SAS Certification Prep Guide: Base Programming for SAS 9, Third Edition

by SAS Institute

Chapter 20: Creating Multiple Observations from a Single Record

Overview

Introduction

Sometimes raw data files contain data for several observations in one record. Data can be stored in this manner to reduce the size of the entire data file.

Each record can contain

- repeating blocks of data that represent separate observations

```
1---+----10---+----20---+----30--
01APR90  68  02APR90  67  03APR90  70
04APR90  74  05APR90  72  06APR90  73
07APR90  71  08APR90  75  09APR90  76
```

Figure 20.1: Repeating Blocks of Data

- an ID field followed by an equal number of repeating fields that represent separate observations

```
1---+----10---+----20---+----30--
001 WALKING AEROBICS CYCLING
002 SWIMMING CYCLING SKIING
003 TENNIS SWIMMING AEROBICS
```

Figure 20.2: ID Field with Repeating Fields

- an ID field followed by a varying number of repeating fields that represent separate observations.

```
1---+----10---+----20---+----30--
001 WALKING
002 SWIMMING CYCLING SKIING
003 TENNIS SWIMMING
```

Figure 20.3: ID Field with Varying Number of Repeating Fields

This chapter shows you several ways of creating multiple observations from a single record.

Objectives

In this chapter, you learn to

- create multiple observations from a single record that contains repeating blocks of data
- create multiple observations from a single record that contains one ID field followed by the *same* number of repeating fields
- create multiple observations from a single record that contains one ID field followed by a *varying* number of repeating fields.

Additionally, you learn to

- hold the current record across iterations of the DATA step
- hold the current record for the next INPUT statement
- execute SAS statements based on a variable's value
- explicitly write an observation to a data set
- execute SAS statements while a condition is true.

Reading Repeating Blocks of Data

Overview

Each record in the file Tempdata contains three blocks of data. Each block contains a date followed by the day's high temperature in a small city located in the southern United States.

Raw Data File Tempdata

```
1---+----10---+----20---+----30--
01APR90 68 02APR90 67 03APR90 70
04APR90 74 05APR90 72 06APR90 73
07APR90 71 08APR90 75 09APR90 76
10APR90 78 11APR90 70 12APR90 69
13APR90 71 14APR90 72 15APR90 74
16APR90 73 17APR90 71 18APR90 75
19APR90 75 20APR90 73 21APR90 75
22APR90 77 23APR90 78 24APR90 80
25APR90 78 26APR90 77 27APR90 79
28APR90 81 29APR90 81 30APR90 84
```

Figure 20.4: Raw Data File Tempdata

You could write a DATA step that reads each record and creates three different Date and Temp variables.

SAS Date Set					
Date1	**Temp1**	**Date2**	**Temp2**	**Date3**	**Temp3**
11048	68	11049	67	11050	70

Figure 20.5: Three Date and Temp Variables

Alternatively, you could create a separate observation for each block of data in a record. This data set is better structured for analysis and reporting with SAS procedures.

SAS Date Set	
Date	**HighTemp**
11048	68
11049	67
11050	70

Figure 20.6: Separate Observations for Each Block of Data in a Record

Holding the Current Record with a Line-Hold Specifier

You need to hold the current record so the input statement can read, and SAS can output, repeated blocks of data on the same record. This is easily accomplished by using a linehold specifier in the INPUT statement.

SAS provides two line-hold specifiers.

- The trailing at sign (@) holds the input record for the execution of the next INPUT statement.
- The double trailing at sign (@@) holds the input record for the execution of the next INPUT statement, even across iterations of the DATA step.

The term *trailing* indicates that the @ or @@ must be the *last* item specified in the INPUT statement. For example, `input Name $20. @; or input Name $20. @@;`

This chapter teaches you how the trailing @@ can be used to hold a record across multiple iterations of the DATA step.

Using the Double Trailing At Sign (@@) to Hold the Current Record

Normally, each time a DATA step executes, the INPUT statement reads the next record. But when you use the trailing @@, the INPUT statement continues reading from the same record.

Table 20.1: Example with the Double Trailing At Sign

```
Input Score;
Input Score @@;
```

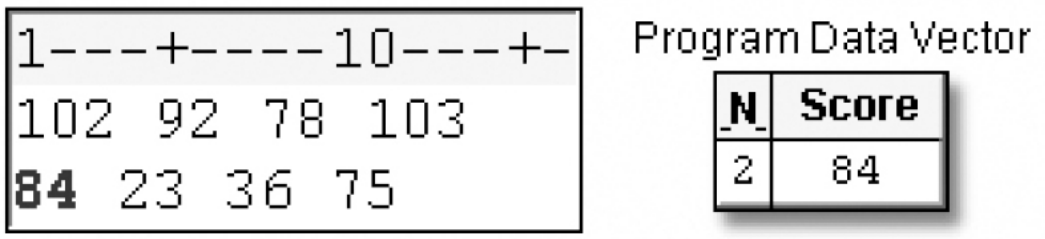


Figure 20.7: Raw Data File and Program Data Vector

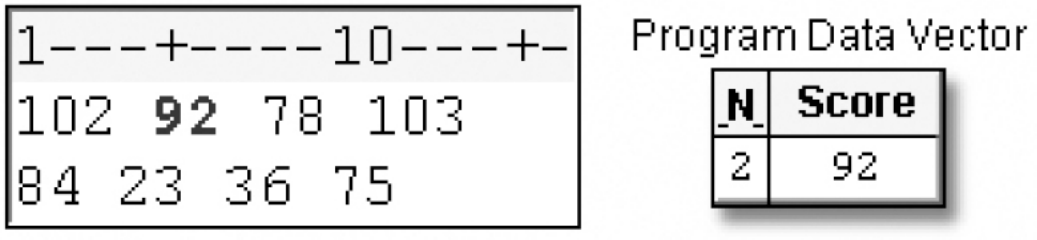


Figure 20.8: Raw Data File and Program Data Vector

The double trailing at sign (@@)

- works like the trailing @ except it holds the data line in the input buffer across multiple executions of the DATA step
- typically is used to read multiple SAS observations from a single data line
- should not be used with the @ pointer control, with column input, nor with the MISSOVER option.

A record that is held by the double trailing at sign (@@) is not released until either of the following events occurs:

- the input pointer moves past the end of the record. Then the input pointer moves down to the next record.

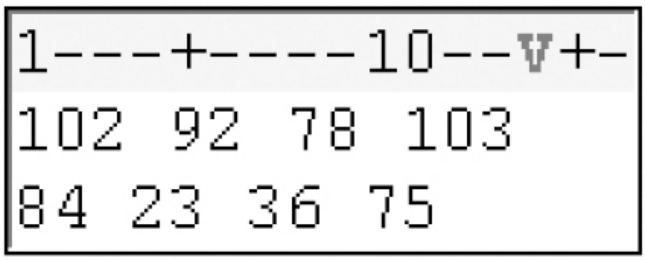


Figure 20.9: Raw Data File

- an INPUT statement that has no trailing at sign executes.

```
input ID $ @@;
.
.
input Department 5.;
```

The following example requires only one INPUT statement to read the values for Date and HighTemp, but the INPUT statement must execute three times for each record.

