Paper S1-01-2013

The Essence of DATA Step Programming

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

ABSTRACT

The fundamental of SAS® programming is DATA step programming. The essence of DATA step programming is to understand how SAS processes the data during the compilation and execution phases. In this paper, you will be exposed to what happens "behind the scenes" while creating a SAS dataset. You will learn how a new dataset is created, one observation at a time, from either a raw text file or an existing SAS dataset, to the program data vector (PDV) and from the PDV to the newly-created SAS dataset. Once you fully understand DATA step processing, learning the SUM and RETAIN statements will become easier to grasp. Relating to this topic, this paper will also cover BY-group processing.

INTRODUCTION

A common befuddlement often facing beginning SAS programmers is that the SAS dataset 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 was not retained correctly. These types of mistakes are most commonly committed because programming novices learn SAS language syntax without understanding the fundamental SAS programming concepts. The purpose of this paper is to guide you through how DATA step programming operates, step by step, by way of providing various examples.

DATA STEP PROCESSING OVERVIEW

A DATA step is processed in two-phase sequences: 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. Both phases do not occur simultaneously.

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, which is different from the explicit loop, by using FOR or WHILE statements.

In Program 1 (below), you will see how DATA step processing works. Program 1 reads the raw data from a text file, *example1.txt*. There are two observations and three variables in this dataset, 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 dataset. You will also notice that a new variable, BMI, is created in this program.

Example1.txt:

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

<u>Program 1:</u>

```
data ex1;
   infile 'C:\Users\Arthur\Documents\WUSS Proposal\Forms\example1.txt';
   input name $ 1-7 height 9-10 weight 12-14;
   BMI = 700*weight/(height*height);
   output;
run;
```

COMPILATION PHASE

Since Program 1 reads raw datasets, 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 a SAS dataset instead of a raw dataset, the input buffer will not be created.

SAS also creates the PDV in the compilation phase (Figure 1). SAS uses the PDV, a memory area on your computer, to build the new dataset. 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. HEIGHT and WEIGHT are the variables that are read from the external raw dataset. BMI is the variable that is created based on HEIGHT and WEIGHT.

Notice that some of the variables in the PDV are marked with (D), which stands for "dropped," and some of the variables are marked with (K), which stands for "kept". The variables marked with (K) will be written to the output dataset; on the other hand, the variables marked with (D) will not. All the automatic variables will not be written to the output dataset.

During the compilation phase, SAS also checks for syntax errors, such as invalid variable names, options, punctuations, misspelled keywords, etc.

Once the compilation is finished, the descriptor portion of the SAS dataset is created, which includes dataset name, the number of observations, and the number, names, and attributes of variables.

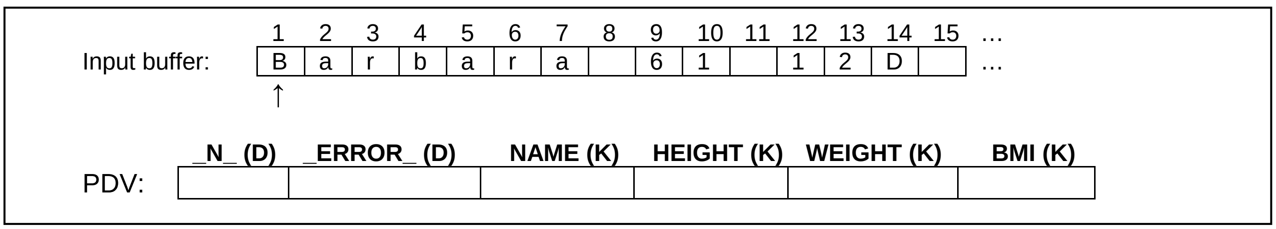

Figure1. Input buffer and the PDV.

