI = 700*weight/(height*height);
    output;
run;
```

SAS Log from Program1:

```
1    data ex1;
2        infile 'W:\SAS Book\dat\example_1.txt';
3        input name $ 1-7 height 9-10 weight 12-14;
4        BMI = 700*weight/(height*height);
5        output;
6    run;

NOTE: The infile 'W:\SAS Book\dat\example_1.txt' is:
      Filename=W:\SAS Book\dat\example_1.txt,
      RECFM=V,LRECL=256,File Size (bytes)=32,
      Last Modified=15Mar2012:09:10:03,
      Create Time=15Mar2012:09:09:22

NOTE: Invalid data for weight in line 1 12-14.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6
1         Barbara 61 12D 14
name=Barbara height=61 weight=. BMI=. _ERROR_=1 _N_=1
NOTE: 2 records were read from the infile 'W:\SAS Book\dat\example3_1.txt'.
      The minimum record length was 14.
      The maximum record length was 14.
NOTE: Missing values were generated as a result of performing an operation on missing
values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 4:14
NOTE: The data set WORK.EX1 has 2 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.15 seconds
      cpu time            0.03 seconds
```

COMPILATION PHASE

Since Program 1 reads in a raw data set, the input buffer is created at the beginning of the compilation phase. The input buffer is used to hold raw data (Figure 1). However, if you read in a SAS data set instead of a raw data file, the input buffer will not be created.

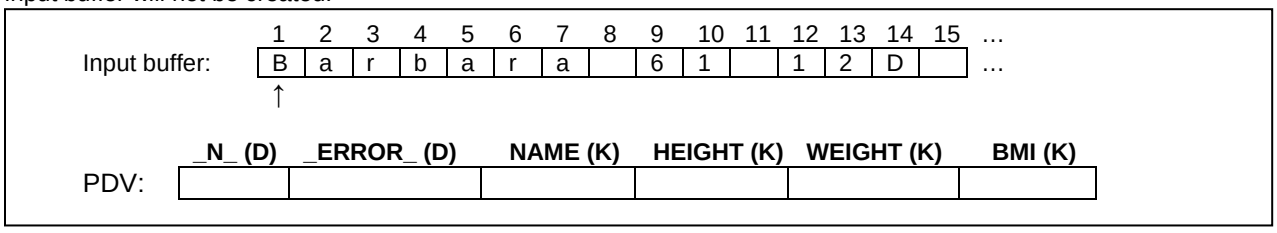

Figure1. Input buffer and the PDV.

SAS also creates the Program Data Vector (PDV) in the compilation phase (Figure 1). SAS uses the PDV, a memory area on your computer, to build the new data set. There are two automatic variables, _N_ and _ERROR_, inside the PDV. _N_ equaling 1 indicates the first observation is being processed, _N_ equaling 2 indicates the second observation is being processed, and so on. The automatic variable _ERROR_ is an indicator variable with values of 1 or 0. _ERROR_ equaling 1 signals the data error of the currently-processed observation, such as reading the data with an incorrect data type. In addition to the two automatic variables, there is one space allocated for each of the

variables that will be created from this DATA step. The variables NAME, HEIGHT and WEIGHT are read in from an external file, whereas the newly-created BMI variable is derived from HEIGHT and WEIGHT.

Notice that some of the variables in the PDV are marked with (D), which stands for "dropped," and others are marked with (K), which stands for "kept". Only the variables marked with (K) will be written to the output data set. Automatic variables, marked with a (D), are never written out.

During the compilation phase, SAS checks for syntax errors, such as invalid variable names, options, punctuations, misspelled keywords, etc. SAS also identifies the type and length of the newly-created variables. At the end of the compilation phase, the descriptor portion of the SAS data set is created, which includes the data set name, the number of observations, and the number, names, and attributes of variables.

EXECUTION PHASE

At the beginning of the execution phase, the automatic variable _N_ is initialized to 1, and _ERROR_ is initialized to 0 since there is no data error. The non-automatic variables are set to missing. Once the INFILE statement identifies the location of the input file, the INPUT statement copies the first data line into the input buffer. Then the INPUT statement reads data values from the record in the input buffer according to instructions from the INPUT statement and writes them to the PDV. The values for NAME and HEIGHT are successfully copied from the input buffer to the PDV. However, the value for WEIGHT is"12D", an invalid numeric value that causes _ERROR_ to be set to 1 and WEIGHT to missing. Meanwhile, an error message is sent to the SAS log indicating the location of the data error (see Log from Program 1). Next, the assignment statement is executed and BMI will remain missing since operations on a missing value will result in a missing value.

When the OUTPUT statement is executed, only the values from the PDV marked with (K) are copied as a single observation to the output SAS data set, EX1. See Figure 2 for a detailed explanation of each step in the first iteration.

At the end of the DATA step the SAS system returns to the beginning of the DATA step to begin the next iteration. The values of the variables in the PDV are reset to missing. The automatic variable _N_ is incremented to 2 and _ERROR_ is set to 0. The second data line is read into the input buffer by the INPUT statement. See the illustration in Figure 3 for details.

At the end of the DATA step for the second iteration, the SAS system again returns to the beginning of the DATA step to begin the next iteration (see Figure 4). The values of the variables in the PDV are reset to missing. The automatic variable _N_ is incremented to 3. SAS attempts to read an observation from the input data set, but it reaches the end-of-file-marker, which means that there are no more observations to read. When an end-of-file marker is encountered, SAS goes to the next DATA or PROC step in the program.

THE OUTPUT STATEMENT

In Program 1, an explicit OUTPUT statement is used to tell SAS to write the current observation from the PDV to a SAS data set immediately. An OUTPUT statement is not actually needed in Program 1 because the contents of the PDV are written out by default with an *implicit* OUTPUT statement at the end of the DATA.

If you place an explicit OUTPUT statement in a DATA step, the explicit OUTPUT statement will override the implicit OUTPUT statement. Once an explicit OUTPUT statement is used to write an observation to an output data set, there is no longer an implicit OUTPUT statement at the end of the DATA step. SAS adds an observation to the output data set only when an explicit OUTPUT statement is executed. Furthermore, more than one OUTPUT statement in the DATA step can be used.

THE DIFFERENCE BETWEEN READING A RAW DATA SET AND A SAS DATA SET

When creating a SAS data set based on a raw data set, SAS initializes each variable value in the PDV to missing at the beginning of each iteration of execution, except for the automatic variables, variables that are named in the RETAIN statement, variables that are created by the SUM statement, data elements in a _TEMPORARY_ array, and variables created in the options of the FILE/INFILE statement.

Often an output data set is created based on an existing SAS data set instead of reading in a raw data file. In this instance, SAS sets each variable to missing in the PDV *only* before the first iteration of the execution. Variables will keep (retain) their values in the PDV until they are replaced by the new values from the input data set. These variables exist in both the input and output data sets. Often when creating a new data set based on the existing SAS data set, you will create new variables based on existing variables; these new variables are not from the input data set. These new variables will be set to missing in the PDV at the beginning of every iteration of the execution.

FIRST ITERATION:

```
data ex1;
```

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	1	0		•	•	•

EXPLANATION: The automatic variable _N_ is initialized to 1 and _ERROR_ is initialized to 0. The non-automatic variables are set to missing.

```
infile 'W:\SAS Book\dat\example_1.txt';
```

EXPLANATION: The INFILE statement identifies the location of the input file.

```
input name $ 1-7 height 9-10 weight 12-14;
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
Input buffer:	B	a	r	b	a	r	a		6	1		1	2	D		…

↑

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	1	1	Barbara	61	•	•

EXPLANATION: The INPUT statement copies the first data line into the input buffer. SAS uses the input pointer to read data from the input buffer to the PDV. At the moment, the input pointer is positioned at the beginning of the input buffer. Next, the INPUT statement reads data values from the record in the input buffer according to instructions from the INPUT statement and writes them to the PDV. First the INPUT statement instructs SAS to read values from columns 1-7 from the input buffer and copies them to the NAME slot in the PDV. The input pointer now rests in column 8, which is immediately after the last value read. Then the INPUT statement instructs SAS to read values from columns 9-10 from the input buffer and assigns them to HEIGHT. Next, SAS attempts to read values from columns 12-14 but 12D is not a valid numeric value; this causes WEIGHT to remain missing and _ERROR_ is set to 1. Meanwhile, an error message will be sent to the SAS log indicating the location of the data error.

```
BMI = 700*weight/(height*height);
```

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	1	1	Barbara	61	•	•

EXPLANATION: The assignment statement is executed and BMI will remain missing since operations on a missing value will result in a missing value.

```
output;
```

Ex1:

NAME	HEIGHT	WEIGHT	BMI
Barbara	61	.	.

