4

BY-Group Processing in the DATA Step

4.1 Introduction to BY-Group Processing

Most of the examples in previous chapters used a data set that contains only 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. *Longitudinal Data and SAS® A Programmer's Guide* (Cody, 2005) provides many useful applications for manipulating 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.

4.1.1 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).

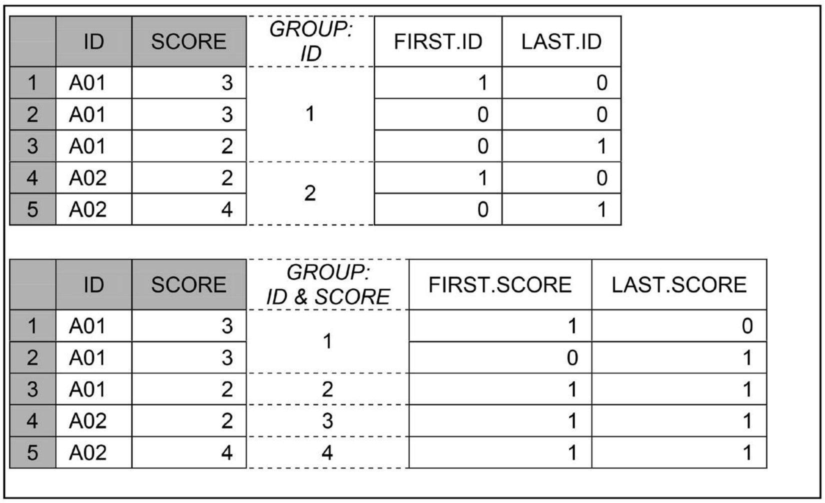
During BY-group processing, SAS creates two temporary indicator variables for each BY variable: FIRST.VARIABLE and LAST.VARIABLE. Because FIRST.VARIABLE and LAST.VARIABLE are temporary variables, they are not sent to the output data set. Both FIRST.VARIABLE and LAST.VARIABLE are initialized to 1 at the beginning of the DATA step execution. Then FIRST.VARIABLE is set to 1 in the program data vector (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 following SAS data set, SAS4_1, which consists of five observations with the values of SCORE for two subjects, A01 and A02.

SAS4_1:

	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 in Figure 4.1. 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



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

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

FIGURE 4.1
The top table illustrates the values for automatic variables FIRST.ID and LAST.ID. The bottom table illustrates the values for FIRST.SCORE and LAST.SCORE.

the PDV. There will be four BY groups based on unique combination values of ID and SCORE.

4.1.2 The Execution Phase of BY-Group Processing

Suppose that you would like to calculate the total scores for each subject in the SAS4_1 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 4.1.

Program 4.1:

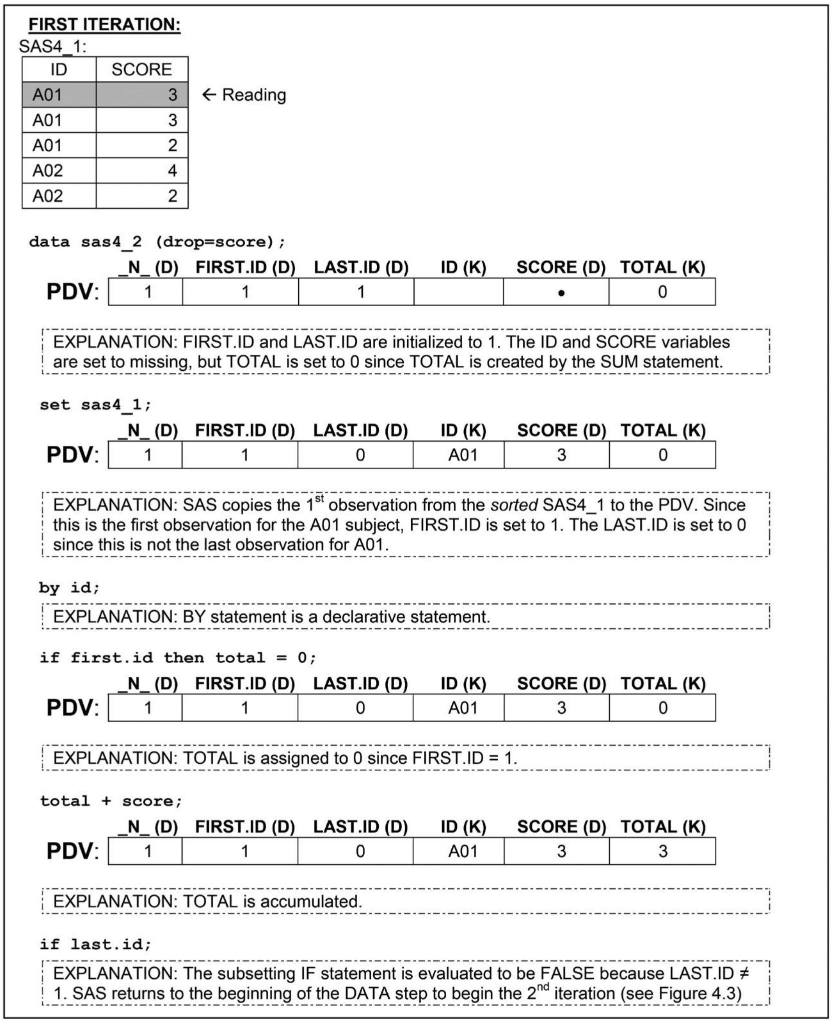
```
proc sort data = sas4_1;
    by id;
run;
data sas4_2 (drop = score);
    set sas4_1;
    by id;
    if first.id then total = 0;
    total + score;/* SUM statement */
    if last.id;
run;

title 'The total scores for each subject';
proc print data = sas4_2;
run;
```

Output from Program 4.1:

```
             The total scores for each subject
                  Obs    ID     total
                   1     A01      8
                   2     A02      6
```

Because the BY statement with ID as the BY variable 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 first iteration of the DATA step execution (see Figure 4.2). ID and SCORE variables are set to missing, but TOTAL is set to 0 because TOTAL is created by the SUM statement. (The SUM statement is the one after the IF-THEN statement.) When the SET statement executes, SAS copies the first observation from the sorted SAS4_1 data set to the PDV. Because this is the first observation for the A01 subject, FIRST.ID is set to 1. The LAST.ID is



FIRST ITERATION:

SAS4_1:

ID	SCORE	
A01	3	← Reading
A01	3	
A01	2	
A02	4	
A02	2	

```
data sas4_2 (drop=score);
```

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

EXPLANATION: 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 sas4_1;
```

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

EXPLANATION: SAS copies the 1st observation from the *sorted* SAS4_1 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;
```

EXPLANATION: BY statement is a declarative statement.

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

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

EXPLANATION: TOTAL is assigned to 0 since FIRST.ID = 1.

```
total + score;
```

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

EXPLANATION: TOTAL is accumulated.