The INPUT statement reads a block of values for Date and HighTemp, and holds the current record by using the trailing @@. The values in the program data vector are written to the data set as an observation, and control returns to the top of the DATA step.

```
data perm.april90;
   infile tempdata;
   input Date : date. HighTemp @@;
```

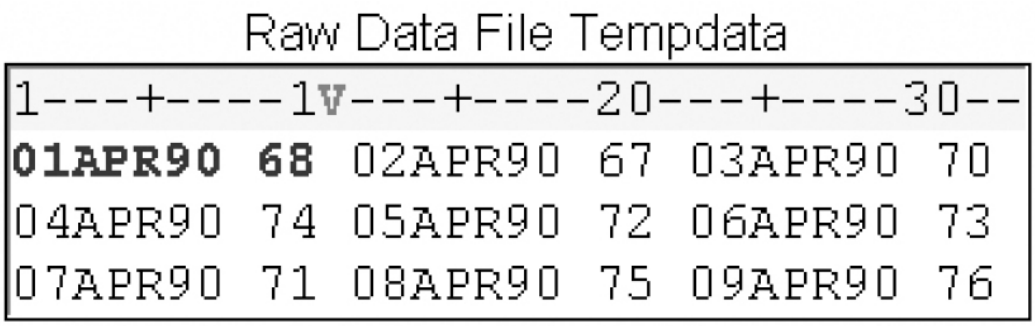


Figure 20.10: Control Returned to the Top of the DATA Step

In the next iteration, the INPUT statement reads the next block of values for Date and HighTemp from the same record.

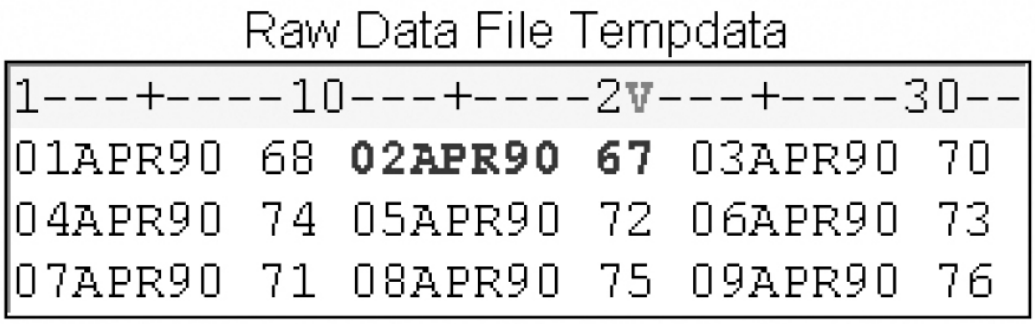


Figure 20.11: Date and High Temp

Completing the DATA Step

You can add a FORMAT statement to the DATA step to display the date or time values with a specified format. The FORMAT statement below uses the DATE*w*. format to display the values for Date in the form *ddmmmyyyy*.

```
data perm.april90;
   infile tempdata;
   input Date : date. HighTemp @@;
   format date date9.;
run;
```

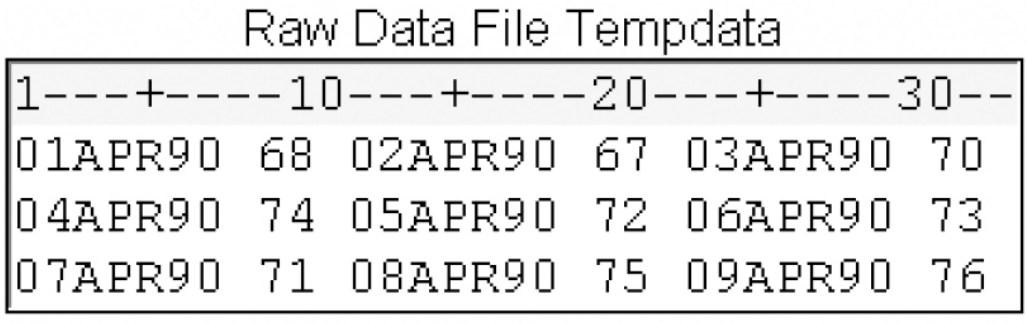


Figure 20.12: Displaying Dates in a Specified Format

DATA Step Processing of Repeating Blocks of Data

Complete DATA Step

```
data perm.april90;
   infile tempdata;
   input Date : date. HighTemp @@;
   format date date9.;
run;
```

Example

As the execution phase begins, the input pointer rests on column 1 of record 1.

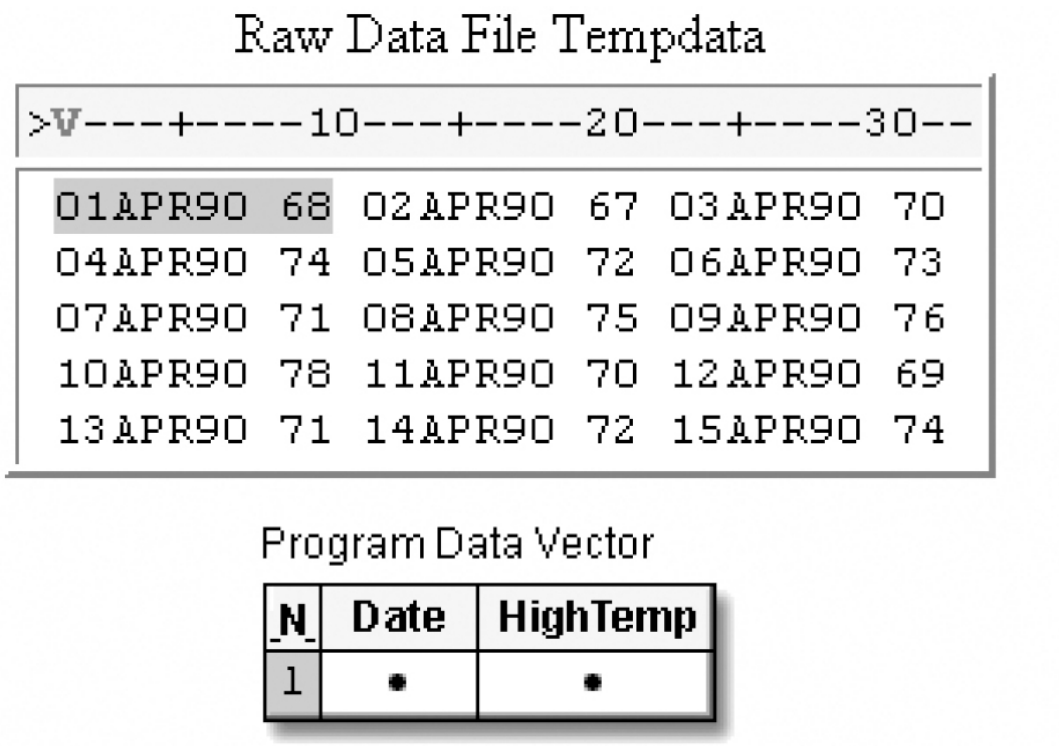


Figure 20.13: Input Pointer on Column 1 of Record 1

During the first iteration of the DATA step, the first block of values for Date and High Temp are read into the program data vector.