EXPLANATION: When the OUTPUT statement is executed, only values marked with (K) from the PDV are copied as a single observation to the SAS dataset EX1.

```
run;
```

EXPLANATION: The SAS system returns to the beginning of the DATA step to begin the next iteration (see Figure 3)

Figure 2. The first iteration of Program 1.

SECOND ITERATION:

```
data ex1;
```

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	2	0		•	•	•

EXPLANATION: The automatic variable _N_ is initialized to 2 and _ERROR_ is initialized to 0 since there is no data error. The non-automatic variables are set to missing.

```
infile 'W:\SAS_Book\dat\example_1.txt';
input name $ 1-7 height 9-10 weight 12-14;
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
Input buffer:	J	o	h	n					6	2		1	7	5		…
	↑															

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	2	0	John	62	175	•

EXPLANATION: The INPUT statement copies the second line of data into the input buffer. The input pointer is positioned at the beginning of the input buffer. Next, the INPUT statement reads the data values from the input buffer to the PDV. There is no data error for this observation.

```
BMI = 700*weight/(height*height);
```

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	2	0	John	62	175	31.86785

EXPLANATION: BMI is calculated.

```
output;
```

Ex1:

NAME	HEIGHT	WEIGHT	BMI
Barbara	61	.	.
John	62	175	31.8678

EXPLANATION: The second observation is created.

```
run;
```

EXPLANATION: The SAS system returns to the beginning of the DATA step to begin the next iteration (see Figure 4)

Figure 3. The second iteration of Program 1.

THIRD ITERATION:

```
data ex1;
```

	_N_ (D)	_ERROR_ (D)	NAME (K)	HEIGHT (K)	WEIGHT (K)	BMI (K)
PDV:	3	0		•	•	•

EXPLANATION: The automatic variable _N_ is initialized to 3 and _ERROR_ is initialized to 0. The non-automatic variables are set to missing. SAS reaches the end-of-file-marker, which means there are no more observations to read. SAS goes to the next DATA or PROC step.

Figure 4. The third iteration of Program 1.

SITUATIONS FOR RETAINING THE VALUES OF NEWLY-CREATED VARIABLES

In some situations, you may want to retain the values from the newly-created variables in the PDV throughout the execution of the DATA step. To prevent the newly-created variables from being initialized to *missing* at the beginning of each iteration of the implicit loop, you can use the RETAIN statement.

THE RETAIN STATEMENT

Suppose you would like to create a new variable that is based on values from previous observations, such as creating a variable that accumulates the values from other numeric variables. Consider the SAS data set, SAS_1. Based on SAS_1, you would like to create a new variable, TOTAL, that is used to accumulate the SCORE variable.

SAS_1:

	ID	SCORE
1	A01	3
2	A02	.
3	A03	4

In order to create an accumulator variable, TOTAL, you need to initialize TOTAL to 0 at the first iteration of the execution. Then at each successive iteration of the execution, add the value from the SCORE variable to the TOTAL variable. Since TOTAL is a new variable that you want to create, TOTAL will be set to missing in the PDV at the beginning of every iteration of the execution. Thus, in order to accumulate the TOTAL variable, you need to retain the value of TOTAL at the beginning of each iteration of the execution. In this situation, you need to use the RETAIN statement. The RETAIN statement has the following form:

RETAIN *variable* <*value*>*;*

In the RETAIN statement, *variable* is the name of the variable that you will want to retain and *value* is a numeric value that is used to initialize the *variable* only at the first iteration of the DATA step execution. If you do not specify an initial value, the retained variable is initialized as missing before the first execution of the DATA step. The RETAIN statement prevents the *variable* from being initialized each time the DATA step executes. Program 2 uses the RETAIN statement to create the TOTAL variable.

Program 2:

```
data ex2;
    set sas_1;
    retain total 0;
    total = sum(total, score);
run;

title 'Creating TOTAL by accumulating SCORE';
proc print data=ex2;
run;
```

Output from Program 2:

```
              Creating TOTAL by accumulating SCORE

                  Obs    ID     score    total

                   1     A01      3        3
                   2     A02      .        3
                   3     A03      4        7
```

Next example also illustrates the use of the RETAIN statement. 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. In Program 3, 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:

	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 from Program 3:

Obs	ID	score
1	A	3
2	B	4
3	C	4
4	D	5
5	E	5
6	F	5

THE SUM STATEMENT

Program 2 can be re-written by using the SUM statement instead of using the RETAIN statement. The SUM statement has the following form:

variable+*expression*;

In the SUM statement, *variable* is the numeric accumulator variable that is to be created and is automatically set to 0 at the beginning of the first iteration of the DATA step execution (and is thus retained in following iterations). *Expression* is any SAS expression. In a situation where e*xpression* is evaluated to a missing value, it is treated as 0. Program 4 is an equivalent version of Program 2 using the SUM statement.

Program 4:

```
data ex3;
    set sas_1;
    total + score;
run;
```

CONDITIONAL PROCESSING

Many applications require the DATA step to process only part of the observations that meet the condition of a specified expression. In this situation, you need to use the subsetting IF statement. The subsetting IF statement has the following form:

IF *expression*;

The *expression* associated with the subsetting IF statement can be any valid SAS expression. If the *expression* is

true for the observation, SAS continues to execute statements in the DATA step and includes the current observation in the data set. The resulting SAS data set contains a subset of the external file or SAS data set. On the other hand, if the *expression* is false, then no further statements are processed for that observation and SAS immediately returns to the beginning of the DATA step. For example, Program 5 creates a data set that only contains the observations in which the SCORE variable is not missing.

Program 5:
```
data ex4;
    set sas_1;
    total + score;
    if not missing(score);
run;

title 'Keep observations only when SCORE is not missing';
proc print data=ex4;
run;
```

Output from Program 5:

Keeping observations for SCORE is not missing

Obs	ID	score	total
1	A01	3	3
2	A03	4	7

Sometimes it is more efficient to specify a condition for excluding observations instead of including observations in the output data set. In this situation, you can combine the subsetting IF statement with the DELETE statement.

IF *expression* **THEN DELETE**;

Program 6 provides an alternative version of Program 5 by utilizing the subsetting IF and DELETE statements.

Program 6:
```
data ex5;
    set sas_1;
    total + score;
    if missing(score) then delete;
run;
```

BY-GROUP PROCESSING

The examples above used a data set that only contains one observation per subject. Sometimes you will also work with data with multiple observations per subject. This type of data often results from repeated measures for each subject and is often called longitudinal data.

Applications that involve longitudinal data often require identifying the beginning or end of measurement for each subject. This can be accomplished by using the BY-group processing method. *BY-group processing* is a method of processing records from data sets that can be grouped by the values of one or more common variables. These "grouping" variables are called the *BY variable*. The value of a BY variable is called the *BY value*. A *BY group* refers to all observations with the same BY value. When there are multiple variables designated as BY variables, a BY group would be a group of records with the same combination of the values of these BY variables with each BY group containing a unique combination of values for the BY variables. During BY-group processing, SAS creates two automatic variables, FIRST.VARIABLE and LAST.VARIABLE, which are used to indicate the beginning or the end of the measurement within each BY group.

THE FIRST.VARIABLE AND THE LAST.VARIABLE

In order to utilize BY-group processing, you need to place a BY statement with one or more BY variable(s) after the SET statement. Furthermore, the input data set also needs to be previously sorted by the BY variable(s).

During BY-group processing, SAS creates two temporary indicator variables for each BY variable: FIRST.VARIABLE and LAST.VARIABLE. Since FIRST.VARIABLE and LAST.VARIABLE are temporary variables, they are not sent to

the output data set. Both FIRST.VARIABLE and LAST.VARIABLE are initialized to 1 at the beginning of the DATA step execution. Then FIRST.VARIABLE is set to 1 in the PDV when SAS reads the first observation in each BY group and is set to 0 when reading the second to the last observation in each BY group. Similarly, LAST.VARIABLE is set to 1 when reading the last observation in each BY group and set to 0 when reading those observations that are not last.

Consider the SAS data set, SAS_2, which consists of five observations with the values of SCORE for two subjects, A01 and A02.

SAS_2:

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

Suppose that the ID variable is the representative BY variable; consequently, there will be two BY groups because there are two distinct values for the ID variable. FIRST.ID and LAST.ID will be created and represented as. If you use ID and SCORE as the BY variables, then in addition to FIRST.ID and LAST.ID, FIRST.SCORE and LAST.SCORE will be created in the PDV. There will be four BY groups based on unique combination values of ID and SCORE (See Figure 5).

	ID	SCORE	*GROUP: ID*	FIRST.ID	LAST.ID	*GROUP: ID & SCORE*	FIRST.SCORE	LAST.SCORE
1	A01	3	1	1	0	1	1	0
2	A01	3		0	0		0	1
3	A01	2		0	1	2	1	1
4	A02	2	2	1	0	3	1	1
5	A02	4		0	1	4	1	1

Figure 5. This figure illustrates the values for automatic variables FIRST.ID, LAST.ID, FIRST.SCORE and LAST.SCORE.

THE EXECUTION PHASE OF BY-GROUP PROCESSING

Suppose that you would like to calculate the total scores for each subject in the SAS_2 data set. To create a variable (TOTAL) that is the total score for each subject, you need to initialize TOTAL to 0 when starting to read the first observation of each subject. Then TOTAL can be accumulated by adding the value from the SCORE variable to TOTAL for each observation. In the end, you can output the TOTAL score when reading the last observation of each subject. Therefore, you need to utilize BY-group processing and use ID as the BY variable. The solution for this problem is shown in Program 7.

Program 7:

```
proc sort data=sas_2;
    by id;
run;

data ex6(drop=score);
    set sas_2;
    by id;
    if first.id then total = 0;
    total + score;
    if last.id;
run;

title 'The total scores for each subject';
proc print data=ex6;
run;
```

Output from Program 7:

```
The total scores for each subject

       Obs    ID     total

        1     A01      8
        2     A02      6
```

SOME APPLICATIONS UTILIZING BY-GROUP PROCESSING

Longitudinal data does not always require BY-group processing when it is read into a SAS data set. Instead, a program that uses BY-group processing does so to identify either the beginning and/or the end of a measurement within a BY group. Therefore, a DATA step that uses by-group processing frequently contains the following:

1. A cumulating variable is initialized to 0 when the FIRST.VARIABLE equals 1.
2. A cumulating variable is accumulated with some values at every iteration of the DATA step.
3. Some calculation needs to be performed when the LAST.VARIABLE equals 1.
4. The contents of the PDV are outputted only when the LAST.VARIABLE equals 1.
5. In addition to the BY variable, an additional variable will also need to be previously sorted. However, only the BY variable is used in the SET statement in the DATA step.

Most applications don't complete all five steps in the list above. For example, Program 6 only completes three:

- TOTAL is initialized to 0 when FIRST.ID equals 1 (# 1)
- TOTAL accumulates its value from the SCORE variable at every iteration of the DATA step (# 2)
- At the end of the DATA step, the contents from the PDV are written to the output data set when LAST.ID equals 1 (# 4)

CALCULATING MEAN SCORE WITHIN EACH BY GROUP

Suppose that you would like to calculate the mean score for each person based on the data set SAS_2. The solution for this problem is similar to the one in Program 7. Even though you are calculating the means instead of the total score, you still need to accumulate all the scores for each subject (TOTAL) and create a "counter" variable (N) to count the number of observations within each BY group. Furthermore, TOTAL and N need to be initialized to 0 when FIRST.ID equals 1. Then you will need to calculate the mean score (MEAN_SCORE) by dividing TOTAL by N and output the result when LAST.ID equals 1 (See Program 8).

Program 8:

```
data ex7 (drop=score);
    set sas_2;
    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;