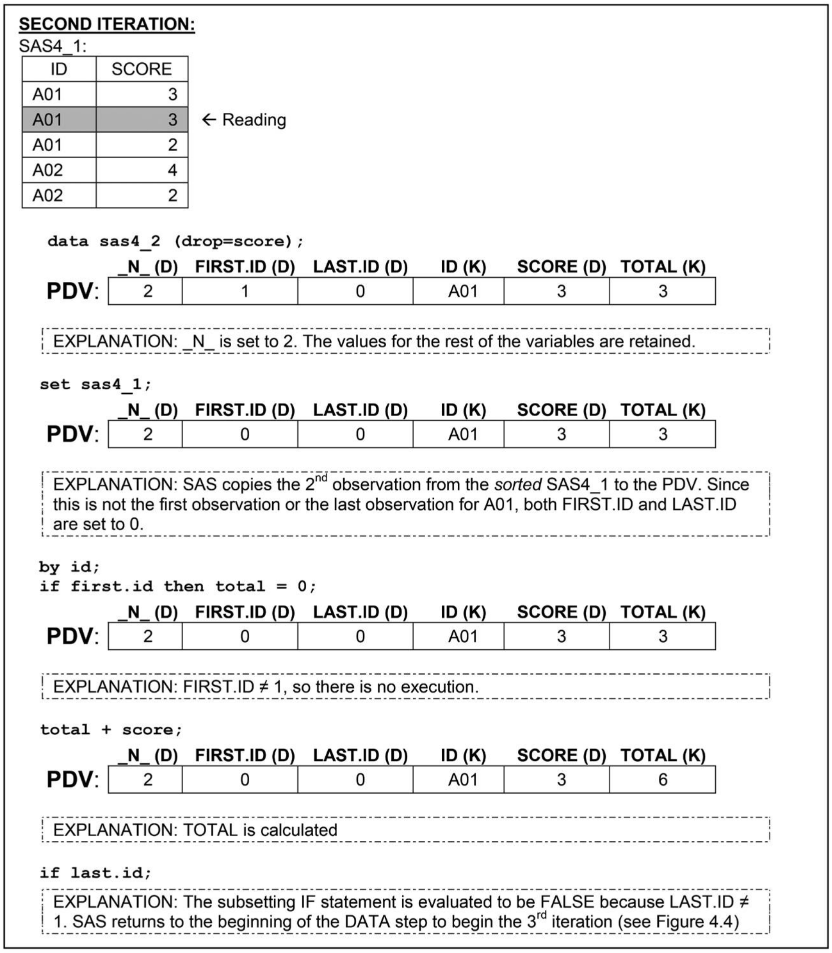
```
if last.id;
```

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 2nd iteration (see Figure 4.3)

FIGURE 4.2
First iteration for Program 4.1.

set to 0 because 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 then total = 0;). Then, 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 sent to the output data set, SAS4_2.

The second iteration (see Figure 4.3) is similar to the first iteration. During this iteration, both FIRST.ID and LAST.ID are set to 0. TOTAL is then



SECOND ITERATION:

SAS4_1:

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

← Reading

```
data sas4_2 (drop=score);
```

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

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

```
set sas4_1;
```

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

EXPLANATION: SAS copies the 2nd observation from the *sorted* SAS4_1 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 then total = 0;
```

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

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

