How Best to Write a Loop in the DATA Step

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

ABSTRACT

During the execution of the DATA step processing, the DATA step works like a loop, repetitively reading the data and creating observations one at a time. We call this type of loop the implicit loop. Sometimes we need to execute certain SAS® statements repeatedly. In this situation, we need to construct an explicit loop by using the DO, DO WHILE, or DO UNTIL statements. There is a wide range of applications for explicit loops, such as generating random samples, reading multiple external data files, and so forth. However, in some scenarios, creating an explicit loop can be very tricky, even for seasoned programmers. Constructing a successful loop is dependent upon grasping SAS® programming fundamentals, such as understanding that the SAS data set is created one observation at a time in the program data vector (PDV). In this paper, you will learn how to create loops with various applications and what happens in the PDV when creating the explicit loop.

INTRODUCTION

A loop is one of the basic logic programming language structures that allows us to execute one or a group of statements repetitively until it reaches a predefined condition. This type of programming language construction is widely used in all computer languages. Compared to other programming languages, understanding the necessities of creating loops is more complex for the SAS language since there are implicit and explicit loops and sometimes beginning programmers can't distinguish clearly between them. Knowing when the time is right to create an explicit loop is one of the challenges that face all beginning programmers.

IMPLICIT AND EXPLICIT LOOPS

COMPILATION AND EXECUTION PHASES

A DATA step is processed in a two-phase sequence: compilation and execution phases. In the compilation phase, each statement is scanned for syntax errors. The Program Data Vector (PDV) is created according to the descriptor portion of the input dataset.

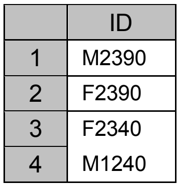
The execution phase starts after the compilation phase. In the execution phase, SAS uses the PDV to build the new dataset. Not all of the variables in the PDV are outputted to the final dataset. The variables in the PDV that are flagged with "K" (which stands for "kept") will be written to the output dataset; on the other hand, the variables flagged with "D" (which stands for "dropped") will not.

During the execution phase, the DATA step works like a loop, repetitively reading data values from the input dataset, executing statements, and creating observations for the output dataset one at a time. This is the implicit loop. SAS stops reading the input file when it reaches the end-of-file marker, which is located at the end of the input data file. At this point, the implicit loop ends.

IMPLICIT LOOPS

The following example shows how the implicit loop is processed. Suppose that you would like to assign each subject in a group of patients in a clinical trial where each patient has a 50% chance of receiving either the drug or a placebo. For illustration purposes, only four patients from the trial are used in the example. The dataset is similar to the one below. The solution is shown in *program 1*.

Patient.sas7bdat



	ID
1	M2390
2	F2390
3	F2340
4	M1240

Program 1:

```
data example1 (drop=rannum);
    set patient;
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
run;
proc print data=example1;
run;
```

Output:

```
Obs     ID       group

 1     M2390       P
 2     F2390       D
 3     F2340       D
 4     M1240       D
```

FIRST ITERATION:

Patient.sas7bdat:

	ID
1	M2390
2	F2390
3	F2340
4	M1240

← Reading

End-of-file marker →

```
data example1 (drop=rannum);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0		•	

EXPLANATION: _N_ is initialized to 1 and _ERROR_ is initialized to 0. The variables ID, GROUP, and RANNUM are set to missing.

```
set patient;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	M2390	•	

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

```
rannum = ranuni(2);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	M2390	0.3699251396	

EXPLANATION: RANNUM is generated by the RANUNI function.

```
if  rannum> 0.5 then group = 'D';  else group ='P';
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	M2390	0.3699251396	P

EXPLANATION: Since RANNUM is not greater than 0.5, GROUP is assigned with value 'P.'

```
run;
```

Example1:

ID	GROUP
M2390	P

EXPLANATION: The implicit OUTPUT statement at the end of the DATA step tells SAS to write observations to the dataset. Since SAS didn't read the end-of-file marker, it returns to the beginning of the DATA step to begin the 2nd iteration.

Figure 1. The first iteration of Program 1.

At the beginning of the execution phase, the automatic variable _N_ is initialized to 1 and _ERROR_ is initialized to 0. _N_ is used to indicate the current observation number. _ERROR_ is more often used to signal the data entry error. The non-automatic variables are set to missing (See Figure 1). Next, the SET statement copies the first observation from the dataset *patient* to the PDV. Then the variable RANNUM is generated by the **RANUNI**[1] function. Since RANNUM is not greater than 0.5, GROUP is assigned with value ‘P’. The implicit OUTPUT statement at the end of the DATA step tells SAS to write the contents from the PDV that is marked with “K” to the dataset *example1*. The SAS system returns to the beginning of the DATA step to begin the second iteration.