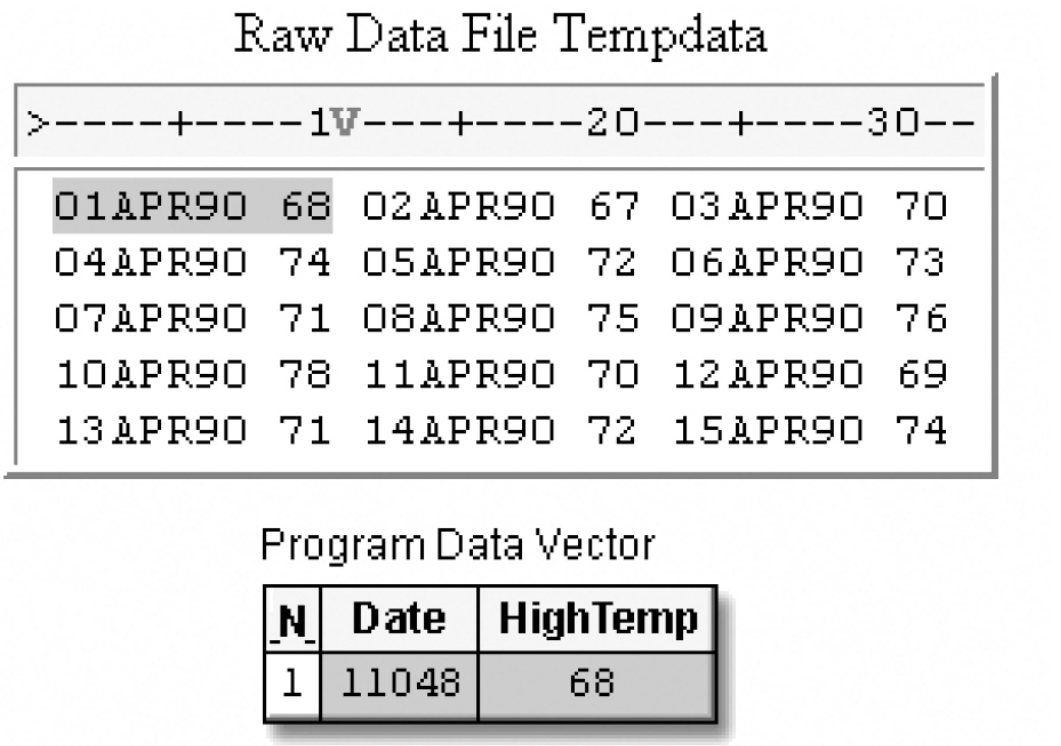


Figure 20.14: Reading the First Block of Values

The first observation is written to the data set.

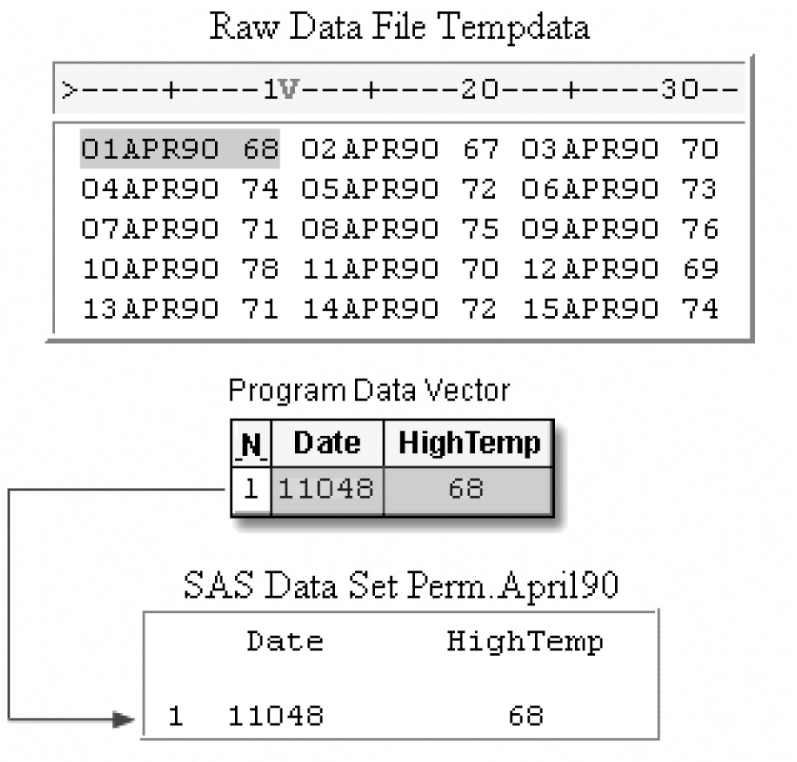


Figure 20.15: Control Returns to the Top of the DATA Step

Control returns to the top of the DATA step, and the values are reset to missing.

Program Date Vector		
_N_	Date	High Temp
2	.	.

Figure 20.16: Reset Values

During the second iteration, the @@ prevents the input pointer from moving down to the next record. Instead, the INPUT statement reads the second block of values for Date and HighTemp from the first record.

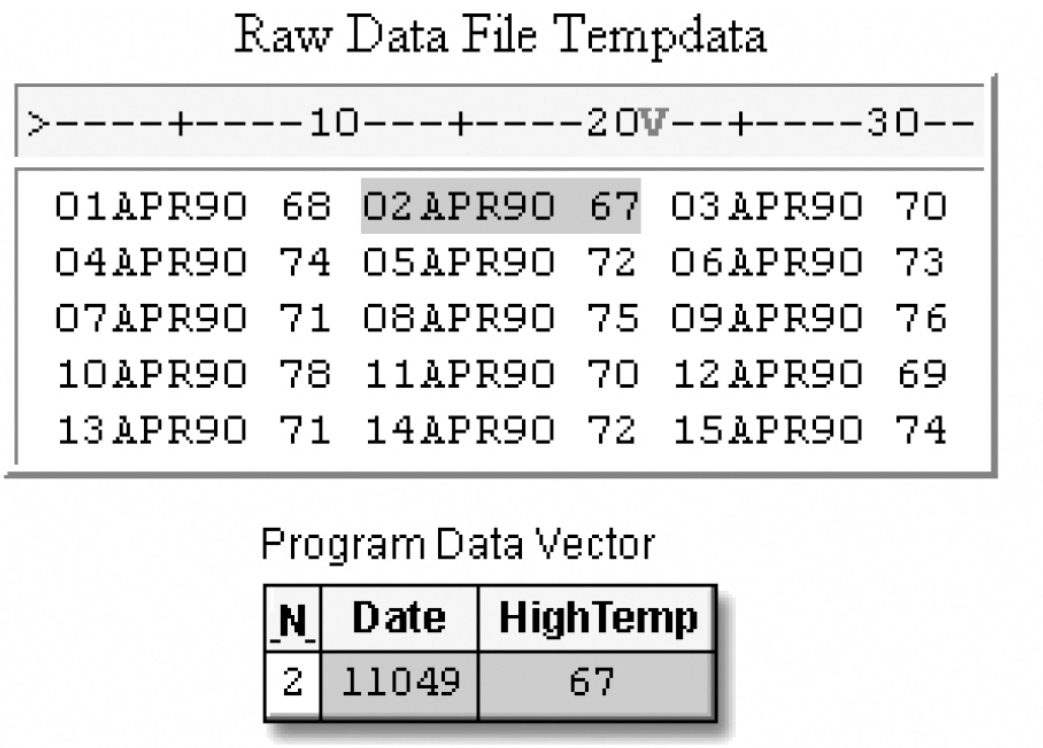


Figure 20.17: Reading the Second Block of Values from the First Record

The second observation is written to the data set, and control returns to the top of the DATA step.