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 first data line is read 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. When the OUTPUT statement is executed, the values from the PDV are copied as a single observation to the SAS dataset *ex1*, but the values from the automatic variables and the variables that are marked (D) are not outputted. Please see Figure 2a to see a detailed explanation of the process of each step for 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. See the illustration in Figure 2b for details.

At the end of the DATA step of the second iteration, the SAS system again returns to the beginning of the DATA step to begin the next iteration (see Figure 2c). The values of the variables in the PDV are reset to missing. The automatic variable _N_ is incremented to 3. Since there are no more records to read, the SAS system goes to the next DATA or PROC step.

THE OUTPUT STATEMENT

In Program 1, the explicit OUTPUT statement was used, which tells SAS to write the current observation from the PDV to a SAS dataset immediately, not at the end of the DATA step. Without using the explicit OUTPUT statement, by default, every DATA step contains an implicit OUTPUT statement at the end of the DATA step that tells the SAS system to write observations to the dataset. However, placing an explicit OUTPUT statement in a DATA step overrides the implicit output; in other words, the SAS system adds an observation to a dataset only when an explicit OUTPUT statement is executed. Once an explicit OUTPUT statement is used to write an observation to a dataset, there is no longer an implicit OUTPUT statement at the end of the DATA step. Furthermore, more than one OUTPUT statement in the DATA step can be used. You will see an example of using multiple OUTPUT statements later in this paper.

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 'C:\Users\Arthur\Documents\WUSS Proposal\Forms\example1.txt';
```

	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		…
	↑															

EXPLANATION: The INFILE statement identifies the location of the input file and the first data line is read 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.

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

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

EXPLANATION: 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 2b)

Figure 2a. The first iteration of Program 1.

SECOND ITERATION:

	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		…
	↑															

```
data ex1;
```

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

EXPLANATION: The second line of data is read into the input buffer. The input pointer is positioned at the beginning of the input buffer. 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 'C:\Users\Arthur\Documents\WUSS Proposal\Forms\example1.txt';
input name $ 1-7 height 9-10 weight 12-14;
```

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

EXPLANATION: The INPUT statement reads the data values from the record in 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	3.09573

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 2c)

Figure 2b. 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. Since there are no more observations to read, the SAS system goes to the next DATA or PROC step.

Figure 2c. The third iteration of Program 1.

THE DIFFERENCE BETWEEN READING A RAW DATASET AND READING A SAS DATASET

When creating a SAS dataset based on a raw dataset, SAS sets 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 created by the SUM statement, data elements in a _TEMPORARY_ array, and variables created in the options of the FILE/INFILE statement.

The dataset that is being created can be referred to as an *output* dataset. The output dataset is the one that follows the keyword DATA. Often the output dataset is created based on the existing SAS dataset instead of from the raw dataset. You can also call the existing SAS dataset that is used to create the output dataset the *input* dataset. The input dataset is the one that follows the keyword SET. When creating a SAS dataset based on an input SAS dataset, 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 dataset. These variables exist in both the input and output datasets. Often when creating a new dataset based on the existing SAS dataset, you will create new variables based on existing variables; these new variables are not from the input dataset. These new variables will be set to missing in the PDV at the beginning of *every* iteration of the execution.

THE RETAIN AND SUM STATEMENTS

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 following SAS dataset, *ex2*[1]. Based on *ex2*, suppose you would like to create a new variable, TOTAL, that is used to accumulate the SCORE variable.

Ex2:

	ID	SCORE
1	A01	3
2	A02	.
3	A03	4

In order to create an accumulator variable, TOTAL, you need to set the TOTAL to 0 at the first iteration of the execution. Then at each 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. 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

The RETAIN statement has the following form:

RETAIN *VARIABLE <VALUE>;*

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. Here is program to create the TOTAL variable by using the RETAIN statement.