At the beginning of the second iteration, since data is read from an existing SAS dataset, value in the PDV for the ID variable is retained from the previous iteration. The newly created variables RANNUM and GROUP are initialized to missing[2]. The automatic variable _N_ is incremented to 2. Next, RANNUM is generated and GROUP is assigned to ‘D’. The implicit OUTPUT statement tells SAS to write the contents from the PDV to the output dataset *example1*. The SAS system returns to the beginning of the DATA step to begin the third iteration.

The entire process for the third and fourth iterations is similar to the previous iterations. Once the fourth iteration is completed, SAS returns to the beginning of the DATA step again. At this time, when SAS attempts to read an observation from the input dataset, it reaches the end-of-file-marker, which means that there are no more observations to read. Thus, the execution phase is completed.

EXPLICIT LOOP

In the previous example, the patient ID is stored in an input dataset. Suppose you don’t have a dataset containing the patient IDs. You are asked to assign four patients with a 50% chance of receiving either the drug or the placebo. Instead of creating an input dataset that stores ID, you can create the ID and assign each ID to a group in the DATA step at the same time. For example

Program 2:

```
data example2(drop = rannum);
    id = 'M2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2340';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'M1240';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;
run;
```

The DATA step in *Program 2* begins with assigning ID numbers and then assigns the group value based on

[1] The **RANUNI** function generates a number following uniform distribution between 0 and 1. The general form is **RANUNI(seed)**, where seed is a nonnegative integer. The RANUNI function generates a stream of numbers based on *seed*. When *seed* is set to 0, which is the computer clock, the generated number cannot be reproduced. However, when *seed* is a non-zero number, the generated number can be produced.

[2] 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. When creating a SAS dataset based on a SAS dataset, SAS sets each variable to missing in the PDV *only* before the first iteration of the execution. Variables will 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. However, the newly created variable, which only exists in the output dataset, will be set to *missing* in the PDV at the beginning of *every* iteration of the execution.

a generated random number. There are four explicit OUTPUT statements[3] that tells SAS to write the current observation from the PDV to the SAS dataset immediately, not at the end of the DATA step. This is what we intended to do. However, without using the explicit OUTPUT statement, we will only create one observation for ID =M1240. Notice that most of the statements above are identical. To reduce the amount of coding, you can simply rewrite the program by placing repetitive statements in a DO loop. Following is the general form for an iterative DO loop:

DO INDEX-VARIABLE = VALUE1, VALUE2, …, VALUEN***;***

SAS STATEMENTS

END;

In the iterative DO loop, you must specify an INDEX-VARIABLE that contains the value of the current iteration. The loop will execute along VALUE1 through VALUEN and the VALUES can be either character or numeric. Here's the improved version of Program 2

Program 3:

```
data example3 (drop = rannum);
    do id = 'M2390', 'F2390', 'F2340', 'M1240';
        rannum = ranuni(2);
        if rannum> 0.5 then group = 'D';
        else group ='P';
        output;
    end;
run;
proc print data=example3;
run;
```

Output:

```
Obs     id      group

 1     M2390      P
 2     F2390      D
 3     F2340      D
 4     M1240      D
```

THE ITERATIVE DO LOOP ALONG A SEQUENCE OF INTEGERS

More often the iterative DO loop along a sequence of integers is used.

DO INDEX-VARIABLE = START **TO** STOP <**BY** INCREMENT>***;***

SAS STATEMENTS

END;

The loop will execute from the START value to the END value. The optional BY clause specifies an increment between START and END. The default value for the INCREMENT is 1. START, STOP, and INCREMENT can be numbers, variables, or SAS expressions. These values are set upon entry into the DO loop and cannot be modified during the processing of the DO loop. However, the INDEX-VARIABLE can be changed within the loop.

Suppose that you are using a sequence of number, say 1 to 4, as patient IDs; you can rewrite the previous program as below:

Program 4:

```
data example4 (drop = rannum);
    do id = 1 to 4;
        rannum = ranuni(2);
        if rannum> 0.5 then group = 'D';
        else group ='P';
        output;
    end;
run;
proc print data=example4;
run;
```

Output:

```
Obs     id      group

 1       1      P
 2       2      D
 3       3      D
 4       4      D
```

Program 4 didn't specify the INCREMENT value, thus the default value 1 is used. You can also decrement a DO loop by using a negative value, such as -1. The execution phase is illustrated in figure 2a and 2b.

[3] 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. 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

```
data example4 (drop=rannum);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	•	•	

EXPLANATION: _N_ is initialized to 1 and _ERROR_ is initialized to 0. The variables ID, GROUP, and RANNUM are set to missing.

FIRST ITERATION OF THE DO LOOP:

```
do id = 1 to 4;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	1	•	

EXPLANATION: ID is assigned to 1 at the beginning of the first DO loop.

```
rannum = ranuni(2);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	1	0.3699251396	

EXPLANATION: RANNUM is generated by the RANUNI function.

