3

Understanding How the DATA Step Works

3.1 DATA Step Processing Overview

A common befuddlement often facing beginning SAS® programmers is that the SAS data set that they create is not what they intended to create—that is, there are more or less observations than intended or the value of the newly created variable is not retained correctly. These types of mistakes occur because new programmers often focus exclusively on language syntax but fail to understand how the DATA step actually works.

The purpose of this chapter is to guide you through how DATA step programming operates, step by step, by way of providing various examples. The material in this chapter is based on a paper that I presented at the Western Users of SAS Software conference (2008).

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

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

- LENGTH: setting the internal variable length
- FORMAT: setting the variable output format
- LABEL: defining variable labels

- DROP: indicating which variables are to be omitted in the output file
- KEEP: indicating which variables are to be included in the output file

In contrast to declarative statements, the order in which executable statements appear in the DATA step matters greatly. For example, to read an external text file, you need to start with the INFILE statement, followed by the INPUT statement. The INFILE statement is used to identify the location of the external file, and the INPUT statement instructs SAS how to read each observation. Thus, you must place the INFILE statement before the INPUT statement because SAS needs to know where to find the external file *before* it can read it.

Program 3.1 illustrates how DATA step processing works. This program reads raw data from a text file, EXAMPLE3_1.TXT, and creates one variable, BMI.

EXAMPLE3_1.TXT contains two observations and three variables, NAME (columns 1–7), HEIGHT (columns 9–10), and WEIGHT (columns 12–14). Notice that the WEIGHT variable for the first observation is entered as "12D", which is a data entry error. Since each variable is occupied in a fixed field and the values for these variables are standard character or numerical values, the column input method is best used to read the raw data set.

EXAMPLE3_1.TXT:

```
12345678901234567890123456789O
Barbara  61 12D
John     62 175
```

Program 3.1:

```
data ex3_1;
    infile 'W:\SAS Book\dat\example3_1.txt';
    input name $ 1-7 height 9-10 weight 12-14;
    BMI = 700*weight/(height*height);
    output;
run;
```

Log from Program 3.1:

```
1 data ex3_1;
2      infile 'W:\SAS Book\dat\example3_1.txt';
3      input name $ 1-7 height 9-10 weight 12-14;
4      BMI = 700*weight/(height*height);
5      output;
6 run;
NOTE: The infile 'W:\SAS Book\dat\example3_1.txt' is:
       Filename = W:\SAS Book\dat\example3_1.txt,
       RECFM = V,LRECL = 256,File Size (bytes) = 32,
```

```
       Last Modified = 15Mar2012:09:10:03,
       Create Time = 15Mar2012:09:09:22
NOTE: Invalid data for weight in line 1 12-14.
RULE:— — +— — 1— — +— — 2— — +— — 3— — +— — 4— — +— — 5— —
+— — 6
1    Barbara 61 12D 14
name = Barbara height = 61 weight =. BMI =. _ERROR_ = 1 _N_ = 1
NOTE: 2 records were read from the infile 'W:\SAS Book\dat\
example3_1.txt'.
     The minimum record length was 14.
     The maximum record length was 14.
NOTE: Missing values were generated as a result of performing
an operation on missing values.
     Each place is given by: (Number of times) at
(Line):(Column).
     1 at 4:14
NOTE: The data set WORK.EX3_1 has 2 observations and 4
variables.
NOTE: DATA statement used (Total process time):
     real time    0.15 seconds
     cpu time     0.03 seconds
```

3.1.1 DATA Step Compilation Phase

Since Program 3.1 reads in a raw data set, the input buffer is created at the beginning of the compilation phase. The input buffer is used to hold raw data (Figure 3.1). However, if you read in a SAS data set instead of a raw data file, the input buffer will not be created.

SAS also creates the program data vector (PDV) in the compilation phase (Figure 3.1). SAS uses the PDV, a memory area on your computer, to build the new data set. There are two automatic variables, _N_ and _ERROR_, inside the PDV. _N_ equaling 1 indicates the first observation is being processed, _N_ equaling 2 indicates the second observation is being processed, and so on. The automatic variable _ERROR_ is an indicator variable with values of 1 or 0. _ERROR_ equaling 1 signals the data error of the currently processed observation, such as reading the data with an incorrect data type. In addition to the two automatic variables, there is one space allocated for

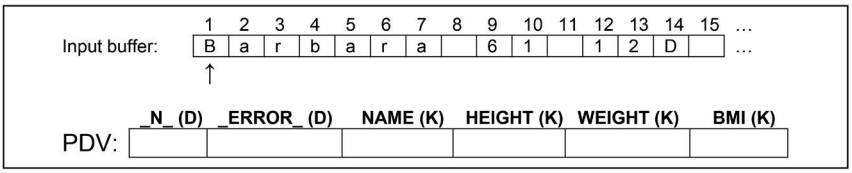

FIGURE 3.1
Input buffer and the program data vector (PDV).

each of the variables that will be created from this DATA step. The variables NAME, HEIGHT, and WEIGHT are read in from an external file, whereas the newly created BMI variable (body mass index) is derived from HEIGHT and WEIGHT.

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

During the compilation phase, SAS checks for syntax errors, such as invalid variable names, options, punctuations, misspelled keywords, etc. SAS also identifies the type and length of the newly created variables.

At the end of the compilation phase, the descriptor portion of the SAS data set is created, which includes the data set name, the number of observations, and the number, names, and attributes of variables. The information about the descriptor portion can be generated via the CONTENTS procedure.