Program 3:

```
data ex2_2;
    set ex2;
    retain total 0;
    total = sum(total, score);
run;
```

The execution phase begins immediately after the completion of the compilation phase. At the beginning of the execution phase, the variables ID and SCORE are set to missing (see Figure 3a); however, the variable TOTAL is initialized to 0 because of the RETAIN statement. Next, the SET statement copies the first observation from the dataset *ex2* to the PDV. The RETAIN statement is a compile-time only statement; it does not execute during the execution phase. Then the variable TOTAL is calculated. Finally, the DATA step execution reaches the final step. Since there is no explicit OUTPUT statement in this program, the implicit OUTPUT statement at the end of the DATA

---

[1] A SAS file has an extension of “sas7bdat”, for example, ex2.sas7bdat. I will not write the extension in this paper for convenience purposes

step tells the SAS system to write observations to the dataset. The SAS system returns to the beginning of the DATA step to begin the second iteration.

FIRST ITERATION:

Ex2:

ID	SCORE
A01	3
A02	.
A03	4

← Reading

```
data ex2_2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	1	0		•	0

EXPLANATION: _N_ is initialized to 1 and _ERROR_ is initialized to 0. The variables ID and SCORE are set to missing; TOTAL is initialized to 0 because of the RETAIN statement.

```
set ex2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	1	0	A01	3	0

EXPLANATION: The SET statement copies the first observation from *ex2* to the PDV.

```
retain total 0;
```

EXPLANATION: The RETAIN statement does not execute during the execution phase.

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

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	1	0	A01	3	3

EXPLANATION: TOTAL is calculated.

```
run;
```

Ex2_2:

ID	SCORE	TOTAL
A01	3	3

EXPLANATION: The implicit OUTPUT statement at the end of the DATA step tells the SAS system to write observations to the dataset. The SAS system returns to the beginning of the DATA step to begin the 2nd iteration (see Figure 3b)

Figure 3a. The first iteration of Program 3.

At the beginning of the second iteration, since data is read from an existing SAS dataset, instead of reading from the raw dataset, values in the PDV for variables ID and SCORE are retained from the previous iteration. The newly created variable TOTAL is also retained because the RETAIN statement is used. See Figures 3b and 3c for the process of the second and third iterations.

SECOND ITERATION:

Ex2:

ID	SCORE
A01	3
A02	.
A03	4

← Reading

```
data ex2_2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	2	0	A01	3	3

EXPLANATION: _N_ is incremented to 2. ID and SCORE are retained from the previous iteration because data are read from an existing SAS dataset. TOTAL is also retained because the RETAIN statement is used.

```
set ex2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	2	0	A02	.	3

EXPLANATION: The SET statement copies the second observation from *ex2* to the PDV.

```
retain total 0;
total = sum(total, score);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	2	0	A02	.	3

EXPLANATION: TOTAL is calculated. When the SUM function is used instead of using the "+" sign, the missing value is treated as 0.

```
run;
```

Ex2_2:

ID	SCORE	TOTAL
A01	3	3
A02	.	3

EXPLANATION: The implicit OUTPUT statement tells the SAS system to write observations to the dataset. The SAS system returns to the beginning of the DATA step to begin the 3rd iteration (see Figure 3c)

Figure 3b. The second iteration of Program 3.

THE SUM STATEMENT

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

VARIABLE + EXPRESSION;

The SUM statement may seem unusual because it does not contain the equal sign. VARIABLE is the numeric accumulator variable that is to be created. VARIABLE is automatically set to 0 at the beginning of the first iteration of the DATA step execution and it is retained in following iterations. EXPRESSION is any SAS expression. In the situation where EXPRESSION is evaluated to a missing value, it is treated as 0. Here is an equivalent version of Program 3 using the SUM statement.

Program 4:
```
data ex2_3;
    set ex2;
    total + score;
run;
```

TRIRD ITERATION:

Ex2:

ID	SCORE
A01	3
A02	.
A03	4

← Reading

```
data ex2_2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	3	0	A02	.	3

EXPLANATION: _N_ is incremented to 3. ID and SCORE are retained from the previous iteration. TOTAL is also retained.

```
set ex2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	3	0	A03	4	3

EXPLANATION: The SET statement copies the third observation from *ex2* to the PDV.