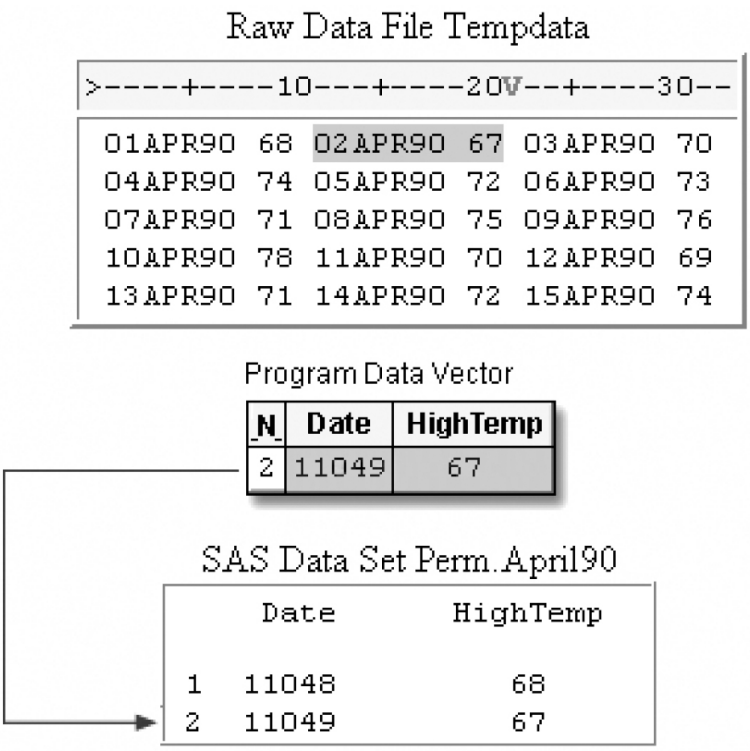


Figure 20.18: Writing the Second Observation to the Data Set

During the third iteration, the last block of values is read and written to the data set as the third observation.

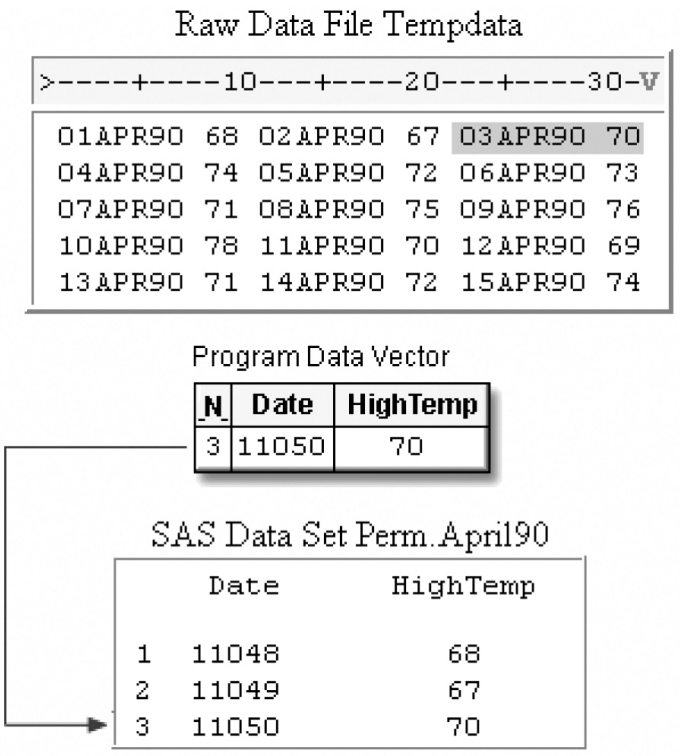


Figure 20.19: Writing the Third Observation to the Data Set

During the fourth iteration, the first block of values in the second record is read and written as the fourth observation.

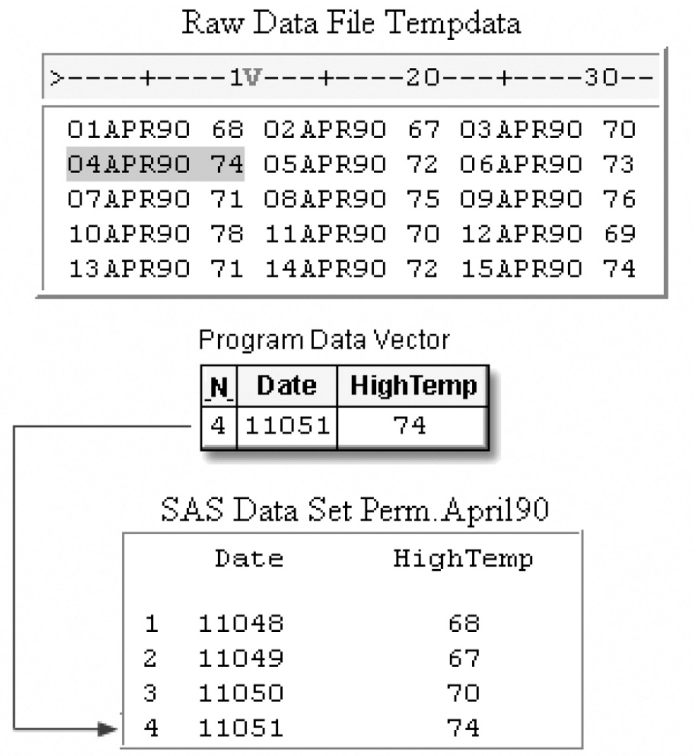


Figure 20.20: Writing the Fourth Observation to the Data Set

The execution phase continues until the last block of data is read.

Raw Data File Tempdata

```
>----+----10---+----20---+----30-V
 01APR90 68 02APR90 67 03APR90 70
 04APR90 74 05APR90 72 06APR90 73
 07APR90 71 08APR90 75 09APR90 76
 10APR90 78 11APR90 70 12APR90 69
 13APR90 71 14APR90 72 15APR90 74
```

SAS Data Set Perm.April90

	Date	HighTemp
1	11048	68
	.	
	.	
	.	
13	11060	71
14	11061	72
15	11062	74

Figure 20.21: Writing the Last Observation to the Data Set

You can display the data set with the PRINT procedure.

```
proc print data=perm.april90;
run;
```

Obs	Date	HighTemp
1	01APR1990	68
2	02APR1990	67
3	03APR1990	70
4	04APR1990	74
5	05APR1990	72
6	06APR1990	73
7	07APR1990	71
8	08APR1990	75
9	09APR1990	76
10	10APR1990	78
11	11APR1990	70
12	12APR1990	69
13	13APR1990	71
14	14APR1990	71
15	15APR1990	74

Figure 20.22: PROC PRINT Output of the Data Set.

Reading the Same Number of Repeating Fields

Overview

So far, you have created multiple observations from a single record by executing the DATA step once for each block of data in a record.

Now, look at another file that is organized differently.

Each record in the file Data97 contains a sales representative's ID number, followed by four repeating fields that represent his or her quarterly sales totals for 1997.

You want to pair each employee ID number with one quarterly sales total to produce a single observation. Four observations are generated from each record.

Raw Data File Data97

```
1---+----10---+----20---+----30---+----40
0734 1,323.34 2,472.85 3,276.65 5,345.52
0943 1,908.34 2,560.38 3,472.09 5,290.86
1009 2,934.12 3,308.41 4,176.18 7,581.81
1043 1,295.38 5,980.28 8,876.84 6,345.94
1190 2,189.84 5,023.57 2,794.67 4,243.35
1382 3,456.34 2,065.83 3,139.08 6,503.49
1734 2,345.83 3,423.32 1,034.43 1,942.28
```

ID	Quarter	Sales
0734	1	1323.34
0734	2	2472.85
0734	3	3276.65
0734	4	5345.52
0943	1	1908.34
0943	2	2560.38
0943	3	3472.09
0943	4	5290.86

Figure 20.23: Multiple Fields for the Same ID

To accomplish this, you must execute the DATA step once for each record, repetitively reading and writing values in one iteration.

This means that a DATA step must

- read the value for ID and hold the current record
- create a new variable named Quarter to identify the fiscal quarter for each sales figure
- read a new value for Sales and write the values to the data set as an observation
- continue reading a new value for Sales and writing values to the data set three more times.