3.1.2 DATA Step 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 nonautomatic variables are set to missing. Once the INFILE statement identifies the location of the input file, the INPUT statement copies the first data line into the input buffer. Then the INPUT statement reads data values from the record in the input buffer according to instructions from the INPUT statement and writes them to the PDV. The values for NAME and HEIGHT are successfully copied from the input buffer to the PDV. However, the value for WEIGHT is "12D", an invalid numeric value that causes _ERROR_ to be set to 1 and WEIGHT to missing. Meanwhile, an error message is sent to the SAS log indicating the location of the data error (see Log from Program 3.1). Next, the assignment statement is executed and BMI will remain missing since operations on a missing value will result in a missing value.

When the OUTPUT statement is executed, only the values from the PDV marked with (K) are copied as a single observation to the output SAS data set, EX3_1. See Figure 3.2 for a detailed explanation of each step in the first iteration.

At the end of the DATA step the SAS system returns to the beginning of the DATA step to begin the next iteration. The values of the variables in the PDV are reset to missing. The automatic variable _N_ is incremented to 2, and _ERROR_ is set to 0. The second data line is read into the input buffer by the INPUT statement. See the illustration in Figure 3.3 for details.

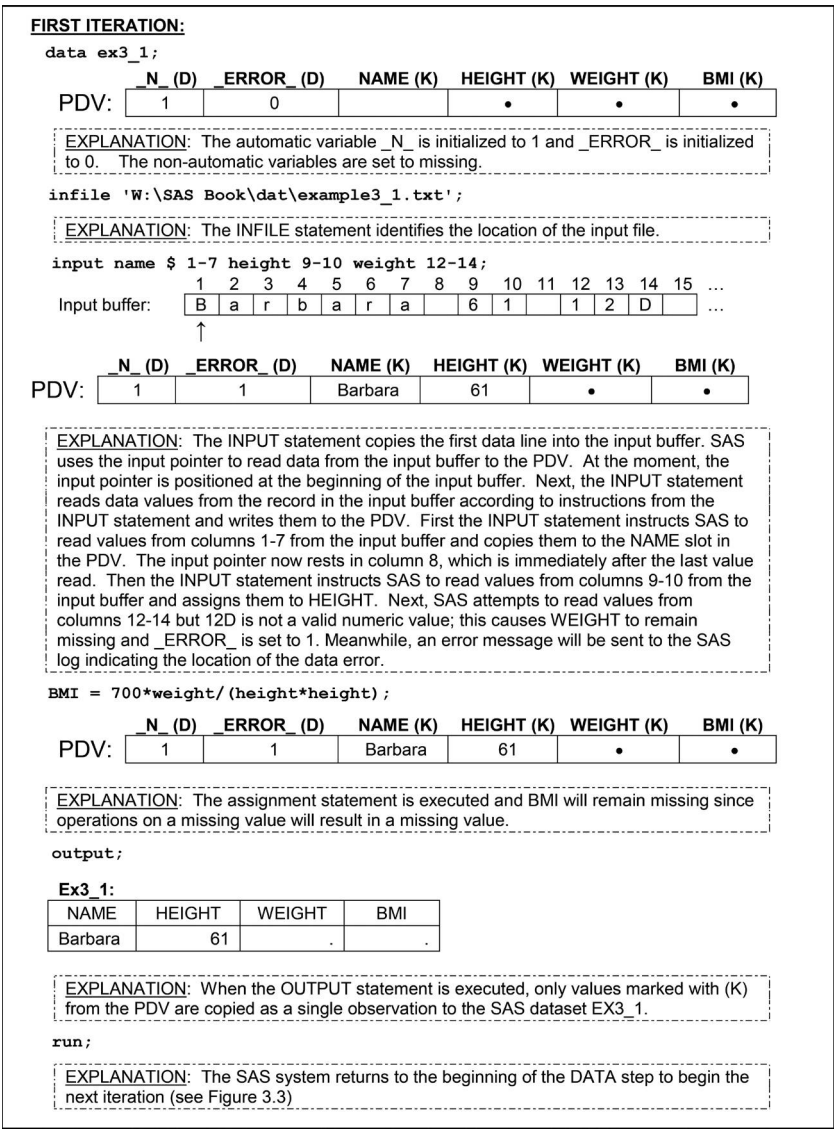

FIGURE 3.2
The first iteration of Program 3.1.

At the end of the DATA step for the second iteration, the SAS system again returns to the beginning of the DATA step to begin the next iteration (see Figure 3.4). The values of the variables in the PDV are reset to missing. The automatic variable _N_ is incremented to 3. SAS attempts to read an observation from the input data set, but it reaches the end-of-file-marker, which means