```
retain total 0;
total = sum(total, score);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	SCORE (K)	TOTAL (K)
PDV:	3	0	A03	4	7

EXPLANATION: TOTAL is calculated.

```
run;
```

Ex2_2:

ID	SCORE	TOTAL
A01	3	3
A02	.	3
A03	4	7

EXPLANATION: The observation from the PDV is written to the dataset. The SAS system returns to the beginning of the DATA step to begin the 4th iteration.

In the 4th and final iteration, there are no more observations to read; the SAS system goes to the next DATA or PROC step.

Figure 3c. The third iteration of Program 3.

THE SUBSETTING IF STATEMENT

You can use the IF statement to continue processing only the observations that meet the condition of the specified expression. The IF statement has the following form:

IF *EXPRESSION;*

The EXPRESSION in the IF statement can be any 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. That is to say, the remaining program statements in the DATA step are not executed and the current observation is not written to the output data set. An example will be shown in the following section.

THE BY-GROUP PROCESSING IN THE DATA STEP

The examples that have been presented so far only contain one observation per subject. Sometimes you will also deal 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. For longitudinal data, sometimes it is useful to be able to identify the beginning or end of measurement for each subject. This can be accomplished by using the BY-group processing method. SAS locates the beginning and end of a BY-group by creating two temporary indicator variables for each BY variable: FIRST.VARIABLE and LAST.VARIABLE. For example, suppose that the ID variable is the BY variable; FIRST.ID and LAST.ID will be created. When FIRST.ID equals 1, SAS reads the first observation in an ID group. Similarly, LAST.ID equals 1 indicates SAS reads the last observation in an ID group.

Since FIRST.VARIABLE and LAST.VARIABLE are temporary variables, they are not being output to the output dataset. Ron Cody's book Longitudinal Data and SAS® – A Programmer's Guide provides practical examples for managing and/or manipulating this type of data. Some of the examples in this paper are adapted from his book.

Consider the following SAS dataset, *ex3*. Each subject has multiple numbers of observations. The number of observations for each subject differs. Suppose that you would like to calculate the total scores for each subject; see Program 5 below. In this program, the SUM statement is used to accumulate the total score. For this problem, the accumulation is completed for each subject, not for all of the subjects combined. Solving this problem can be achieved in three steps: initializing the TOTAL to 0 when starting to read the first observation of each subject; creating TOTAL by accumulating SCORE for each subject; outputting the TOTAL score when reading the last observation of each subject. Therefore, utilize the BY-group processing and the ID will be the BY variable. To use the BY-group processing in the DATA step, It is important to sort the data by the BY variable (ID) first.

Ex3:

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

<u>Program 5:</u>

```
proc sort data=ex3;
    by id;
run;
data ex3_1 (drop=score);
    set ex3;
    by id;
    if first.id = 1 then total = 0;
    total + score;
    if last.id = 1;
run;
```

Since the BY statement was used after the SET statement in the DATA step, two automatic variables, FIRST.ID and LAST.ID, are created in the PDV. Both FIRST.ID and LAST.ID are initialized to 1 before the fist iteration of the DATA step execution (see Figure 4a). ID and SCORE variables are set to missing, but TOTAL is set to 0 since TOTAL is created by the SUM statement. When the SET statement executes, SAS copies the first observation from the sorted *ex3* to the PDV. Since this is the first observation for the A01 subject, FIRST.ID is set to 1. The LAST.ID is set to 0 since this is not the last observation. Next, TOTAL is assigned to 0 because this is the first observation for ID A01 (SAS statement: **`if first.id = 1 then total = 0`**). The SUM statement accumulates the TOTAL variable. Because the subsetting IF statement is evaluated to be false (LAST.ID does not equal 1), SAS immediately returns to the beginning of the DATA step. That means the contents of the PDV are not output to the SAS dataset *ex3_1.*

The second iteration (see Figure 4b) is similar to the first iteration. During this iteration, both FIRST.ID and LAST.ID are set to 0. TOTAL is then accumulated. But the PDV contents are not outputted to the SAS dataset either since this is not the last observation for A01.