title 'The mean score for each subject';
proc print data=ex7;
run;
```

Output from Program 8:

The mean score for each subject

Obs	ID	total	n	mean_ score
1	A01	8	3	2.66667
2	A02	6	2	3.00000

OBTAINING THE MOST RECENT NON-MISSING DATA WITHIN EACH BY-GROUP

Longitudinal data, such as patients with repeated measurements over time, are often encountered in the clinical field. For example, a patient may have multiple measurements of weight, blood pressure, total cholesterol, or blood glucose level over several medical visits, but may not have all these values recorded every time. Researchers may be interested in a database depicting the most recent available information on their patients.

For example, the data set SAS_3 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 variable in the resulting data set for this variable. For instance, the TGL measurement is missing for A03.

SAS_3:

	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	

Since you only need to keep the most recent record, the data set has to be sorted by the PATID and VISIT variables in ascending order. However, when utilizing BY-group processing in the DATA step, you only need to use PATID as the BY variable. One idea for solving this problem is that you initially assign TGL_NEW and SMOKE_NEW to missing values when reading the first observation for each patient from the sorted data set. At each iteration of the DATA step, you will assign the values from the TGL and SMOKE variables to TGL_NEW and SMOKE_NEW (respectively) provided that TGL and SMOKE are not missing. Then you will output the values in the PDV when reading the last observation of each patient. Since TGL_NEW and SMOKE_NEW are newly-created variables, you need to retain their values by using the RETAIN statement. The solution for this problem is illustrated in Program 9.

Program 9:

```
proc sort data=sas_3;
    by patid visit;
run;

data ex8 (drop= visit tgl smoke);
    set sas_3;
    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;

title 'The data contains the most current non-missing values';
proc print data=ex8;
run;
```

Output from Program 9:

The data contains the most current non-missing values

Obs	patid	tgl_new	smoke_new
1	A01	150	Y
2	A02	210	N
3	A03	.	Y
4	A04	190	N
5	A05	189	

USING THE PUT STATEMENT TO OBSERVE THE CONTENTS OF THE PDV

Programming errors are can be categorized as either syntax or logic errors. Syntax errors are often easier to detect than logic errors since SAS not only stops programs due to syntax errors but also generates detailed error messages in the log window. On the other hand, logic errors often result in generating an unintended data set and they are difficult to debug. One way to detect a logic error is to use the PUT statement in the DATA step, which can be used to generate the contents of each variable in the PDV on the SAS log. The PUT statement has the following form:

PUT *variable* | *variable-list* | *character-string*;

The PUT statement can combine explanatory text strings in quotes with the values for selected variables and write the compound output to the SAS log. When using the keyword _ALL_, the PUT statement will output the contents from all the variables, including the automatic variables, to the SAS log. For example, Program 10 uses the PUT statement to send the contents in the PDV to the SAS log after each statement is executed in the DATA step.

Program 10:

```
data ex9;
    put "1st PUT" _all_;
    set sas_1;
    put "2nd PUT" _all_;
    total + score;
    put "3rd PUT" _all_;
    if not missing(score);
    put "4th PUT" _all_;
run;
```

Log from Program 10:

```
68   data ex9;
69       put "1st PUT" _all_;
70       set sas_1;
71       put "2nd PUT" _all_;
72       total + score;
73       put "3rd PUT" _all_;
74       if not missing(score);
75       put "4th PUT" _all_;
76   run;

1st PUTID=  SCORE=. total=0 _ERROR_=0 _N_=1
2nd PUTID=A01 SCORE=3 total=0 _ERROR_=0 _N_=1
3rd PUTID=A01 SCORE=3 total=3 _ERROR_=0 _N_=1
4th PUTID=A01 SCORE=3 total=3 _ERROR_=0 _N_=1
1st PUTID=A01 SCORE=3 total=3 _ERROR_=0 _N_=2
2nd PUTID=A02 SCORE=. total=3 _ERROR_=0 _N_=2
3rd PUTID=A02 SCORE=. total=3 _ERROR_=0 _N_=2
1st PUTID=A02 SCORE=. total=3 _ERROR_=0 _N_=3
2nd PUTID=A03 SCORE=4 total=3 _ERROR_=0 _N_=3
3rd PUTID=A03 SCORE=4 total=7 _ERROR_=0 _N_=3
4th PUTID=A03 SCORE=4 total=7 _ERROR_=0 _N_=3
1st PUTID=A03 SCORE=4 total=7 _ERROR_=0 _N_=4
NOTE: There were 3 observations read from the data set WORK.SAS3_1.
NOTE: The data set WORK.EX3_4 has 2 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds
```

CONCLUSION

To be a successful SAS programmer, you must be able to thoroughly comprehend how DATA steps are processed. The most important part of DATA step processing is to understand how data is transformed to the PDV and how data is copied from the PDV to a new dataset.

REFERENCES

Li, Arthur. 2013. Handbook of SAS® DATA Step Programming. Chapman and Hall/CRC.

CONTACT INFORMATION

Arthur X. 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