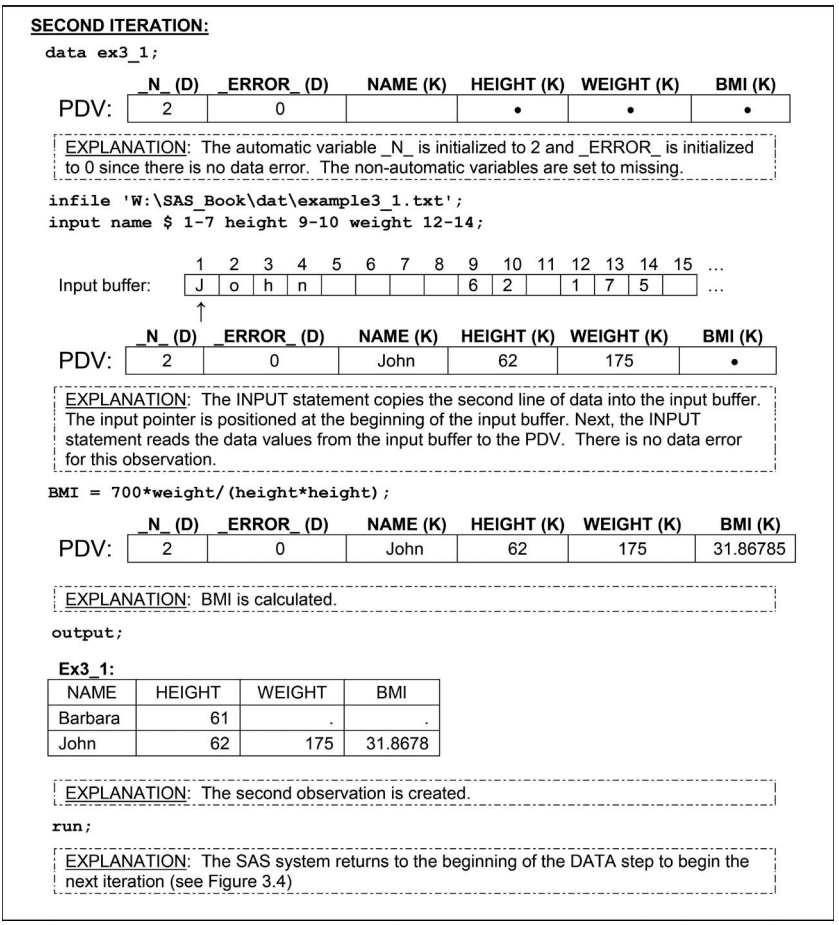

FIGURE 3.3
The second iteration of Program 3.1.

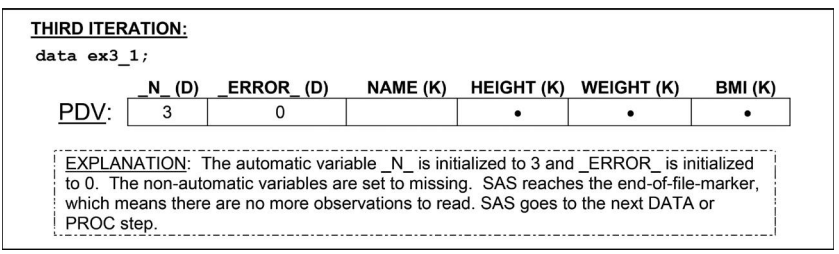

FIGURE 3.4
The third iteration of Program 3.1.

that there are no more observations to read. When an end-of-file marker is encountered, SAS goes to the next DATA or PROC step in the program. (*Program 3.1* is illustrated in *program3.1.pdf*.)

3.1.3 The Importance of the OUTPUT Statement

In Program 3.1, an explicit OUTPUT statement is used to tell SAS to write the current observation from the PDV to a SAS data set immediately. An OUTPUT statement is not actually needed in Program 3.1 because the contents of the PDV are written out by default with an *implicit* OUTPUT statement at the end of the DATA step—exactly where the explicit OUTPUT statement has been placed. In Program 3.2, the explicit OUTPUT statement has been removed so that you can see that the implicit OUTPUT statement works just like its explicit counterpart.

Program 3.2:

```
data ex3_1;
    infile 'W:\SAS Book\dat\example3_1.txt';
    input name $ 1-7 height 9-10 weight 12-14;
    BMI = 700*weight/(height*height);
run;
```

If you place an explicit OUTPUT statement in a DATA step, the explicit OUTPUT statement will override the implicit OUTPUT statement; in other words, once an explicit OUTPUT statement is used to write an observation to an output data set, there is no longer an implicit OUTPUT statement at the end of the DATA step. The SAS system adds an observation to the output data set only when an explicit OUTPUT statement is executed. Furthermore, more than one OUTPUT statement in the DATA step can be used. You will see an example of using multiple OUTPUT statements later in this chapter.

3.1.4 The Difference between Reading a Raw Data Set and a SAS Data Set

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

Often an output data set is created based on an existing SAS data set instead of reading in a raw data file. In this instance, SAS sets each variable to missing in the PDV *only* before the first iteration of the execution. Variables will retain their values in the PDV until they are replaced by the new values

from the input data set. These variables exist in both the input and output data sets. Often when creating a new data set based on the existing SAS data set, you will create new variables based on existing variables; these new variables are not from the input data set. These new variables will be set to missing in the PDV at the beginning of every iteration of the execution.

3.2 Retaining the Value of Newly Created Variables

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

3.2.1 The RETAIN Statement

Suppose you would like to create a new variable based on values from previous observations, such as creating a variable that accumulates the values from other numeric variables. Consider the following SAS data set, SAS3_1. Based on SAS3_1, suppose you would like to create a new variable, TOTAL, that is used to accumulate the SCORE variable.