In the third iteration (see Figure 4c), FIRST.ID is set to 0 and LAST.ID is set to 1. TOTAL is accumulated. The subsetting IF statement is evaluated to be true. Then SAS reaches the end of the DATA step and the implicit OUTPUT statement copies the contents from the PDV (ID and TOTAL) to the SAS dataset *ex3_1* (SCORE variable is dropped in the DATA statement). The rest of the iterations are similar to the iterations explained above. See Figures 4d- 4e for details.

FIRST ITERATION:

Ex3:

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

← Reading

```
data ex3_1;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	1	0	1	1		•	0

EXPLANATION: Both FIRST.ID and LAST.ID are initialized to 1. The ID and SCORE variables are set to missing, but TOTAL is set to 0 since TOTAL is created by the SUM statement.

```
set ex3;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	1	0	1	0	A01	3	0

EXPLANATION: SAS copies the first observation from the **sorted** *ex3* to the PDV. Since this is the first observation for the A01 subject, FIRST.ID is set to 1. The LAST.ID is set to 0 since this is not the last observation for A01.

```
by id;
if first.id = 1 then total = 0;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	1	0	1	0	A01	3	0

EXPLANATION: TOTAL is assigned to 0.

```
total + score;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	1	0	1	0	A01	3	3

EXPLANATION: TOTAL is accumulated.

```
if last.id = 1;
```

EXPLANATION: The subsetting IF statement is evaluated to be FALSE because LAST.ID ≠ 1. SAS returns to the beginning of the DATA step to begin the 2^{nd} iteration (see Figure 4b)

Figure 4a. First iteration for Program 5.

SECOND ITERATION:

Ex3:

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

← Reading

```
data ex3_1;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	2	0	1	0	A01	3	3

EXPLANATION: _N_ is incremented to 2. The values for the rest of the variables are retained.

```
set ex3;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	2	0	0	0	A01	4	3

EXPLANATION: SAS copies the second observation from the **sorted** *ex3* to the PDV. Since this is not the first observation or the last observation for A01, both FIRST.ID and LAST.ID are set to 0.

```
by id;
if first.id = 1 then total = 0;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	2	0	0	0	A01	4	3

EXPLANATION: FIRST.ID ≠ 1, so there is no execution.

```
total + score;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	2	0	0	0	A01	4	7

EXPLANATION: TOTAL is calculated

```
if last.id = 1;
```

EXPLANATION: The subsetting IF statement is evaluated to be FALSE because LAST.ID ≠ 1. SAS returns to the beginning of the DATA step to begin the 3rd iteration (see Figure 4c)

Figure 4b. Second iteration for Program 5.

THIRD ITERATION:

Ex3:

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

← Reading

```
data ex3_1;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	3	0	0	0	A01	4	7

EXPLANATION: _N_ is incremented to 3. The values for the rest of the variables are retained.

```
set ex3;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	3	0	0	1	A01	2	7

EXPLANATION: SAS copies the third observation from the **sorted** *ex3* to the PDV. Since this is not the first observation, FIRST.ID is set to 0. However, LAST.ID is set to 1 since this is the last observation for A01.

```
by id;
if first.id = 1 then total = 0;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	3	0	0	1	A01	2	7

EXPLANATION: FIRST.ID ≠ 1, so there is no execution.