```
total + score;
```

PDV:	_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
	2	0	0	A01	3	6

EXPLANATION: TOTAL is calculated

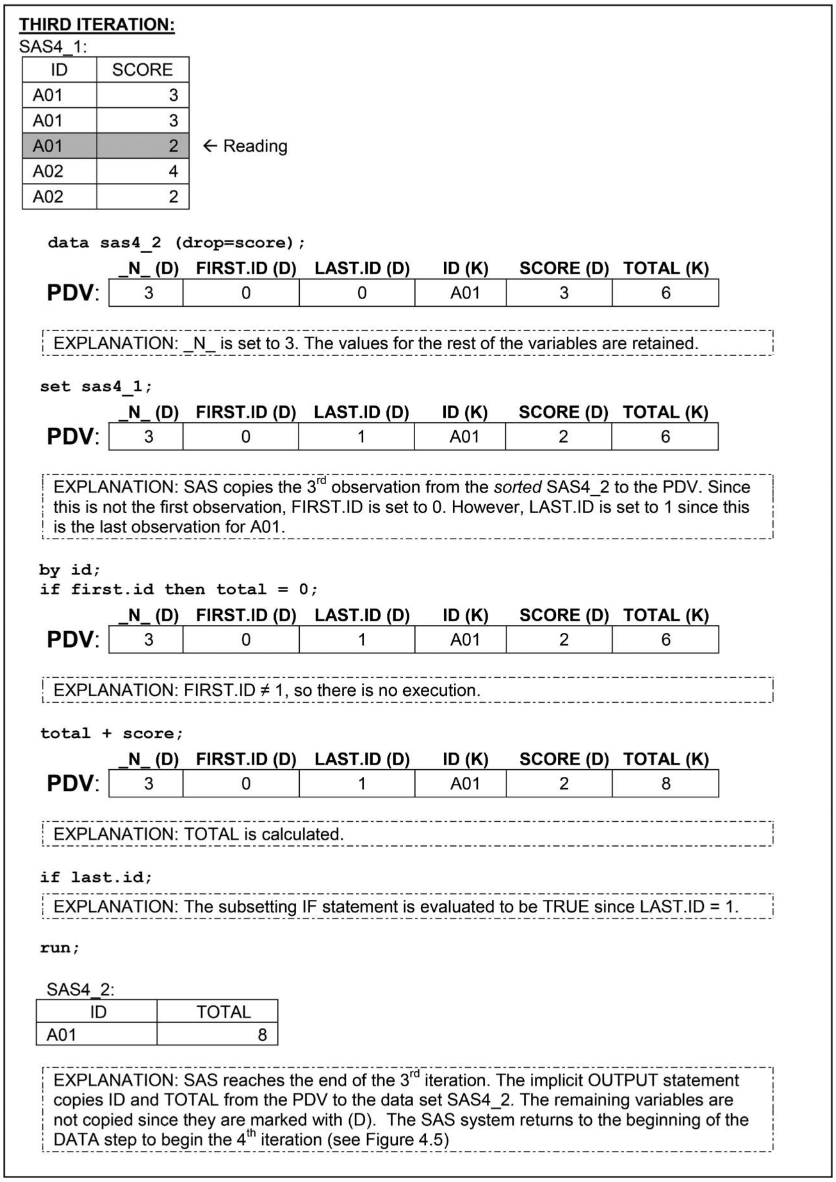
```
if last.id;
```

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 4.4)

FIGURE 4.3
Second iteration for Program 4.1.

accumulated. But again the PDV contents are not output to the SAS data set because this is not the last observation for A01.

In the third iteration (see Figure 4.4), 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 data set sas4_2 (the SCORE variable is dropped in the DATA statement). The rest of the iterations are similar to the iterations explained above. See Figures 4.5 and 4.6 for details. (*Program 4.1* is illustrated in *program4.1.pdf*.)



THIRD ITERATION:

SAS4_1:

ID	SCORE	
A01	3	
A01	3	
A01	2	← Reading
A02	4	
A02	2	

```
data sas4_2 (drop=score);
```

PDV:

_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
3	0	0	A01	3	6

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

```
set sas4_1;
```

PDV:

_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
3	0	1	A01	2	6

EXPLANATION: SAS copies the 3rd observation from the *sorted* SAS4_2 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 then total = 0;
```

PDV:

_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
3	0	1	A01	2	6

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

```
total + score;
```

PDV:

_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
3	0	1	A01	2	8

EXPLANATION: TOTAL is calculated.

```
if last.id;
```

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

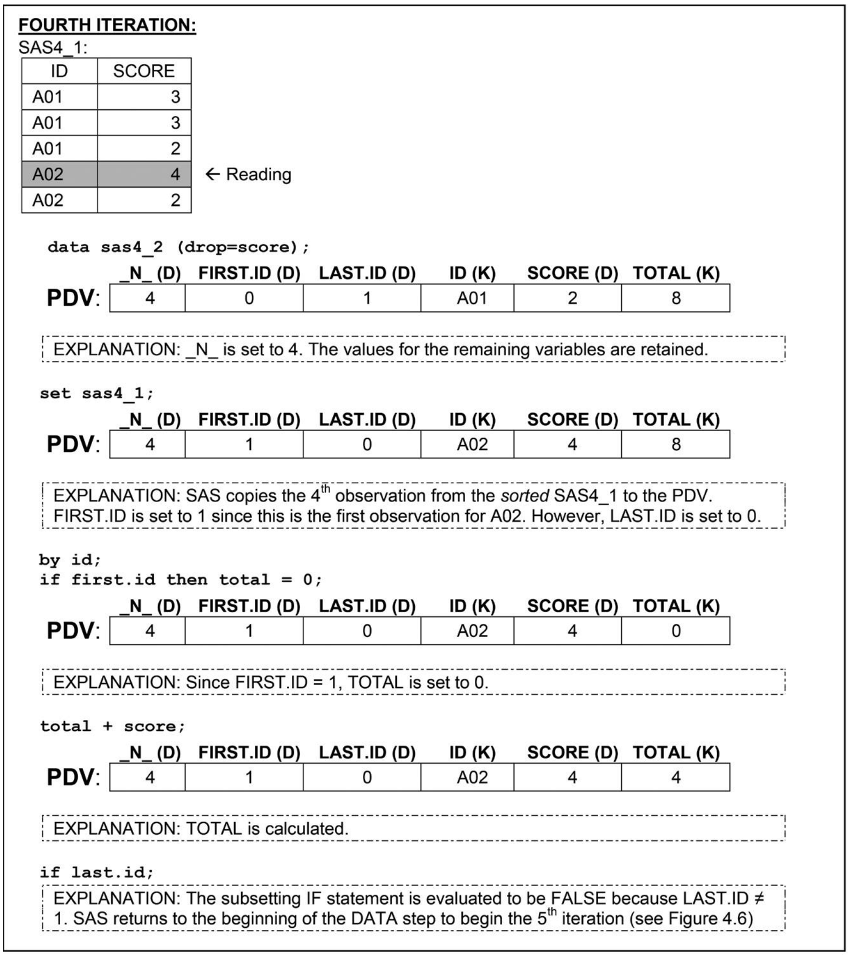
```
run;
```

SAS4_2:

ID	TOTAL
A01	8

EXPLANATION: SAS reaches the end of the 3rd iteration. The implicit OUTPUT statement copies ID and TOTAL from the PDV to the data set SAS4_2. 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 4.5)

FIGURE 4.4
Third iteration for Program 4.1.



FOURTH ITERATION:

SAS4_1:

ID	SCORE	
A01	3	
A01	3	
A01	2	
A02	4	← Reading
A02	2	

```
data sas4_2 (drop=score);
```

	_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	0	1	A01	2	8

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

```
set sas4_1;
```

	_N_ (D)	FIRST.ID (D)	LAST.ID (D)	ID (K)	SCORE (D)	TOTAL (K)
PDV:	4	1	0	A02	4	8

EXPLANATION: SAS copies the 4th observation from the *sorted* SAS4_1 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 then total = 0;
```

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

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

```
total + score;
```

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

EXPLANATION: TOTAL is calculated.