SAS3_1:

	ID	SCORE
1	A01	3
2	A02	.
3	A03	4

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

```
RETAIN variable <value>;
```

In the RETAIN statement, *variable* is the name of the variable that you will want to retain, and *value* is a numeric value that is used to initialize the *variable* only at the first iteration of the DATA step execution. If you do not specify an initial value, the retained variable is initialized as missing before

the first execution of the DATA step. The RETAIN statement prevents the *variable* from being initialized each time the DATA step executes. Program 3.3 uses the RETAIN statement to create the TOTAL variable.

The RETAIN statement is a declarative statement and does not execute during the DATA step execution phase. Where you place the RETAIN statement within the DATA step will not affect the value assigned to TOTAL, but where the statement appears *will* affect the column position of the TOTAL variable. For example, placing the RETAIN statement after the SET statement will generate an output data set starting with ID and SCORE and followed by TOTAL because the compiler encounters ID and SCORE in the SET statement before it gets to TOTAL in the RETAIN statement. If you want the resulting data set to begin with the TOTAL variable, you need to place the RETAIN statement, which contains the TOTAL variable, before the SET statement.

Program 3.3:

```
data ex3_2;
     set sas3_1;
     retain total 0;
     total = sum(total, score);
run;

title 'Creating TOTAL by accumulating SCORE';
proc print data = ex3_2;
run;
```

Output from Program 3.3:

```
Creating TOTAL by accumulating SCORE
  Obs       ID      score    total
   1        A01       3        3
   2        A02       .        3
   3        A03       4        7
```

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 3.5); however, the variable TOTAL is initialized to 0 because of the RETAIN statement. Next, the SET statement copies the first observation from the data set SAS3_1 to the PDV. The RETAIN statement is a compile-time only statement; it does not execute during the execution phase. Only then will the variable TOTAL be calculated. Finally, the DATA step execution reaches the final step. Because there is no explicit OUTPUT statement in this program, the implicit OUTPUT statement at the end of the DATA step tells the SAS system to write observations to the data set. The SAS system returns to the beginning of the DATA step to begin the second iteration.

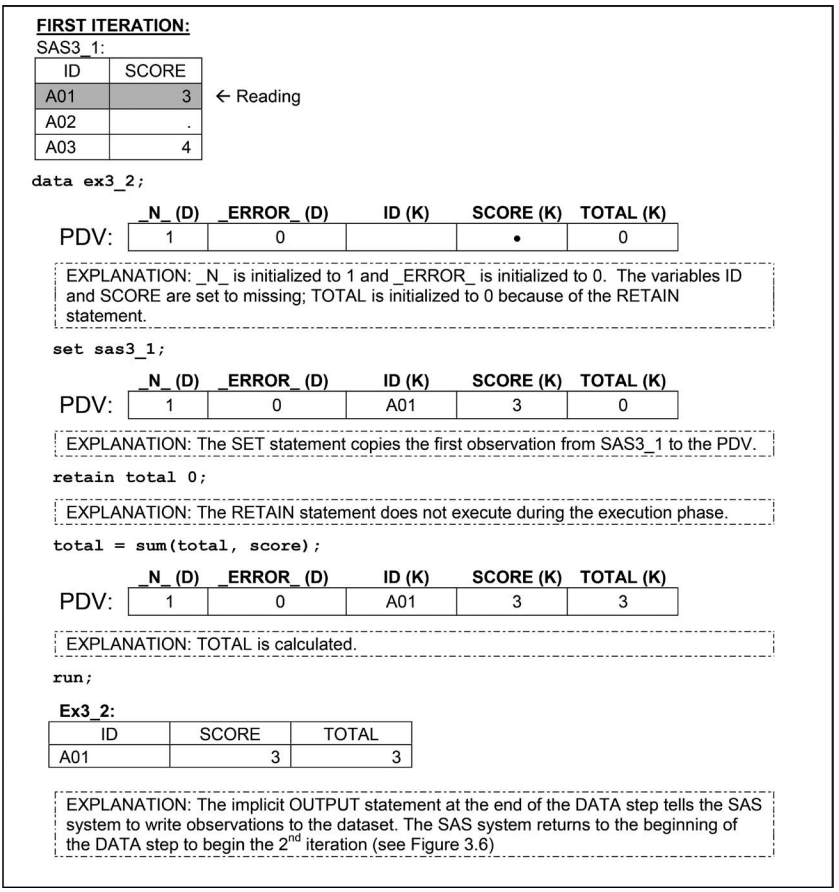

FIGURE 3.5
The first iteration of Program 3.3.

At the beginning of the second iteration, because data is read from an existing SAS data set, instead of reading from the raw data set, 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 3.6 and 3.7 for the processes of the second and third iterations. (*Program 3.3* is illustrated in *program3.3.pdf*.)

3.2.2 The SUM Statement

Program 3.3 can be rewritten by using the SUM statement instead of using the RETAIN statement. The SUM statement has the following form:

```
variable+expression;
```