Using the Single Trailing At Sign (@) to Hold the Current Record

First, you need to read the value for ID and hold the record so that subsequent values for Sales can be read.

```
data perm.sales97;
   infile data97;
   input ID $
```

Raw Data File Data97

```
1---V----10---+----20---+----30---+----40
0734 1,323.34 2,472.85 3,276.65 5,345.52
0943 1,908.34 2,560.38 3,472.09 5,290.86
1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Figure 20.24: Holding a Record

You are already familiar with the trailing @@, which holds the current record across multiple iterations of the DATA step.

However, in this case, you want to hold the record with the trailing @, so that a second INPUT statement can read the multiple sales values from a single record within the same iteration of the DATA step. Like the trailing @@, the single trailing @

- enables the next INPUT statement to continue reading from the same record
- releases the current record when a subsequent INPUT statement executes without a line-hold specifier.

It's easy to distinguish between the trailing @@ and the trailing @ by remembering that

- the double trailing at sign (@@) holds a record across multiple iterations of the DATA step until the end of the record is reached.
- the single trailing at sign (@) releases a record when control returns to the top of the DATA step.

In this example, the first INPUT statement reads the value for ID and uses the trailing @ to hold the current record for the next INPUT statement in the DATA step.

```
data perm.sales97;
   infile data97;
   input ID $ @;
   input Sales : comma. @;
output;
```

Raw Data File Data97

```
1---V----10---+----20---+----30---+----40
0734 1,323.34 2,472.85 3,276.65 5,345.52
0943 1,908.34 2,560.38 3,472.09 5,290.86
1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Figure 20.25: Reading the Value for ID

The second INPUT statement reads a value for Sales and holds the record. The COMMAw.d informat in the INPUT statement reads the numeric value for Sales and removes the embedded commas. An OUTPUT statement writes the observation to the SAS data set, and the DATA step continues processing.

Additional Note Notice that the COMMA*w.d* informat does *not* specify a *w* value. Remember that list input reads values until the next blank is detected. The default length of numeric variables is 8 bytes, so you don't need to specify a *w* value to determine the length of a numeric variable.

When all of the repeating fields have been read and output, control returns to the top of the DATA step, and the record is released.

```
data perm.sales97;
   infile data97;
   input ID $ @;
```

```
   input Sales : comma. @;
   output;
   input Sales : comma. @;
   output;
   input Sales : comma. @;
   output;
   input Sales : comma. @;
   output;
run;
```

Raw Data File Data97

```
1---+----10--V+----20---+----30---+----40
0734 1,323.34 2,472.85 3,276.65 5,345.52
0943 1,908.34 2,560.38 3,472.09 5,290.86
1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Figure 20.26: Reading the Value for Sales

More Efficient Programming

Each record contains four different values for the variable Sales, so the INPUT statement must execute four times. Rather than writing four INPUT statements, you can execute one INPUT statement repeatedly in an iterative DO loop.

Each time the loop executes, you need to write the values for ID, Quarter, and Sales as an observation to the data set. This is easily accomplished by using the OUTPUT statement.

```
data perm.sales97;
   infile data97;
   input ID $ @;
   do Quarter=1 to 4;
      input Sales : comma. @;
      output;
   end;
run;
```

By default, every DATA step contains an implicit OUTPUT statement at the end of the step. Placing an explicit OUTPUT statement in a DATA step overrides the automatic output, and SAS adds an observation to a data set only when the explicit OUTPUT statement is executed.

Processing a DATA Step That Contains an Iterative DO Loop

Now that the program is complete, let's see how SAS processes a DATA step that contains an iterative DO loop.

```
data perm.sales97;
   infile data97;
   input ID $ @;
   do Quarter=1 to 4;
      input Sales : comma. @;
      output;
   end;
run;
```

During the first iteration, the value for ID is read and Quarter is initialized to **1** as the loop begins to execute.

Raw Data File Data97

```
>----V----10---+----20---+----30---+----40-
 0734 1,323.34 2,472.85 3,276.65 5,345.52
 0943 1,908.34 2,560.38 3,472.09 5,290.86
 1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Program Data Vector

_N_	ID	Quarter	Sales
1	0734	1	.

Figure 20.27: Reading the Value for ID and Initializing Quarter

The INPUT statement reads the first repeating field and assigns the value to Sales in the program data vector. The @ holds the current record.

Raw Data File Data97

```
>----+----10--V+----20---+----30---+----40-
 0734 1,323.34 2,472.85 3,276.65 5,345.52
 0943 1,908.34 2,560.38 3,472.09 5,290.86
 1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Program Data Vector

_N_	ID	Quarter	Sales
1	0734	1	1323.34

Figure 20.28: Results of the INPUT Statement

The OUTPUT statement writes the values in the program data vector to the data set as the first observation.

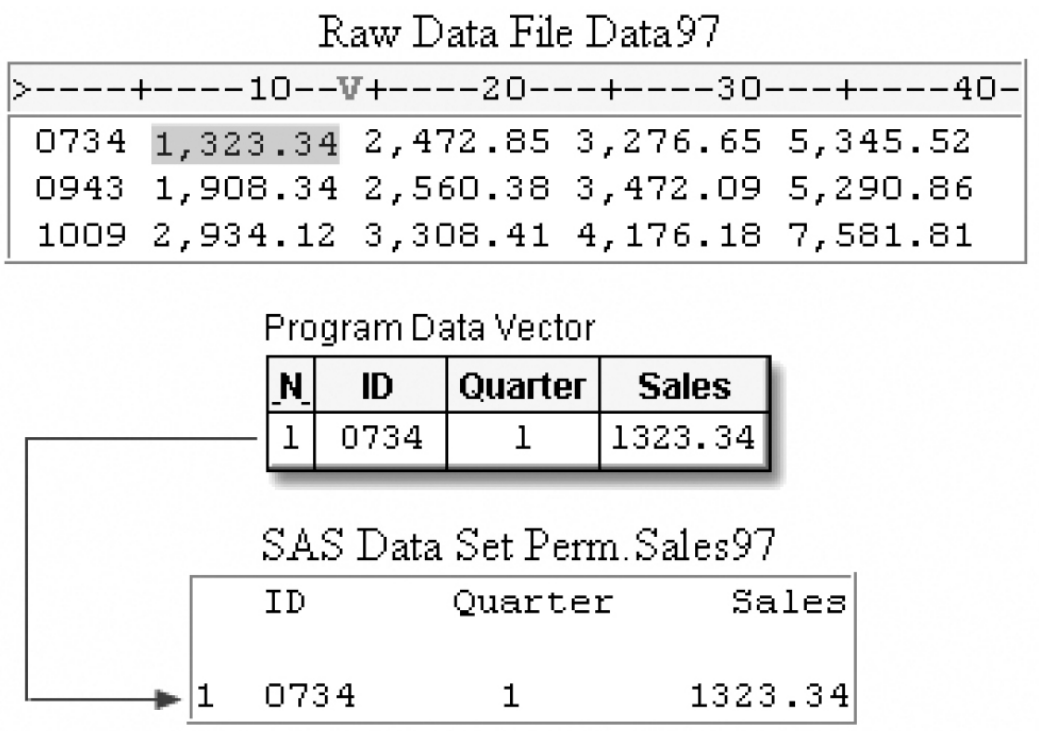


Figure 20.29: Results of the OUTPUT Statement

The END statement indicates the bottom of the loop, but control returns to the DO statement, not to the top of the DATA step. Now the value of Quarter is incremented to **2**.

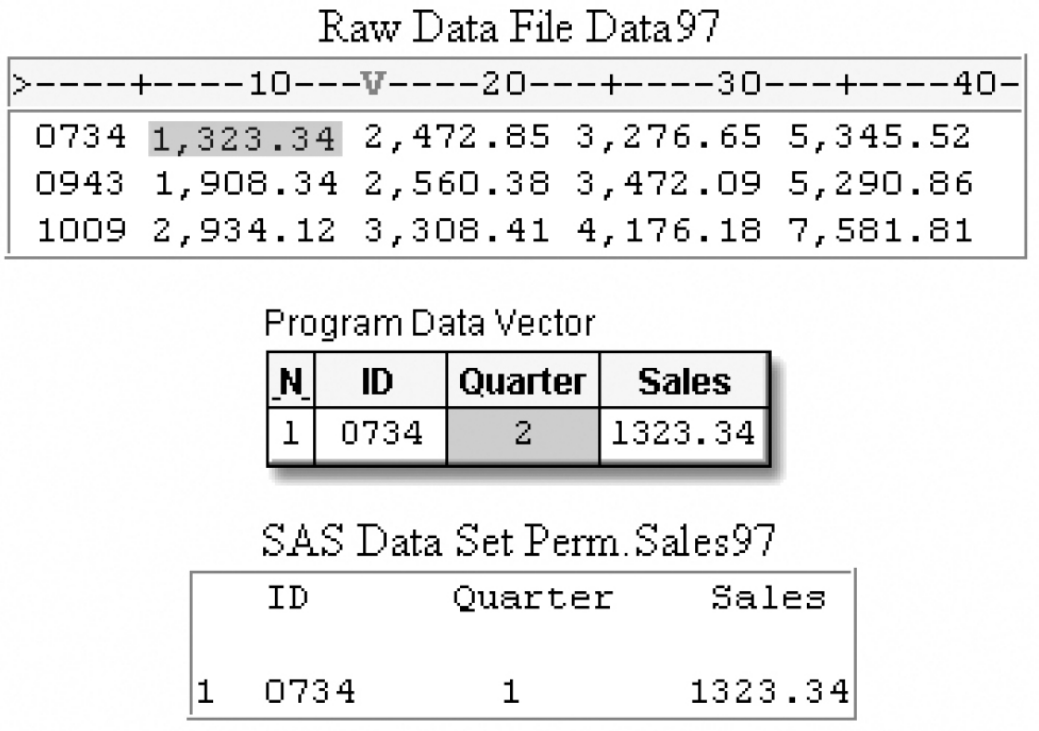


Figure 20.1: Results of the END Statement

The INPUT statement executes again, reading the second repeating field and storing the value for Sales in the program data vector.

Raw Data File Data97