```
if last.id;
```

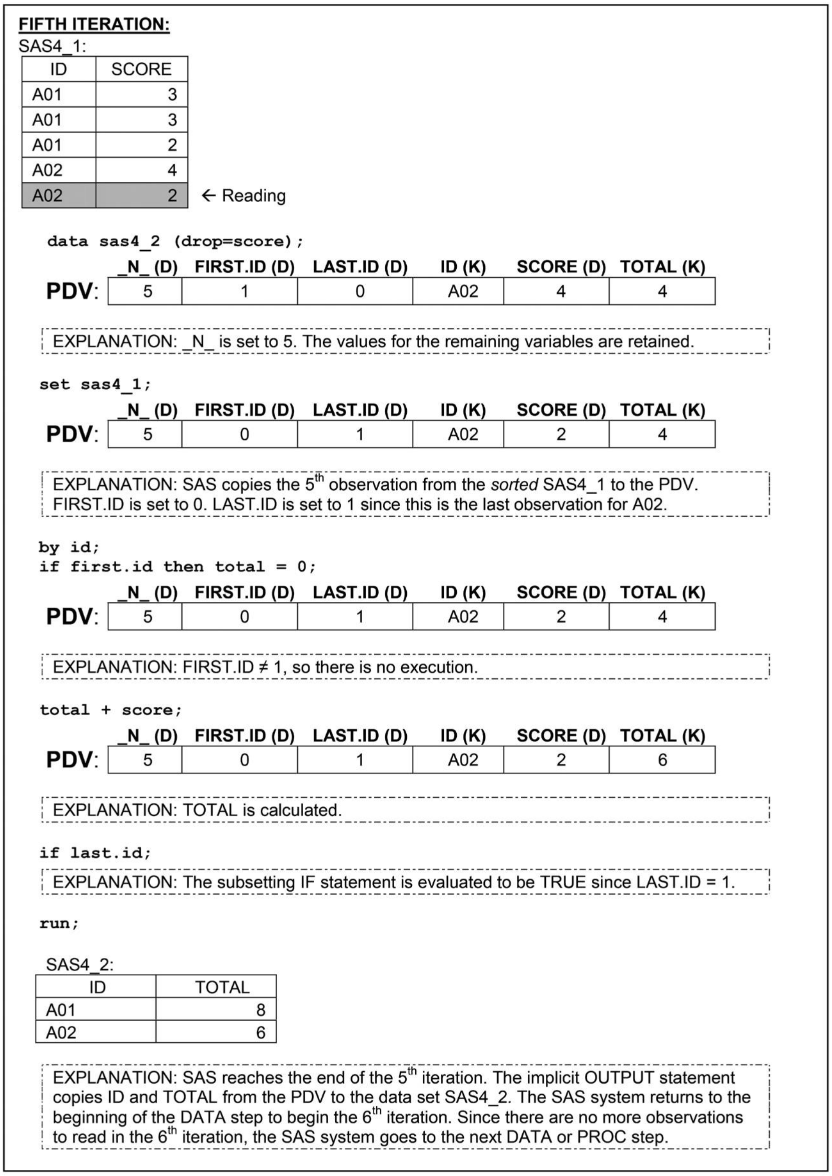
EXPLANATION: The subsetting IF statement is evaluated to be FALSE because LAST.ID ≠ 1. SAS returns to the beginning of the DATA step to begin the 5th iteration (see Figure 4.6)

FIGURE 4.5
Fourth iteration for Program 4.1.

4.2 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 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.



FIFTH ITERATION:

SAS4_1:

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

← Reading

```
data sas4_2 (drop=score);
```

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

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

```
set sas4_1;
```

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

EXPLANATION: SAS copies the 5th observation from the *sorted* SAS4_1 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 then total = 0;
```

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

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

```
total + score;
```

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

EXPLANATION: TOTAL is calculated.

```
if last.id;
```

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

```
run;
```

SAS4_2:

ID	TOTAL
A01	8
A02	6

EXPLANATION: SAS reaches the end of the 5th iteration. The implicit OUTPUT statement copies ID and TOTAL from the PDV to the data set SAS4_2. 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 4.6
Fifth iteration for Program 4.1.

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 output only when the LAST.VARIABLE equals 1.
5. In addition to the BY variable, another variable will 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 4.1 completes only 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).

This section covers some commonly encountered applications that utilize BY-group processing.

4.2.1 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 SAS4_1. One way to accomplish this task is to use the MEANS procedure and use SCORE in the VAR statement and ID in the CLASS statement. Alternatively, you can also use BY-group processing in the DATA step to create a data set that contains the mean score for each subject.

The solution for this problem is similar to the one in Program 4.1. 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 4.2.)

Program 4.2:

```
data sas4_mean (drop = score);
    set sas4_1;
    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 = sas4_mean;
run;
```

Output from Program 4.2:

```
       The mean score for each subject
                                 mean_
       Obs    ID     total  n      score
        1     A01      8    3      2.66667
        2     A02      6    2      3.00000
```

4.2.2 Creating Data Sets with Duplicate or Non-Duplicate Observations

A common task in examining a data set is checking when the data set contains duplicate observations. Again, you can use BY-group processing to identify duplicated observations.

The first two observations in SAS4_1 are identical. Suppose that you would like to create two data sets: one with observations with non-duplicated records and one containing observations with duplicated records from the data set SAS4_1. Because a duplicated record will have the same value for both the ID and SCORE variables, both ID and SCORE variables will be used as the BY variables. A non-duplicated record is the one where both FIRST. SCORE and LAST.SCORE equal 1; otherwise, it will be a duplicated record. The solution for this program is in Program 4.3.