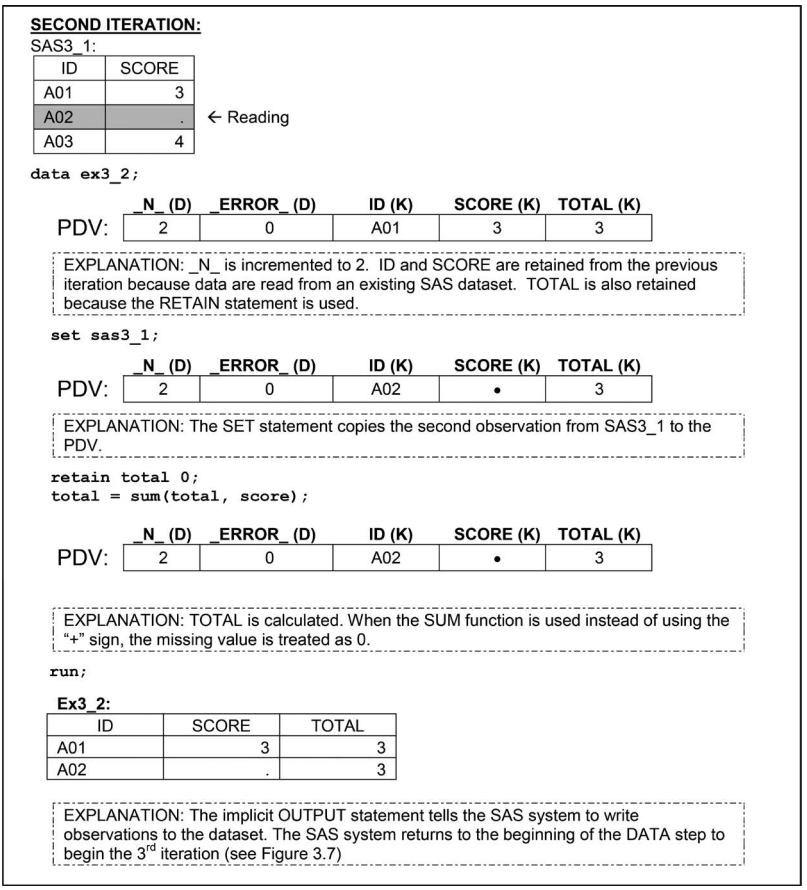

FIGURE 3.6
The second iteration of Program 3.3.

The SUM statement may seem unusual because it does not contain the equal sign. In the SUM statement, *variable* is the numeric accumulator variable that is to be created and is automatically set to 0 at the beginning of the first iteration of the DATA step execution (and is thus retained in following iterations). *Expression* is any SAS expression. In a situation where e*xpression* is evaluated to a missing value, it is treated as 0. Program 3.4 is an equivalent version of Program 3.3 using the SUM statement.

Program 3.4:

```
data ex3_3;
    set sas3_1;
    total + score;
run;
```

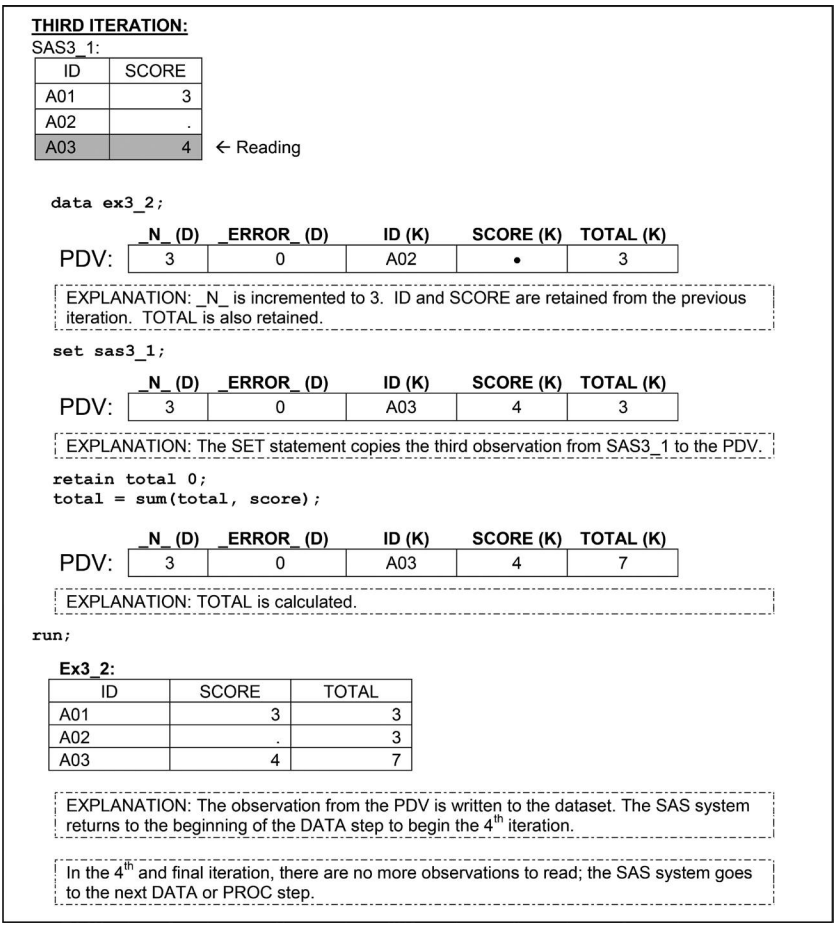

FIGURE 3.7
The third iteration of Program 3.3.

3.3 Conditional Processing in the DATA Step

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

3.3.1 The Subsetting IF Statement

The subsetting IF statement has the following form:

```
IF expression;
```

The *expression* associated with the subsetting IF statement can be any valid SAS expression. If the *expression* is true for the observation, SAS continues to execute statements in the DATA step and includes the current observation in the data set. The resulting SAS data set contains a subset of the external file or SAS data set. On the other hand, if the *expression* is false, then no further statements are processed for that observation and SAS immediately returns to the beginning of the DATA step. 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.

For example, Program 3.5 creates a data set that contains only the observations in which the SCORE variable is not missing.