```
>----+----10---+----20-V-+----30---+----40-
 0734 1,323.34 2,472.85 3,276.65 5,345.52
 0943 1,908.34 2,560.38 3,472.09 5,290.86
 1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Program Data Vector

_N_	ID	Quarter	Sales
1	0734	2	2472.85

SAS Data Set Perm.Sales97

	ID	Quarter	Sales
1	0734	1	1323.34

Figure 20.30: Results of the Second Reading of the INPUT Statement

The OUTPUT statement writes the values in the program data vector as the second observation.

Raw Data File Data97

```
>----+----10---+----20-V-+----30---+----40-
 0734 1,323.34 2,472.85 3,276.65 5,345.52
 0943 1,908.34 2,560.38 3,472.09 5,290.86
 1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Program Data Vector

_N_	ID	Quarter	Sales
1	0734	2	2472.85

SAS Data Set Perm.Sales97

	ID	Quarter	Sales
1	0734	1	1323.34
2	0734	2	2472.85

Figure 20.31: Results of the Second Reading of the OUTPUT Statement

The loop continues executing while the value for Quarter is **3**, and then **4**. In the process, the third and fourth observations are written.

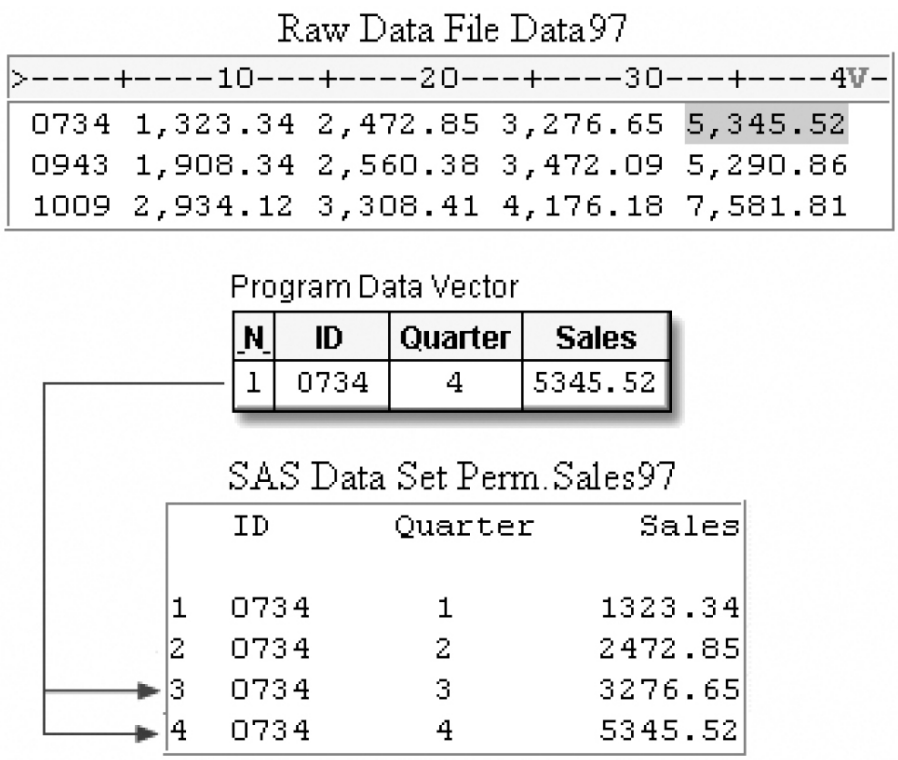
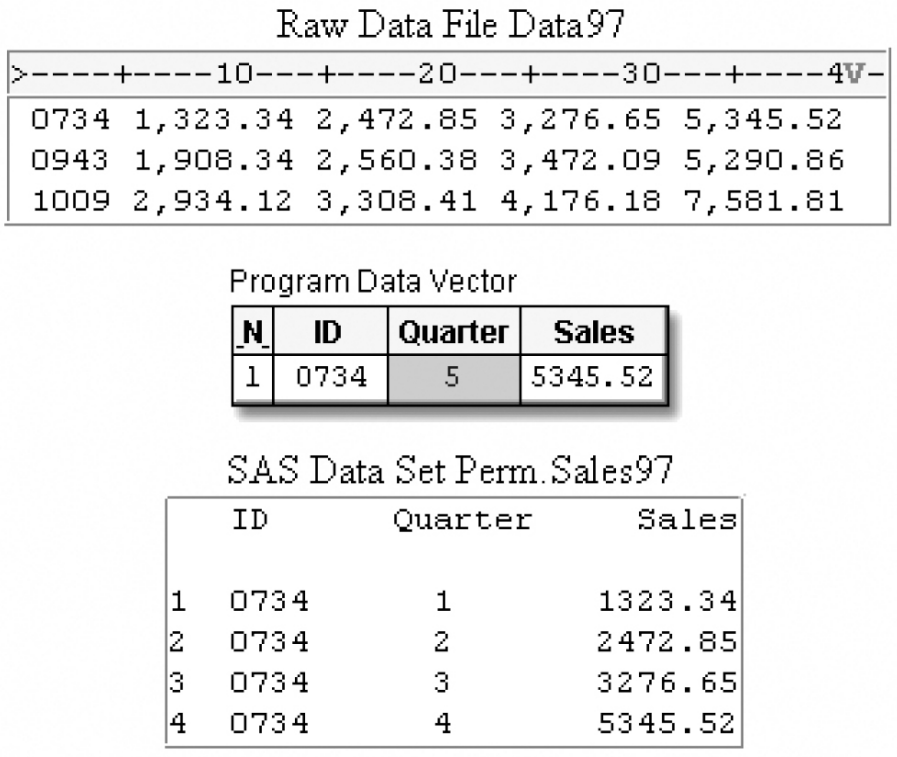


Figure 20.32: Writing the Third and Fourth Observations

After the fourth observation is written, Quarter is incremented to **5** at the bottom of the DO loop and control returns to the top of the loop. The loop does not execute again because the value of Quarter is now greater than **4**.



Control returns to the top of the DATA step, and the input pointer moves to column 1 of the next record. The variable values in the program data vector are reset to missing.

Raw Data File Data97