```
If rannum> 0.5 then group = 'D';  else group ='P';
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	1	0.3699251396	P

EXPLANATION: Since RANNUM is not greater than 0.5, GROUP is assigned with value 'P.'

```
output;
```

Example3:

ID	GROUP
1	P

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example3*. SAS reaches the end of the DO loop.

SECOND ITERATION OF THE DO LOOP:

```
do id = 1 to 4;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	2	0.3699251396	P

EXPLANATION: ID is incremented to 2; SINCE 2 IS ≤ 4, the 2nd iteration continues.

```
rannum = ranuni(2);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	2	0.9401774313	P

EXPLANATION: RANNUM is generated by the RANUNI function.

```
if rannum> 0.5 then group = 'D';  else group ='P';
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	2	0.9401774313	D

EXPLANATION: Since RANNUM is greater than 0.5, GROUP is assigned with value 'D.'

```
output;
```

Example3:

ID	GROUP
1	P
2	D

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example3*. SAS reaches the end of the DO loop.

Figure 2a. The first two iterations of the DO loop in Program 4.

THIRD ITERATION OF THE DO LOOP:

```
do id = 1 to 4;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	3	0.9401774313	D

EXPLANATION: ID is incremented to 3; since 3 is ≤ 4, the 3rd iteration continues.

```
rannum = ranuni(2);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	3	0.7996486122	D

EXPLANATION: RANNUM is generated by the RANUNI function.

```
if rannum> 0.5 then group = 'D';  else group ='P';
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	3	0.7996486122	D

EXPLANATION: Since RANNUM is greater than 0.5, GROUP is assigned with value 'D.'

```
output;
```

Example3:

ID	GROUP
1	P
2	D
3	D

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example3*. SAS reaches the end of DO loop.

FOURTH ITERATION OF THE DO LOOP:

```
do id = 1 to 4;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	4	0.7996486122	D

EXPLANATION: ID is incremented to 4; since 4 is ≤ 4, the 4th iteration continues.

```
rannum = ranuni(2);
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	4	0.5187972908	D

EXPLANATION: RANNUM is generated by the RANUNI function.

```
if rannum> 0.5 then group = 'D';  else group ='P';
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	4	0.5187972908	D

EXPLANATION: Since RANNUM is greater than 0.5, GROUP is assigned with value 'D.'

```
output;
```

Example3:

ID	GROUP
1	P
2	D
3	D
4	D

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example3*. SAS reaches the end of DO loop.

```
end;
```

	_N_ (D)	_ERROR_ (D)	ID (K)	RANNUM (D)	GROUP(K)
PDV:	1	0	5	0.5187972908	D

EXPLANATION: ID is incremented to 5; since 5 is > 4, the loop ends.

```
run;
```

EXPLANATION: There will be no explicit OUTPUT statement. Since we didn't read an input dataset, the DATA step execution ends.

Figure 2b. The last two iterations of the DO loop in Program 4.

EXECUTING DO LOOPS CONDITIONALLY

Program 4 used an iterative DO loop, which requires that you specify the *number* of iterations for the DO loop. Sometimes you will need to execute statements repetitively until a condition is met. In this situation, you need to use either the DO WHILE or DO UNTIL statements. Following is the syntax for the DO WHILE statement:

DO WHILE (EXPRESSION)***;***
SAS STATEMENTS
END;

In the DO WHILE loop, the EXPRESSION is evaluated at the top of the DO loop. The DO loop will not execute if the EXPRESSION is false. We can rewrite *program 4* by using the DO WHILE loop.

Program 5:

```
data example5 (drop=rannum);
    do while (id <4);
        id + 1;
        rannum = ranuni(2);
        if  rannum> 0.5 then group = 'D';
        else group ='P';
        output;
    end;
run;
```

In *program 5*, the ID variable is created inside the loop by using the SUM statement[4]. Thus, the variable ID is initialized to 0 in the PDV at the beginning of the DATA step execution, which is before the DO WHILE statement. At the beginning of the loop, the condition is being checked. Since ID equals 0, which satisfies the condition (ID < 4), the first iteration begins. For each iteration, the ID variable is accumulated from the SUM statement. The iteration will be processed until the condition is not met.

Alternatively, you can also use the DO UNTIL loop to execute the statements conditionally. Unlike DO WHILE loops, the DO UNTIL loop evaluates the condition at the end of the loop. That means the DO UNTIL loop always executes at least once. The DO UNTIL loop follows the form below:

DO UNTIL (EXPRESSION)***;***
SAS STATEMENTS
END;

The following program is based on *program 4* by using the DO UNTIL loop.

Program 6:

```
data example6 (drop=rannum);
    do until (id >=4);
        id +1;
        rannum = ranuni(2);
        if rannum > 0.5 then group = 'D';
        else group ='P';
        output;
    end;
run;
```

NESTED LOOPS

You can place a loop within another loop. To continue with the previous example, suppose that you would like to assign 12 subjects from 3 cancer centers (“COH”, “UCLA”, and “USC”) with 4 subjects per center, where each patient has a 50% chance of receiving either the drug or a placebo. A nested loop can be used to solve this problem. In the outer loop, the INDEX-VARIABLE, CENTER, is assigned to the values with the name of the three cancer centers. For each iteration of the outer loop, there is an inner loop that is used to assign each patient to a group.

[4] The variable that is created by the SUM statement is automatically set to 0 at the beginning of the first iteration of the DATA step execution and it is retained in following iterations.

Program 7:

```
data example7 (drop=rannum);
   length center $4;
   do center = "COH", "UCLA", "USC";
      do id = 1 to 4;
         if ranuni(2) > 0.5 then group = 'D';
         else group ='P';
         output;
      end;
   end;
run;
proc print data=example7;
run;
```

Output:

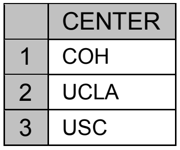
```
Obs    center    id    group

  1     COH       1      P
  2     COH       2      D
  3     COH       3      D
  4     COH       4      D
  5     UCLA      1      D
  6     UCLA      2      D
  7     UCLA      3      P
  8     UCLA      4      P
  9     USC       1      P
 10     USC       2      P
 11     USC       3      D
```

COMBINING IMPLICIT AND EXPLICIT LOOPS

In *example 7*, all the observations were created from one DATA step since we didn't read any input data. Sometimes it is necessary to use an explicit loop to create multiple observations for each observation that is read from an input dataset. In *example 7*, we used an outer loop to create a CENTER variable. Suppose the values for CENTER is stored in a SAS dataset. For each center, you need to assign 4 patients where each patient has a 50% chance of receiving either the drug or a placebo. Following is the program to solve this problem:

Cancer_center.sas7bdat



	CENTER
1	COH
2	UCLA
3	USC

Program 8:

```
data example8 (drop=rannum);
    set cancer_center;
    do id = 1 to 4;
        if  ranuni(2)> 0.5 then group = 'D';
        else group ='P';
        output;
    end;
run;
```

Program 8 is an example of using both implicit and explicit loops. The DATA step is an implicit loop that is used to read observations from the input dataset, *Cancer_center.* For each CENTER that is being read, there is an explicit loop to assign patients to either 'D' or 'P.'

UTILIZING LOOPS TO CREATE SAMPLES

In some situations, you may want to draw samples from a dataset. There are two kinds of sampling schemes: systematic and random samples.

DIRECT-ACCESS MODE

Usually when reading the entire SAS dataset, by default, SAS reads the dataset sequentially. SAS reads one observation for each iteration of the DATA step. This process will stop once it reaches the end-of-file marker. Instead of reading data sequentially, SAS can also access an observation directly via direct-access mode.

There are three important components for using the direct-access mode. First, you need to tell SAS which observation you would like to select. The step is done by using the **POINT =** in the SET statement, which has the following form:

SET SAS-DATA-SET **POINT** = VARIABLE;

The VARIABLE specified by the **POINT =** option is a temporary variable and it is not outputted to the output dataset. This variable is set to 0 in the PDV at the very beginning of the DATA step. Then it must be assigned to an observation number before the SET statement.

When using direct-access mode, SAS will not be able to detect the end-of-file marker. Without telling SAS explicitly when to stop processing, it will cause infinite looping. Therefore, in order to utilize the direct-access mode, you need to use the **STOP** statement at the end of the DATA step. Here's the general form:

STOP;

Recall that SAS writes observations from the PDV to the output dataset at the end of the DATA step if there is no explicit **OUTPUT** statement in the DATA step. However, if you use the **STOP** statement, the DATA step processing will stop *before* the end of the DATA step. Thus, the last step to use direct-access mode is to write an explicit **OUTPUT** statement before the **STOP** statement.

Suppose you have the following dataset and you would like to create a dataset which only contains the fifth observation of this data. You can write the following code.

sbp.sas7bdat

	id	sbp
1	01	145
2	02	119
3	03	126
4	04	106
5	05	151
6	06	112
7	07	127
8	08	119
9	09	113

Program 9:

```
data example9;
    obs_n = 5;
    set sbp point= obs_n;
    output;
    stop;
run;
```

CREATING A SYSTEMATIC SAMPLE

A systematic sample is created by selecting every k^{th} observation from an original dataset. In other words, the systematic sample cannot be created sequentially; hence, a direct-access mode must be used. You can create a systematic sample by using an iterative DO loop, which requires providing START, STOP, and INCREMENT values. Normally, START is set to 1, STOP is set to the total number of observations from the input dataset, and INCREMENT is set to *k* that indicates every k^{th} observation that you want to select.

You can determine the total number of observations by running the CONTENTS procedure. Instead of running a separate procedure, you can also provide the total number of observations by using the **NOBS =** option in the SET statement. Here's the general form for the **NOBS =** option:

SET SAS-DATA-SET **NOBS** = VARIABLE;

The VARIABLE specified by the **NOBS =** option is a temporary variable that contains the number of observations of the SAS-DATA-SET. It will not be outputted to the final dataset. The VARIABLE is created automatically based on the descriptor portion of the SAS-DATA-SET during the compilation phase. It will retain its value throughout the execution phase. For example, to create a systematic sample that contains every 3rd observation from the *sbp.sas7bdat*, you can write the following code. Figure 3a and 3b illustrate the execution process in detail. To simplify the illustration, the automatic variable _ERROR_ is not presented in the figures.

Notice that the automatic variable _N_ is 1 throughout the execution phase because SAS didn't read the input data sequentially. For each iteration of the DO loop, SAS uses the direct-access mode to read observations based on the observation number given from the CHOOSE variable.

Program 10:

```
data example10;
    do choose = 1 to total by 3;
        set sbp point=choose nobs=total;
        output;
    end;
    stop;
run;
proc print data=example10;
run;
```

Output:

```
Obs     id    sbp

  1     01    145
  2     04    106
  3     07    127
```

```
data example10;
```

PDV:	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
	1	0	9		•

EXPLANATION: _N_ is initialized to 1. CHOOSE is set to 0. TOTAL, the total number of observations, is set to 9. ID and SBP are set to missing.

FIRST ITERATION OF THE DO LOOP:

```
do choose = 1 to total by 3;
```

PDV:	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
	1	1	9		•

EXPLANATION: CHOOSE is assigned to 1 at the beginning of the first DO loop.

```
set sbp point=choose nobs=total;
```

PDV:	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
	1	1	9	01	145

EXPLANATION: SAS reads the first observation via direct-access mode.

```
output;
```

Example9:

ID	SBP
01	145

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example9*. SAS reaches the end of the DO loop.

Figure 3a. The first iteration of the DO loop in Program 10.

SECOND ITERATION OF THE DO LOOP:

```
do choose = 1 to total by 3;
```

	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
PDV:	1	4	9	01	145

EXPLANATION: CHOOSE is incremented to 4; since 4 ≤ TOTAL, which is 9, the 2nd iteration continues.

```
set sbp point=choose nobs=total;
```

	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
PDV:	1	4	9	04	106

EXPLANATION: SAS reads the 4th observation via direct-access mode.

```
output;
```

Example9:

ID	SBP
01	145
04	106

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example9*. SAS reaches the end of the DO loop.

THIRD ITERATION OF THE DO LOOP:

```
do choose = 1 to total by 3;
```

	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
PDV:	1	7	9	04	106

EXPLANATION: CHOOSE is incremented to 7; since 7 ≤ TOTAL, which is 9, the 3rd iteration continues.

```
set sbp point=choose nobs=total;
```

	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
PDV:	1	7	9	07	127

EXPLANATION: SAS reads the 7th observation via direct-access mode.

```
output;
```

Example9:

ID	SBP
01	145
04	106
07	127

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example9*. SAS reaches the end of the DO loop.

```
end;
```

	_N_ (D)	CHOOSE (D)	TOTAL (D)	ID (K)	SBP(K)
PDV:	1	11	9	07	127

EXPLANATION: CHOOSE is incremented to 11; since 11 > TOTAL, the loop ends.

```
stop;
run;
```

EXPLANATION: The STOP statement stops the DATA step processing.

Figure 3b. The last two iterations of the DO loop in Program 10.

CREATING A RANDOM SAMPLE WITH REPLACEMENT

A random sample is created from an original dataset on a random basis. A random sample with replacement means that an observation is replaced back into the original dataset after it is chosen. Hence, any observations can be chosen more than once. Suppose you would like to create a sample of 3 observations with replacement from *sbp.sas7bdat*.

Program 11:

```
data example11 (drop= i);
    do i =1 to 3;
        choose = ceil(ranuni(5)*total);
        set sbp point=choose nobs=total;
        output;
    end;
    stop;
run;
proc print data=example11;
run;
```

Output:

```
Obs     id     sbp


 1      09     113
 2      08     119
 3      09     113
```

An important step in creating a random sample with replacement is generating a random number that indicates which observation needs to be selected. In *program 11*, to select observations randomly, an integer between 1 and the total number of observations needs to be generated within each iteration of the loop. Since the RANUNI function generates a number between 0 and 1, when multiplying this generated number, **ranuni(5)** with the total number of observations (TOTAL), the resulting number will be between 0 and the number of observations. Since you need an integer value, you can use the CEIL function, which returns the smallest integer that is greater than or equal to its argument.

CREATING A RANDOM SAMPLE WITHOUT REPLACEMENT

A random sample without replacement means that once an observation is randomly selected, it cannot be replaced back into the original dataset. Thus, any observations cannot be chosen more than once. The algorithm to generate a random sample without replacement is more complicated than the one with replacement. The following example is adapted from “SAS PROGRAMMING II: Manipulating Data in the Data Step”.

The algorithm is summarized in Table 1. RANNUM is the random number that is generated for each observation. SIZE is the sample size which is decremented by 1 for each observation that is being selected for the final sample. LEFT is used to keep track of the total number of observations and decremented by 1 once an observation has been processed. PCT is calculated by SIZE divided by LEFT. An observation with RANNUM that is less than PCT will be chosen for the final sample. The loop stops when SIZE reaches 0. A DO WHILE loop will be suitable for this problem.

	id	sbp		randnum	size	left	Pct = size/left	Rannum < Pct?	Choose
1	01	145	←	0.22	3	9	0.33	YES	1
2	02	119	←	0.64	2	8	0.25	NO	NO
3	03	126	←	0.79	2	7	0.29	NO	NO
4	04	106	←	0.11	2	6	0.33	YES	4
5	05	151	←	0.06	1	5	0.2	YES	5
6	06	112							
7	07	127							
8	08	119							
9	09	113							

Loop Stop!

Table1: Algorithm for program 12.

Program 12:

```
data example12 (drop=size left randnum);
    size = 3;
    left = total;
    do while (size > 0);
        choose + 1;
        randnum = ranuni(12);
        pct = size / left;
        if randnum < pct then do;
            set sbp point=choose nobs=total;
            output;
            size = size - 1;
        end;
        left = left - 1;
    end;
    stop;
run;
proc print data=example12;
run;
```

Output:

```
Obs     id     sbp

  1     01     145
  2     04     106
  3     05     151
```

UTILIZING LOOP TO READ A LIST OF EXTERNAL FILES

THE INFILE STATEMENT WITH THE END= OPTION

To read an external file, you can use the INFILE statement. In this paper, the external file refers to the raw data or the text file. For example, to read the following external file, *text1.txt*, which is located in “C:\”, you can write the following code:

text1.txt:

```
01 145
02 119
```

Program 13:

```
data example13;
    infile "C:\text1.txt";
    input id $ sbp;
run;
```

Since there are two observations in the external file, SAS will use two DATA step iterations (the implicit loop) to read the data. Like a SAS dataset, the external file also contains an end-of-file marker. When SAS reaches the end-of-file marker, it stops reading.

Alternatively, you can use an explicit loop to read the external file. In order to construct an explicit loop, you need to either specify the number of iterations for the iterative DO loop or you need to specify a condition for the DO WHILE or DO UNTIL loops. One way to specify a condition is by telling SAS to read the observations until it reads the last record. In order for SAS to detect reading the last record, you can create a temporary variable by using the **END =** option in the INFILE statement. Here’s the general form:

INFILE FILE-SPECIFICATION **END** = VARIABLE;

The VARIABLE specified by the **END =** option is set to 1 when SAS reads the last record of the external file; otherwise it sets to 0. The following program uses the DO UNTIL loop to read the external file.

Program 14:

```
data example14;
    infile "C:\text1.txt" end = last;
    do until (last = 1);
        input id $ sbp;
        output;
    end;
run;
```

In *program 14*, the DATA step iteration only goes through once. Within this iteration, the DO UNTIL loop iterates twice to read the two observations in *text1.txt*. During the DATA step iteration, the automatic variable _N_ is set to 1. Once the DATA step iteration is over, SAS starts the second iteration of the DATA step and _N_ is incremented to 2. At this point, SAS reaches the end-of-file marker and it stops processing. The end-of-file marker is detected by the DATA step, not by the DO UNTIL loop.

THE INFILE STATEMENT WITH THE FILEVAR = OPTION

In the INFILE statement, you generally specify the name and the location of the external file immediately after the key word INFILE. Alternatively, you can use the **FILEVAR =** option in the INFILE statement to read an external file that is specified by the **FILEVAR =** option. It has the following form:

INFILE FILE-SPECIFICATION **FILEVAR** = VARIABLE;

Like the automatic variable, the VARIABLE specified in the **FILEVAR =** option is not outputted into the final dataset. This VARIABLE contains the name of the external file and must be created before the **INFILE** statement. When you use the **FILEVAR=** option, the file-specification is just a placeholder, not an actual filename. For example, *program 13* can be re-written as below:

Program 15:

```
data example15;
    filename = "C:\text1.txt";
    infile dummy filevar = filename;
    input id $ sbp;
run;
```

Log:

```
167  data example14;
168      filename = "C:\text1.txt";
169      infile dummy filevar = filename;
170      input id $ sbp;
171  run;

NOTE: The infile DUMMY is:
      File Name=C:\text1.txt,
      RECFM=V,LRECL=256

NOTE: 2 records were read from the infile DUMMY.
      The minimum record length was 6.
      The maximum record length was 6.
NOTE: The data set WORK.EXAMPLE13 has 2 observations and 2 variables.
```

Notice that in the SAS log, SAS uses the placeholder, *dummy*, to report the name of the external file that is being read.

READING MULTIPLE EXTERNAL FILES

There are situations when you may want to read a group of external files into SAS and concatenate them into one dataset. Each of the external files has identical formats. Instead of reading them individually by using separate DATA steps, you can read them all by using the **FILEVAR =** option in the **INFILE** statement in one single DATA step. The **FILEVAR =** option will cause the INFILE statement to close the current input file and open a new one which is the **FILEVAR =** option. For example, suppose that there are three external files, *text1.txt*, *text2.txt*, and *text3.txt*.

text1.txt:

```
01 145
02 119
```

text2.txt:

```
03 126
04 106
```

text3.txt:

```
05 140
06 118
```

To read these files by using only one single DATA step, a few things need to be considered before creating the SAS code. In *program 15*, three statements were used to read the external file: the first statement created the temporary variable that contained the external file name, immediately followed by the INFILE and INPUT statements. In order to read a group of external files, these three statements needed to be placed inside of a loop. Notice that the names of the external files are *text1.txt*, *text2.txt*, and *text3.txt,* which suggests that you create an iterative DO loop and iterate between 1 and 3. Within the DO loop, create the variable, NEXT, to contain the name of the external file by concatenating the "C:\text", loop index, and ".txt" via the via **||** operator. If you use the index variable that contains numeric values between 1 and 3, then you can use the PUT function to convert the numeric values into character values. For example,

```
next = "C:\text" || put(i, 1.) || ".txt";
```

An explicit **OUTPUT** statement is also necessary to write the current contents from the PDV to the output dataset within the loop. Since you are using the **FILEVAR =** option to control closing the current input file and opening a new file, SAS will not be able to detect the end-of-file marker. Thus, you also need to place a **STOP** statement outside the loop. In order to see which observation is read from which file, a FILENAME variable is also created in the following program:

Program 16:

```
data example16 (drop=i);
    do i = 1 to 3;
        next = "C:\text" || put(i, 1.) || ".txt";
        infile temp filevar = next;
        fileName = next;
        input id $ sbp;
        output;
    end;
    stop;
run;
proc print data=example16;
run;
```

Output:

```
Obs       fileName       id    sbp

 1      C:\text1.txt     01    145
 2      C:\text2.txt     03    126
 3      C:\text3.txt     05    140
```

Program 16 didn't create the output that we expected; only the first observation from each external file was outputted to the dataset. The INPUT statement within the loop only read one line of the external file. Once one iteration of the DO loop has completed, the following iteration starts to read a new file that is specified by the NEXT variable. In order to read all the observations of each file, you can utilize the **END =** option and include a **DO UNTIL** loop within the iterated DO loop. The modified program is listed below and a detailed explanation of the program is illustrated in *figure 4a – 4c.*

Program 17:

```
data example17 (drop=i);
    do i = 1 to 3;
        next = "C:\text" || put(i, 1.)|| ".txt";
        do until (last);
            infile dummy filevar = next end=last;
            input id $ sbp;
            output;
        end;
    end;
    stop;
run;
proc print data=example17;
run;
```

Output:

```
Obs     id    sbp

 1      01    145
 2      02    119
 3      03    126
 4      04    106
 5      05    140
 6      06    118
```

```
data example17 (drop=i);
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	•		0		•

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

FIRST ITERATION OF THE DO LOOP (OUTER LOOP):

```
do i = 1 to 3;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	1		0		•

EXPLANATION: I is assigned to 1 at the beginning of the first DO loop.

```
next = "C:\text" || put(i, 1.) || ".txt";
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	1	C:\text1.txt	0		•

EXPLANATION: The NEXT variable is assigned with the value 'C:\text1.txt'.

FIRST ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last);
```

EXPLANATION: The DO UNTIL loop evaluates the condition at the end of the loop.