Program 3.5:

```
data ex3_4;
    set sas3_1;
    total + score;
    if not missing(score);
run;

title 'Keep observations only when SCORE is not missing';
proc print data = ex3_4;
run;
```

Output from Program 3.5:

```
        Keeping observations for SCORE is not missing
             Obs        ID        score      total
              1         A01         3          3
              2         A03         4          7
```

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

```
IF expression THEN DELETE;
```

Program 3.6 provides an alternative version of Program 3.5 by utilizing the subsetting IF and DELETE statements.

Program 3.6:

```
data ex3_5;
    set sas3_1;
    total + score;
    if missing(score) then delete;
run;
```

3.3.2 Detecting the End of a Data Set by Using the END= Option

Sometimes you may want to create a data set that contains only the contents from the PDV when reading the last observation from the input data set. You can create a temporary variable by using the END= option in the SET statement as a flag to signal when the last observation is being read. The END= option has the following form:

```
SET SAS-data-set END= variable;
```

The *variable* after the keyword END= is a temporary variable that contains an end-of-file indicator. The *variable* is initialized to zero at the beginning of the DATA step iteration and is set to 1 when the SET statement reads the last observation of the input data set. Since the *variable* is a temporary variable, it is not added to the output data set.

Program 3.7 calculates the total score and lists the total number of observations from the data set Sas3_1.

Program 3.7:

```
data total_score(keep = total n);
    set sas3_1 end = last;
    total + score;
    n + 1;
    if last;
run;

title 'Only keep the last observation';
proc print data = total_score;
run;
```

Output from Program 3.7:

```
Only keep the last observation
    Obs       total     n
     1          7       3
```

3.3.3 Restructuring Data Sets from Wide Format to Long Format

Restructuring data sets 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 ARRAY processing described in Chapter 6 or the TRANSPOSE procedure described in Chapter 10. However, this can also be

accomplished without advanced techniques for simple cases. The example in this chapter is adapted from *Applied Statistics and the SAS® Programming Language* (Cody and Smith, 1991).

Suppose that you are transforming data from the *wide* format (data set WIDE) to the *long* format (data set LONG). Notice that data in the *long* format has a variable, TIME, that distinguishes 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 data set, there will be only two iterations for the DATA step processing. That means you need to generate the output up to three times for each iteration. In some iterations, the output might not be generated three times because missing values in variables S1–S3 will not be output in the LONG data set. Program 3.8 illustrates the data transformation by using multiple OUTPUT statements in one DATA step.

Program 3.8:

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

In Program 3.8, 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 data set 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 data set. Next, assign value 2 to the TIME variable and assign the value from S2 to SCORE and output the data set again. Similar processes are repeated; assign 3 to TIME, assign S3 to SCORE, and then 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 output to the final data set. See Figure 3.8 for more details. Note that the automatic variable _ERROR_ is not shown in the figure for simplicity purposes.

The second iteration (see Figure 3.9) 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 data set because SCORE equals missing. (*Program 3.8* is illustrated in *program3.8.pdf*.)

3.4 Debugging Techniques

Programming errors can be categorized as either syntax or logic errors. Syntax errors are often easier to detect than logic errors since SAS not only stops programs due to syntax errors but also generates detailed error messages in the log window. On the other hand, logic errors often result in generating an unintended data set and they are difficult to debug. This section covers the two most useful debugging strategies for detecting logic errors: utilizing the PUT statement in the DATA step and using the DATA step debugger.

3.4.1 Using the PUT Statement to Observe the Contents of the PDV

A logic error is often created due to a lack of knowing the contents of the PDV during the DATA step execution. If you are not sure what the contents of the PDV are during each step of the DATA step processing, use a PUT statement inside the DATA step, which will generate the contents of each variable in the PDV on the SAS log. This strategy was introduced in the *Longitudinal Data and SAS® A Programmer's Guide* (Cody, 2005). The PUT statement has the following form:

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

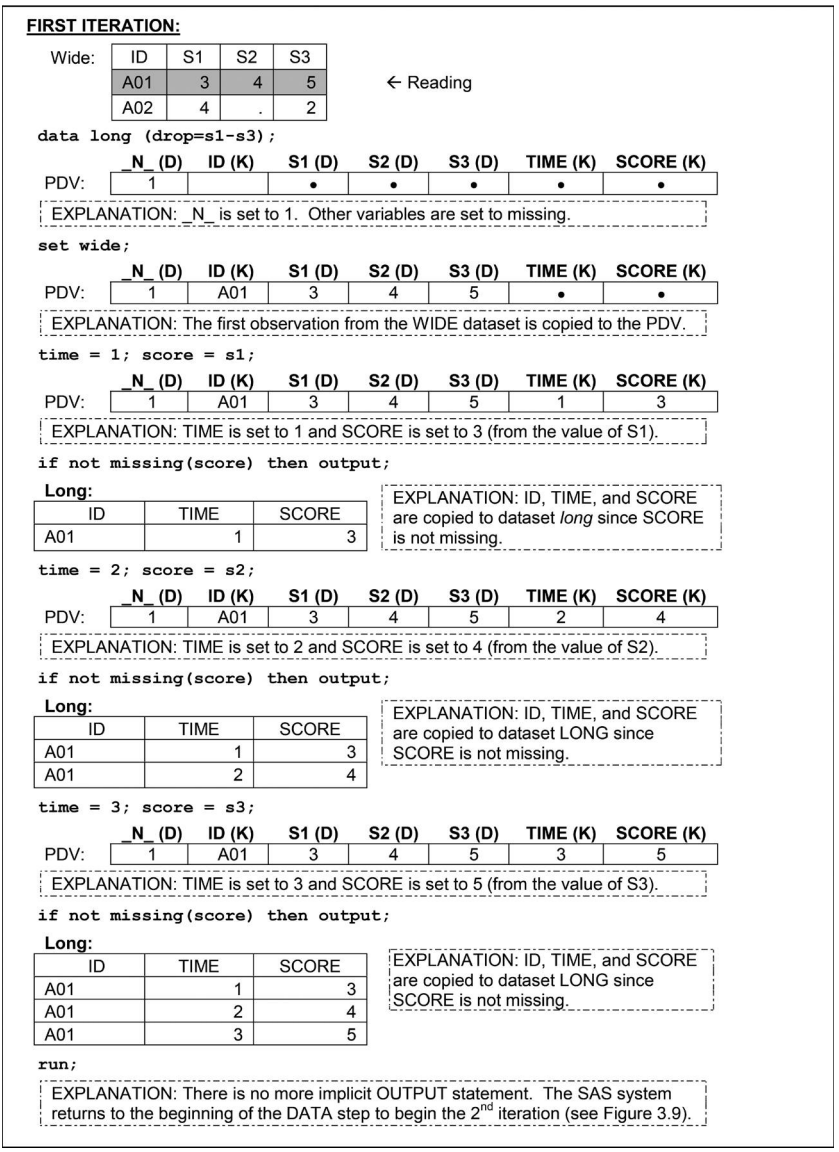

FIGURE 3.8
First iteration for Program 3.8.

The PUT statement can combine explanatory text strings in quotes with the values for selected variables and write the compound output to the SAS log. The keyword _ALL_ (which is an example of the variable-list; see Chapter 1 for details) means that all the variables, including the automatic variables, will be output to the SAS log. For example, Program 3.9 uses the PUT

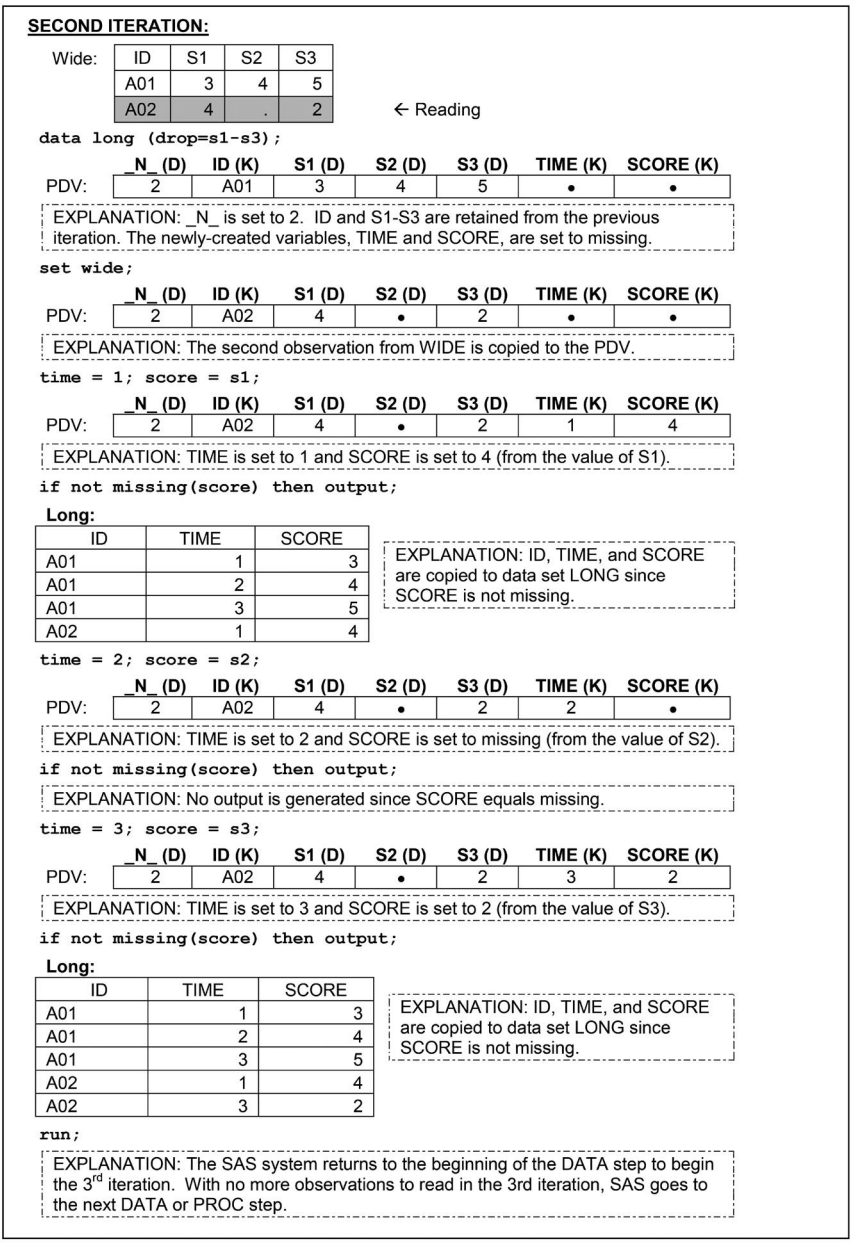

FIGURE 3.9
Second iteration for Program 3.8.

statement to send the contents in the PDV to the SAS log after each statement is executed in the DATA step.