```
>V---+----10---+----20---+----30---+----40-
 0734 1,323.34 2,472.85 3,276.65 5,345.52
 0943 1,908.34 2,560.38 3,472.09 5,290.86
 1009 2,934.12 3,308.41 4,176.18 7,581.81
```

Program Date Vector

_N_	ID	Quarter	Sales
2	.	.	.

SAS Date Set Perm.Sales97

	ID	Quarter	Sales
1	0734	1	1323.34
2	0734	2	2472.52
3	0734	3	3276.65
4	0734	4	5345.52

Figure 20.33: Returning Control to the Top of the DATA Step

When the execution phase is complete, you can display the data set with the PRINT procedure.

```
proc print data=perm.sales97;
run;
```

Obs	ID	Quarter	Sales
1	0734	1	1323.34
2	0734	2	2472.85
3	0734	3	3276.65
4	0734	4	5345.52
5	0943	1	1908.34
6	0943	2	2560.38
7	0943	3	3472.09
8	0943	4	5290.86
9	1009	1	2934.12
10	1009	2	3308.41
11	1009	3	4176.18
12	1009	4	7581.81

Figure 20.34: Output of PROC PRINT

Reading a Varying Number of Repeating Fields

Overview

So far, each record in the file Data97 has contained the same number of repeating fields.

Raw Data File Data97

```
1---+----10---+----20---+----30---+----40
0734 1,323.34 2,472.85 3,276.65 5,345.52
0943 1,908.34 2,560.38 3,472.09 5,290.86
1009 2,934.12 3,308.41 4,176.18 7,581.81
1043 1,295.38 5,980.28 8,876.84 6,345.94
1190 2,189.84 5,023.57 2,794.67 4,243.35
1382 3,456.34 2,065.83 3,139.08 6,503.49
1734 2,345.83 3,423.32 1,034.43 1,942.28
```

Figure 20.35: Raw Data File Data97

But suppose that some of the employees quit at various times. Their records might not contain sales totals for the second, third, or fourth quarter. These records contain a variable number of repeating fields.

Raw Data File Data97

```
1---+----10---+----20---+----30---+----40
1824 1,323.34 2,472.85
1943 1,908.34
2046 1,423.52 1,673.46 3,276.65
2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Figure 20.36: Raw Data File Data97 Showing Empty Records

The DATA step that you just wrote won't work with a variable number of repeating fields because now the value of Quarter is not constant for every record.

```
data perm.sales97;
   infile data97;
   input ID $ @;
   do Quarter=1 to 4;
      input Sales : comma. @;
      output;
   end;
run;
```

Using the MISSOVER Option

You can adapt the DATA step to accommodate a varying number of Sales values.

Like the previous example with the same number of repeating fields, your DATA step must read the same record repeatedly. However, you need to prevent the input pointer from moving to the next record when there are missing Sales values.

You can use the MISSOVER option in an INFILE statement to prevent SAS from reading the next record when missing values are encountered at the end of a record. Essentially, records that have a varying number of repeating fields are records that contain missing values, so you need to specify the MISSOVER option here as well.

Because there is at least one value for the repeating field, Sales, in each record, the first INPUT statement reads *both* the value for ID and the first Sales value for each record. The trailing @ holds the record so that any subsequent repeating fields can be read.

```
data perm.sales97;
   infile data97 missover;
```

```
input ID $ Sales : comma. @;
```

Raw Data File Data97

```
1---+----10--V+----20---+----30---+----40
1824 1,323.34 2,472.85
1943 1,908.34
2046 1,423.52 1,673.46 3,276.65
2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Figure 20.37: Holding a Record

Additional Note SAS provides several options to control reading past the end of a line. You've seen the MISSOVER option for setting remaining INPUT statement variables to missing values if the pointer reaches the end of a record. You can also use other options such as the TRUNCOVER option, which reads column or formatted input when the last variable that is read by the INPUT statement contains varying-length data. The TRUNCOVER option assigns the contents of the input buffer to a variable when the field is shorter than expected.

Other related options include FLOWOVER (the default), STOPOVER, and SCANOVER. For more information about TRUNCOVER and related options, see the SAS documentation.

Executing SAS Statements While a Condition Is True

Now consider how many times to read each record. Earlier, you created an index variable named Quarter whose value ranged from **1** to **4** because there were four repeating fields.

Now you want to read the record only while a value for Sales exists. Use a DO WHILE statement instead of the iterative DO statement, enclosing the expression in parentheses. In the example below, the DO WHILE statement executes while the value of Sales is not equal to a missing value (which is represented by a period).

```
data perm.sales97;
   infile data97 missover;
   input ID $ Sales : comma. @;
   do while (sales ne .);
```

Creating a Counter Variable

Because the DO WHILE statement does not create an index variable, you can create your own "counter" variable. You can use a sum statement to increment the value of the counter variable each time the DO WHILE loop executes.

In the example below, the assignment statement that precedes the loop creates the counter variable Quarter and assigns it an initial value of zero. Each time the DO WHILE loop executes, the sum statement increments the value of Quarter by one.

```
data perm.sales97;
   infile data97 missover;
   input ID $ Sales : comma. @;
   Quarter=0;
   do while (sales ne .);
      quarter+1;
```

Completing the DO WHILE Loop

Now look at the other statements that should be executed in the DO WHILE loop. First, you need an OUTPUT statement to write the current observation to the data set. Then, another INPUT statement reads the next value for Sales and holds the record. You complete the DO WHILE loop with an END statement.

```
data perm.sales97;
   infile data97 missover;
   input ID $ Sales : comma. @;
   Quarter=0;
   do while (sales ne .);
      quarter+1;
```

```
      output;
      input sales : comma. @;
   end;
run;
```

Processing a DATA Step That Has a Varying Number of Repeating Fields

This example uses the following DATA step:

```
data perm.sales97;
   infile data97 missover;
   input ID $ Sales : comma. @;
   Quarter=0;
   do while (sales ne .);
      quarter+1;
      output;
      input sales : comma. @;
   end;
run;
```

During the first iteration of the DATA step, values for ID and Sales are read. Quarter is initialized to *0*.

Raw Data File Data97

```
>----+----10--V+----20---+----30---+----40-
1824 1,323.34 2,472.85
1943 1,908.34
2046 1,423.52 1,673.46 3,276.65
2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Program Data Vector

_N_	ID	Sales	Quarter
1	1824	1323.34	0

Figure 20.38: Initializing the Value of Quarter to 0

The DO WHILE statement checks to see if Sales has a value, which it does, so the other statements in the DO loop execute. The Value of Quarter is incremented by 1 and the current observation is written to the data set.