```
total + score;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	3	0	0	1	A01	2	9

EXPLANATION: TOTAL is calculated.

```
if last.id = 1;
```

EXPLANATION: The subsetting IF statement is evaluated to be TRUE since LAST.ID equals 1.

```
run;
```

Ex3_1:

ID	TOTAL
A01	9

EXPLANATION: SAS reaches the end of the 3rd iteration. The implicit OUTPUT statement copies ID and TOTAL from the PDV to the dataset *ex3_1.* The remaining variables are not copied since they are marked with (D). The SAS system returns to the beginning of the DATA step to begin the 4th iteration (see Figure 4d)

Figure 4c. Third iteration for Program 5.

FOURTH ITERATION:

Ex3:

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

← Reading (A02, 4)

```
data ex3_1;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	0	0	1	A01	2	9

EXPLANATION: _N_ is incremented to 4. The values for the remaining variables are retained.

```
set ex3;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	0	1	0	A02	4	9

EXPLANATION: SAS copies the fourth observation from the **sorted** *ex3* to the PDV. FIRST.ID is set to 1 since this is the first observation for A02. However, LAST.ID is set to 0.

```
by id;
if first.id = 1 then total = 0;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	0	1	0	A02	4	0

EXPLANATION: Since FIRST.ID = 1, TOTAL is set to 0.

```
total + score;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	0	1	0	A02	4	4

EXPLANATION: TOTAL is calculated.

```
if last.id = 1;
```

EXPLANATION: The subsetting IF statement is evaluated to be FALSE because LAST.ID ≠ 1. SAS returns to the beginning of the DATA step to begin the 5th iteration (see Figure 4e)

Figure 4d. Fourth iteration for Program 5.

FIFTH ITERATION:

Ex3:

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

← Reading

```
data ex3_1;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	5	0	1	0	A02	4	4

EXPLANATION: _N_ is incremented to 5. The values for the remaining variables are retained.

```
set ex3;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	5	0	0	1	A02	2	4

EXPLANATION: SAS copies the fifth observation from the **sorted** *ex3* to the PDV. FIRST.ID is set to 0. LAST.ID is set to 1 since this is the last observation for A02.

```
by id;
if first.id = 1 then total = 0;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	5	0	0	1	A02	2	4

EXPLANATION: FIRST.ID ≠ 1, so there is no execution.

```
total + score;
```

	_N_ (D)	_ERROR_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	5	0	0	1	A02	2	6

EXPLANATION: TOTAL is calculated.

```
if last.id = 1;
```

EXPLANATION: The subsetting IF statement is evaluated to be TRUE since LAST.ID equals 1.

```
run;
```

Ex3_1:

ID	TOTAL
A01	9
A02	6

EXPLANATION: SAS reaches the end of the 5th iteration. The implicit OUTPUT statement copies ID and TOTAL from the PDV to the dataset *ex3_1.* The SAS system returns to the beginning of the DATA step to begin the 6th iteration

Since there are no more observations to read in the 6th iteration, the SAS system goes to the next DATA or PROC step.

Figure 4e. Fifth iteration for Program 5.

RESTRUCTURING DATASETS

Restructuring datasets denotes transforming data from one observation per subject (the *wide* format) to multiple observations per subject (the *long* format) or transforming data from the *long* format to data in the *wide* format. The purpose of the transformation to different formats is to suit the data format requirement for different types of statistical procedures. This type of data transformation can be easily done by using more advanced programming techniques, such as using ARRAY or using PROC TRANSPOSE. However, this can also be accomplished without advanced techniques for more simple cases.

FROM WIDE FORMAT TO LONG FORMAT

Suppose that you are transforming data from the *wide* format to the *long* format such as the example below. Notice that data in the *long* format has a variable, TIME, that is used to distinguish the different measurements for each subject in the *wide* format. The original variables in the *wide* format, S1 – S3, become the variable SCORE in the *long* format.

Wide:

	ID	S1	S2	S3
1	A01	3	4	5
2	A02	4	.	2

Long:

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

Since only two observations need to be read from the *wide* dataset, there will be only two iterations for the DATA step processing. That means you need to generate the output three times for each iteration. Also, any missing values in variables S1 – S3 will not be outputted in the *long* dataset. Here is a solution for using multiple OUTPUT statements in one DATA step.

Program 6:

```
data long (drop=s1-s3);
    set wide;
    time = 1;
    score = s1;
    if not missing(score) then output;
    time = 2;
    score = s2;
    if not missing(score) then output;
    time = 3;
    score = s3;
    if not missing(score) then output;
run;
```

In Program 6, immediately after the SET statement, the TIME variable is set to 1. Next, the value from S1 is assigned to the SCORE variable. Now all the elements for the first observation in the *long* dataset are ready for outputting. Before outputting, check whether the SCORE value is missing or not; if it is not missing, use the explicit OUTPUT statement to create the first observation for the *long* dataset. Next, assign value 2 to the TIME variable and assign the value from S2 to SCORE and output the dataset again. Similar processes are repeated; assign 3 to TIME, assign S3 to SCORE, and output. Within the first iteration of the DATA step processing, the values for TIME and SCORE are being replaced three times. Once they are replaced, they are outputted to the final dataset. See Figure 5a for more details.

The second iteration (see Figure 5b) is similar to the first iteration. The only difference is that S2 is missing. After the value for S2 is assigned to SCORE, the contents in the PDV are not copied to the final dataset because SCORE equals missing.

FIRST ITERATION:

Wide:

ID	S1	S2	S3
A01	3	4	5
A02	4	.	2

← Reading

```
data long (drop=s1-s3);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	1	0		•	•	•	•	•

EXPLANATION: _N_ is set to 1 and _ERROR_ is set to 0. Other variables are set to missing.

```
set wide;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	1	0	A01	3	4	5	•	•

EXPLANATION: The first observation from the *wide* dataset is copied to the PDV.

```
time = 1; score = s1;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	1	0	A01	3	4	5	1	3

EXPLANATION: TIME is set to 1 and SCORE is set to 3 (from the value of S1).

```
if not missing(score) then output;
```

Long:

ID	TIME	SCORE
A01	1	3

EXPLANATION: ID, TIME, and SCORE are copied to dataset *long* since SCORE is not missing.

```
time = 2; score = s2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	1	0	A01	3	4	5	2	4

EXPLANATION: TIME is set to 2 and SCORE is set to 4 (from the value of S2).

```
if not missing(score) then output;
```

Long:

ID	TIME	SCORE
A01	1	3
A01	2	4

EXPLANATION: ID, TIME, and SCORE are copied to dataset *long* since SCORE is not missing.

```
time = 3; score = s3;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	1	0	A01	3	4	5	3	5

EXPLANATION: TIME is set to 3 and SCORE is set to 5 (from the value of S3).

```
if not missing(score) then output;
```

Long:

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

EXPLANATION: ID, TIME, and SCORE are copied to dataset *long* since SCORE is not missing.

```
run;
```

EXPLANATION: There is no more implicit OUTPUT statement. The SAS system returns to the beginning of the DATA step to begin the 2nd iteration (see Figure 5a).

Figure 5a. First iteration for Program 6.

SECOND ITERATION:

Wide:

ID	S1	S2	S3
A01	3	4	5
A02	4	.	2

← Reading

```
data long (drop=s1-s3);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	2	0	A01	3	4	5	•	•

EXPLANATION: _N_ is incremented to 2. ID and S1-S3 are retained from the previous iteration. The newly-created variables, TIME and SCORE, are set to missing.

```
set wide;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	2	0	A02	4	•	2	•	•

EXPLANATION: The second observation from the *wide* dataset is copied to the PDV.

```
time = 1; score = s1;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	2	0	A02	4	•	2	1	4

EXPLANATION: TIME is set to 1 and SCORE is set to 4 (from the value of S1).

```
if not missing(score) then output;
```

Long:

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

EXPLANATION: ID, TIME, and SCORE are copied to dataset *long* since SCORE is not missing.