Program 3.9:

```
data ex3_4;
    put "1st PUT" _all_;
    set sas3_1;
    put "2nd PUT" _all_;
    total + score;
    put "3rd PUT" _all_;
    if not missing(score);
    put "4th PUT" _all_;
run;
```

Log from Program 3.9:

```
68 data ex3_4;
69   put "1st PUT" _all_;
70   set sas3_1;
71   put "2nd PUT" _all_;
72   total + score;
73   put "3rd PUT" _all_;
74   if not missing(score);
75   put "4th PUT" _all_;
76 run;


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

3.4.2 Using the DATA Step Debugger

SAS DATA step debugger is a part of Base SAS software and is described in the "DATA Step Debugger" article of SAS documentation.

You can use the DATA step debugger to identify logic errors by examining the contents of the PDV interactively while executing one DATA step at a time. The DATA step debugger cannot be used for a PROC step. To invoke the DATA step debugger, you need to add the DEBUG option to the DATA statement. Program 3.10 illustrates how to use the DATA step debugger.

Program 3.10:

```
data ex3_4/debug;
    set sas3_1;
    total + score;
    if not missing(score);
run;
```

Submitting a DATA step that contains the DEBUG option will invoke the DEBUGGER LOG and DEBUGGER SOURCE windows. You can enter the debugger commands after the prompt (>) in the last line of the DEBUGGER LOG. All the generated results from the debugger commands will be recorded in the DEBUGGER LOG window. DATA step debugger contains numerous commands that allow you to execute, bypass, or suspend one or more statements, examines the values of selected variables in the PDV at any point of execution, displays the attributes of selected variables, and so on. Only a small number of commands are introduced in this section.

In the DEBUGGER SOURCE window, the line numbers that correspond to the program are the same as the ones in the SAS log window. The DEBUGGER SOURCE window enables you to view the position in the DATA step while you debug your program. The DATA step execution pauses before the execution of the statement that is being highlighted. Notice that the current highlighted statement is the SET statement, which is immediately the next statement to be executed. At this point, to display the contents in the PDV, you can use the EXAMINE debugger command, which has the following form:

```
EXAMINE _ALL_ <format> | variable-1 <format-1> <...variable-n
        <format-n>>
```

In the EXAMINE command, you can either use the keyword _ALL_ to display the contents of all variables or only the selected variables in the PDV. The *format* option can be used to display the variables in either SAS built-in

or user-defined formats. For example, to display the contents of all variables in the PDV, you can submit the following command:

```
examine _all_
```

Here are the generated results from the submitted statement above:

Debugger Log:

```
Stopped at line 30 column 5
> examine _all_
ID =
score =.
total = 0
_ERROR_ = 0
_N_ = 1
```

If you want to examine the content for only one variable, for example, TOTAL, you can type the following command line:

```
examine total
```

In the debugging mode, you can execute the DATA step either one statement or several statements at a time by using the STEP command. The general form of the STEP command is as follows:

```
STEP <n>
```

The optional *n* in the STEP command is used to specify the number of statements to execute. If you want to execute only one statement at the moment, you can simply hit the ENTER key because by default the STEP command is associated with the ENTER key.

Another useful debugging command is the BREAK command, which is used to suspend the execution of a program at an executable statement. The BREAK command has the following form:

```
BREAK location <WHEN expression>
```

You can specify a line number at the *location* at which to set a breakpoint. The BREAK command is often used with the GO command. The GO command is used to start or resume the execution of the DATA step. For example, if you want to execute the DATA step until it reaches line 47, you can submit the following two statements:

```
break 47
go
```

In some situations, stopping at a certain line in the DATA step at each iteration of the DATA step execution might not be efficient, especially when you are reading a large data set. Thus, you can utilize the WHEN *expression* in the BREAK command to set up a breakpoint when a certain condition is met. For example, the following command set a breakpoint at line 47 when the SCORE variable is missing:

```
break 47 when score =.
```

To stop the debugging mode of the DATA step execution, you can use the QUIT command:

```
QUIT
```

Submitting the QUIT command terminates a debugger session and returns control to the SAS session.

Exercises

Exercise 3.1. Consider the following data set, PROB3_1.SAS7BDAT:

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

Notice that the SCORE variable contains missing values for some observations. 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 observation. The resulting data will look as shown below:

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

Exercise 3.2. Consider the following data set, PROB3_2.SAS7BDAT:

	ID	GROUP	S1	S2	S3
1	A01	A	3	4	5
2	A01	B	2	3	9
3	A02	A	4	.	2
4	A02	B	4	5	3

Restructure PROB3_2.SAS7BDAT to the format exactly like as shown below. The variable MEASURE can be created by using the CATS function, which is used to concatenate character strings. You can find out how to use the CATS function in Chapter 9 or in the "CATS Function" article of SAS documentation.

	ID	MEASURE	SCORE
1	A01	A1	3
2	A01	A2	4
3	A01	A3	5
4	A01	B1	2
5	A01	B2	3
6	A01	B3	9
7	A02	A1	4
8	A02	A3	2
9	A02	B1	4
10	A02	B2	5
11	A02	B3	3