Program 4.3:

```
proc sort data = sas4_1;
    by id score;
run;

data sas4_1_s sas4_1_d;
    set sas4_1;
    by id score;
    if first.score and last.score then output sas4_1_s;
    else output sas4_1_d;
run;

title 'Non-duplicated records';
proc print data = sas4_1_s;
run;

title 'Duplicated records';
proc print data = sas4_1_d;
run;
```

Output from Program 4.3:

```
Non-duplicated records
Obs    ID     SCORE
 1     A01      2
 2     A02      2
 3     A02      4

Duplicated records
Obs    ID     SCORE
 1     A01      3
 2     A01      3
```

4.2.3 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 PATIENT contains the triglyceride (TGL) measurement and smoking status (SMOKE) for patients for different time periods. Notice that some patients have only 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, 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 have missing values for either TGL or SMOKE. In this situation, you need to use the missing variable in the resulting data set for this variable. For instance, the TGL measurement is missing for A03.

PATIENTS:

	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	

Because you need to keep only 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 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. Because 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 4.4.

Program 4.4:

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

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

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

Output from Program 4.4:

```
        The data contains the most current non-missing values
                                                     smoke_
                   Obs     patid     tgl_new          new
                    1       A01        150             Y
                    2       A02        210             N
                    3       A03         .              Y
                    4       A04        190             N
                    5       A05        189
```

4.2.4 Restructuring Data Sets from Long Format to Wide Format

Restructuring data sets from the *wide* format to the *long* format was illustrated in Chapter 3. You can also transform the data set from the *long* format to the *wide* format by using BY-group processing. A solution to solve this problem is illustrated in Program 4.5. A more efficient way to solve this problem is shown in Chapter 6, "Array Processing." Data set LONG from Chapter 3 is used again as input for Program 4.5. This time, WIDE is generated as output.

Program 4.5:

```
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;
run;
```

Program 4.5 begins by sorting the LONG data set by ID and TIME. Sorting the variable TIME within each ID is important because it ensures that the horizontal order of S1–S3 in the WIDE data set for each subject can be matched correctly with the vertical order of SCORE in the LONG data set.

Because you are reading five observations from the LONG data set but creating only two observations, it means that you are *not* copying data from the PDV to the final data set at each iteration. As a matter of fact, you need to generate only one observation once all the observations for each subject have been processed. That means that the newly created variables (S1–S3) in the final data set 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 data set 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. (*Program 4.5* is illustrated in *program4.5.pdf*.)

Exercises

Exercise 4.1. Consider the following data set, PROB4_1.SAS7BDAT:

	ID	TIME	SCORE
1	A	1	3
2	A	2	.
3	A	3	4
4	B	1	.
5	B	2	5
6	B	3	.

The SCORE variable is recorded at three different time points for each subject. For this exercise, you need to modify the SCORE variable. If SCORE is missing for the current observation, use the SCORE value from the previous recorded time point within each ID.

The resulting data will look as shown below:

	ID	TIME	SCORE
1	A	1	3
2	A	2	3
3	A	3	4
4	B	1	.
5	B	2	5
6	B	3	5

Exercise 4.2. Program 4.4 creates a data set that contains the most recent non-missing data within each BY group. Modify this program by adding two additional variables, TGL_YEAR and SMOKE_YEAR, that contain the corresponding year when the most recent non-missing data were measured. The resulting data will look like as shown below:

	PATID	**TGL_NEW**	**SMOKE_NEW**	**TGL_YEAR**	**SMOKE_YEAR**
1	A01	150	Y	2007	2005
2	A02	210	N	2006	2006
3	A03	.	Y	.	2005
4	A04	190	N	2006	2007
5	A05	189		2005	.

Exercise 4.3. Use the same data set from Exercise 4.2 (PATIENTS. SAS7BDAT) to create a new SAS data set that contains only the first two visits of each patient. If a patient has only one visit, you will need to keep only this one observation for this patient. The resulting data will look as shown below:

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