```
time = 2; score = s2;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	2	0	A02	4	•	2	2	•

EXPLANATION: TIME is set to 2 and SCORE is set to missing (from the value of S2).

```
if not missing(score) then output;
```

EXPLANATION: No output is generated since SCORE equals missing.

```
time = 3; score = s3;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	S1 (D)	S2 (D)	S3 (D)	TIME (K)	SCORE (K)
PDV:	2	0	A02	4	•	2	3	2

EXPLANATION: TIME is set to 3 and SCORE is set to 2 (from the value of S3).

```
if not missing(score) then output;
```

Long:

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

EXPLANATION: ID, TIME, and SCORE are copied to dataset *long* since SCORE is not missing.

```
run;
```

EXPLANATION: The SAS system returns to the beginning of the DATA step to begin the 3rd iteration. With no more observations to read in the 3rd iteration, SAS goes to the next DATA or PROC step.

Figure 5b. Second iteration for Program 6.

FROM LONG FORMAT TO WIDE FORMAT

Transforming data from the *long* format to the *wide* format is a little more complicated. Details for each single step will not be covered. However, if you attempt to draw the PDV yourself, you will be able to figure-out the contents of the PDV within each iteration of the DATA step processing. The SAS code is listed below:

Long:

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

Wide:

	ID	S1	S2	S3
1	A01	3	4	5
2	A02	4	.	2

Program 7:

```
proc sort data=long;
    by id time;
run;
data wide (drop=time score);
    set long;
    by id;
    retain s1-s3;
    if first.id then do;
        s1 = .;  s2 = .; s3 = .;
    end;
    if time = 1 then s1 = score;
    else if time = 2 then s2 = score;
    else s3 = score;
    if last.id = 1;
run;
```

Program 7 begins by sorting the *long* dataset by ID and TIME. Sorting the variable TIME within each ID is important because it ensures the horizontal order of S1 – S3 in the *wide* dataset for each subject can be matched correctly with the vertical order of SCORE in the *long* dataset.

Since you are reading five observations from the *long* dataset but only creating two observations, it means that you are *not* copying data from the PDV to the final dataset at each iteration. As a matter of fact, you only need to generate one observation once all the observations for each subject have been processed. That means that the newly- created variables S1 – S3 in the final dataset need to retain their values; otherwise S1 – S3 will be initialized to missing at the beginning of each iteration of the DATA step processing.

Notice that subject A02 is missing one observation for TIME equaling 2. The value of S2 from the previous subject (A01) will be copied to the dataset *wide* for the subject A02 instead of a missing value because S2 is being retained. To avoid this problem, initialize S1 – S3 to missing when processing the first observation for each subject.

USING THE PUT STATEMENT TO OBSERVE THE CONTENTS OF THE PDV DURING EACH STEP

If you are not sure what the contents of the PDV are during each step of the DATA step processing, use the PUT statement inside the DATA step, which will generate the contents of each variable in the PDV on the SAS log. For example,

Program 8:

```
data ex2_2;
    put "1ST PUT" _all_;
    set ex2;
    put "2ND PUT" _all_;
    retain total 0;
    put "3RD PUT" _all_;
    total = sum(total, score);
    put "4TH PUT" _all_;
run;
```

The PUT statement can combine text strings in quotations with the contents of the variable on the SAS log. The keyword _ALL_ means that all the variables, including the automatic variables, will be output to the SAS log. Here is the corresponding log from Program 8.

SAS log:

```
289  data ex2_2;
290      put "1ST PUT" _all_;
291      set ex2;
292      put "2ND PUT" _all_;
293      retain total 0;
294      put "3RD PUT" _all_;
295      total = sum(total, score);
296      put "4TH PUT" _all_;
297  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=0 _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
4TH 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=3 _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
```

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 Li
City of Hope National Medical Center
Department 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