```
infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	1		1	4	5		…

EXPLANATION: The INFILE statement reads the first data line from 'text1.txt' into the input buffer.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	1	C:\text1.txt	0	01	145

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.

```
output;
```

Example16:

ID	SBP
01	145

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

SECOND ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last); infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	2		1	1	9		…

EXPLANATION: INFILE reads the second data line into the input buffer. Since this is the last record of the 'text1.txt', LAST is set to 1 in the PDV.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	1	C:\text1.txt	1	02	119

EXPLANATION: : INPUT reads data values from the input buffer and writes them to the PDV.

```
output;
```

Example16:

ID	SBP
01	145
02	119

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

```
end;
```

EXPLANATION: Since LAST = 1, the inner loop ends.

Figure 4a. The first iteration of the DO loop in Program 17.

SECOND ITERATION OF THE DO LOOP (OUTER LOOP):

```
do i = 1 to 3;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	2	C:\text1.txt	1	02	119

EXPLANATION: I is assigned to 2; since I ≤ 3, the second iteration of the outer loop continues.

```
next = "C:\text" || put(i, 1.) || ".txt";
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	2	C:\text2.txt	1	02	119

EXPLANATION: The NEXT variable is assigned with the value ‘C:\text2.txt’.

FIRST ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last);
```

EXPLANATION: The DO UNTIL loop evaluates the condition at the end of the loop.

```
infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	3		1	2	6		…

EXPLANATION: INFILE reads first data line from ‘text2.txt’ into the input buffer. Since this is not the last record of ‘text2.txt’, LAST is set to 0 in the PDV.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	2	C:\text2.txt	0	03	126

EXPLANATION: : The INPUT statement reads data values from the input buffer to the PDV.

```
output;
```

Example16:

ID	SBP
01	145
02	119
03	126

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

SECOND ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last);
infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	4		1	0	6		…

EXPLANATION: INFILE statement reads the second data line into the input buffer. Since this is the last record of ‘text2.txt’, LAST is set to 1 in the PDV.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	2	C:\text2.txt	1	04	106

EXPLANATION: INPUT statement reads data values from the input buffer and writes them to the PDV.

```
output;
```

Example16:

ID	SBP
01	145
02	119
03	126
04	106

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

```
end;
```

EXPLANATION: : Since LAST =1, the inner loop ends.

Figure 4b. The second iteration of the DO loop in Program 17.

THIRD ITERATION OF THE DO LOOP (OUTER LOOP):

```
do i = 1 to 3;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	3	C:\text2.txt	1	04	106

EXPLANATION: I is assigned to 3; since I ≤ 3, the third iteration of the outer loop continues.

```
next = "C:\text" || put(i, 1.) || ".txt";
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	3	C:\text3.txt	1	04	106

EXPLANATION: The NEXT variable is assigned with the value ‘C:\text3.txt’.

FIRST ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last); infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	5		1	4	0		…

EXPLANATION: INFILE reads the first data line from ‘text2.txt’ into the input buffer. Since this is not the last record of ‘text3.txt’, LAST is set to 0 in the PDV.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	3	C:\text3.txt	0	05	140

EXPLANATION: : The INPUT statement reads data values from the input buffer to the PDV.

```
output;
```

Example16:

ID	SBP
01	145
02	119
03	126
04	106
05	140

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

SECOND ITERATION OF THE DO UNTIL LOOP (INNER LOOP):

```
do until (last);  infile temp filevar = next end=last;
```

	1	2	3	4	5	6	7	…
Input buffer:	0	6		1	1	8		…

EXPLANATION: The INFILE statement reads the second data line into the input buffer. Since this is the last record of ‘text3.txt’, LAST is set to 1 in the PDV.

```
input id $ sbp;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	3	C:\text3.txt	1	06	118

EXPLANATION: INPUT statement reads data values from the input buffer and writes them to the PDV.

```
output;
```

Example16:

ID	SBP
01	145
02	119
03	126
04	106
05	140
06	118

EXPLANATION: The OUTPUT statement tells SAS to write observations to *Example12*. SAS reaches the end of the DO UNTIL loop.

```
end;
```

EXPLANATION: Since LAST = 1, the inner loop ends.

```
end;
```

	_N_(D)	I (D)	NEXT (D)	LAST (D)	ID (K)	SBP(K)
PDV:	1	4	C:\text3.txt	1	06	118

EXPLANATION: I is incremented to 4; since I > 3, the outer loop ends.

```
stop;
run;
```

EXPLANATION: The STOP statement stops the DATA step processing.

Figure 4c. The third iteration of the DO loop in Program 17.

ARRAY

There is a wide range of applications in using loop structures with ARRAY processing. Since ARRAY is a large and different topic, we are not covering ARRAY in this talk.

CONCLUSION

Loops are important programming language structures that allow us to create more simplified and efficient programming codes. There are wide ranges of applications in utilizing loop structures. However, in order to use loop structures correctly, we need to understand how DATA steps are processed. Even for experienced programmers, when trying to debug our programming errors, we often realize that most of the errors are closely related to programming fundamentals, which is understanding how the PDV works.

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