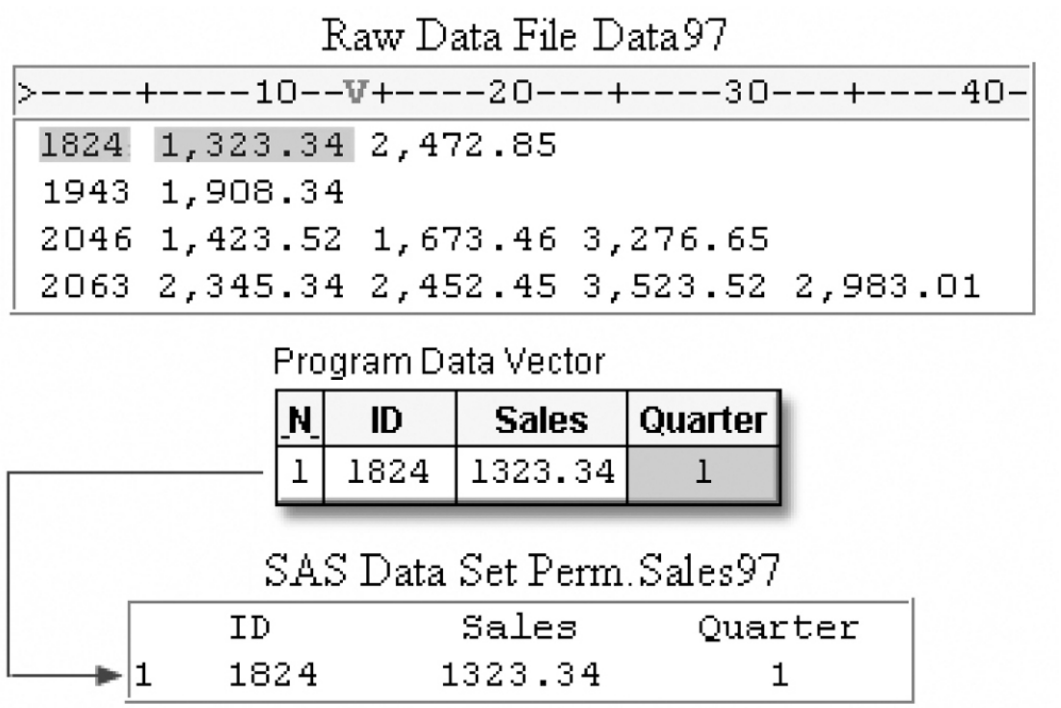


Figure 20.39: Executing the DO Loop

The INPUT statement reads the next value for Sales, the end of the loop is reached, and control returns to the DO WHILE statement.

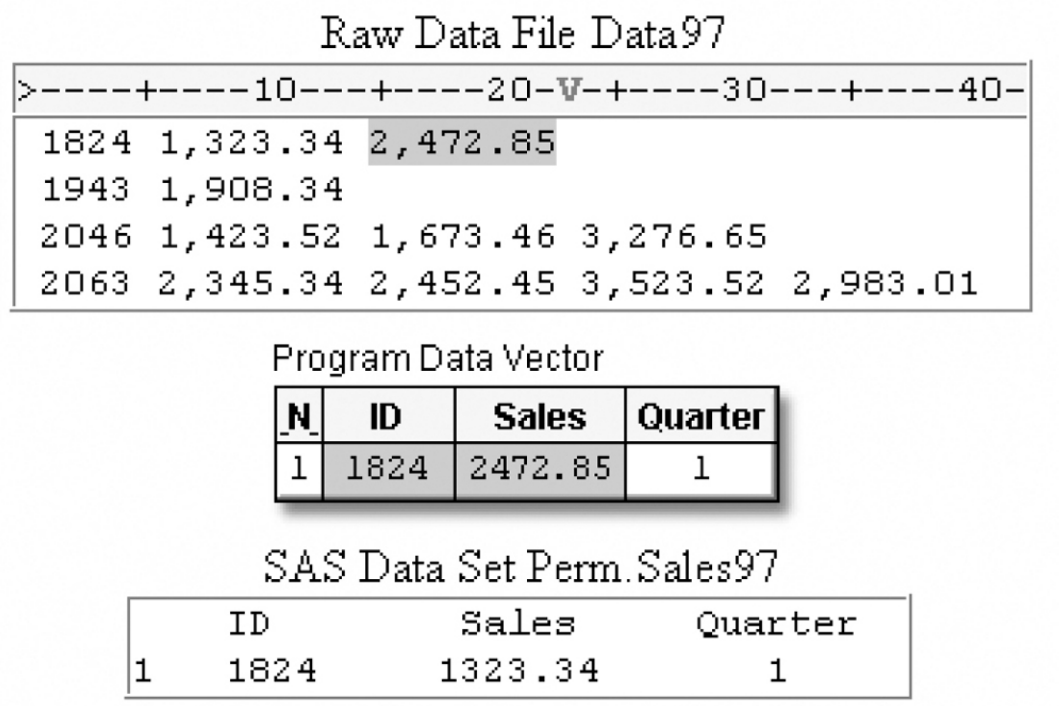


Figure 20.40: Returning Control to the DO WHILE Statement

The condition is checked and Sales still has a value, so the loop executes again.

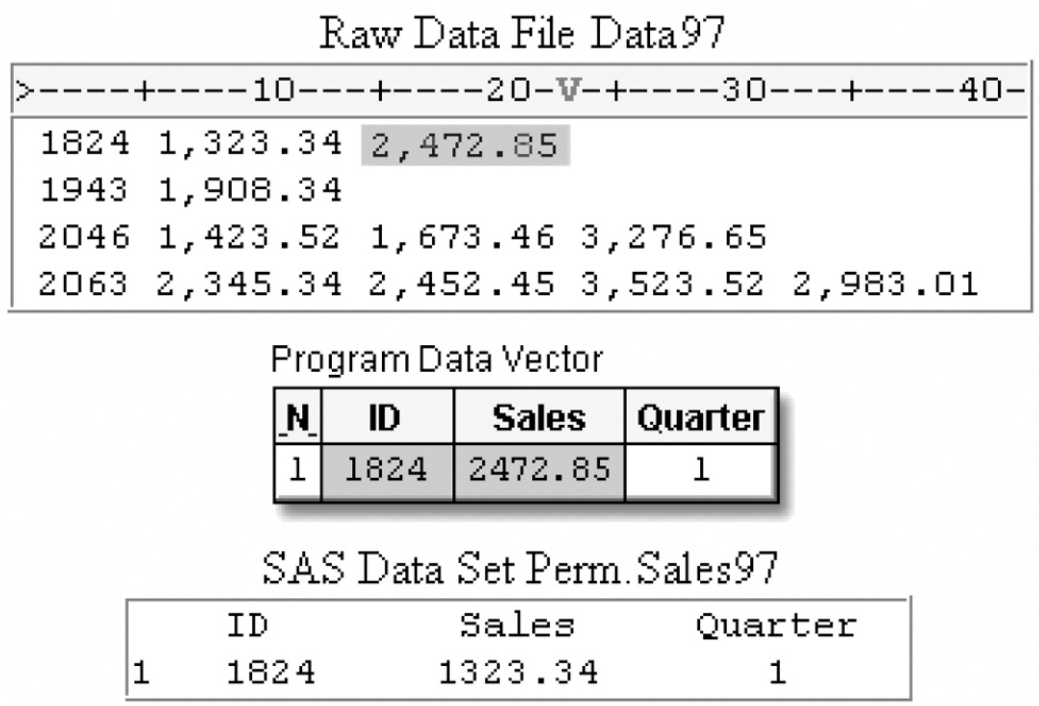
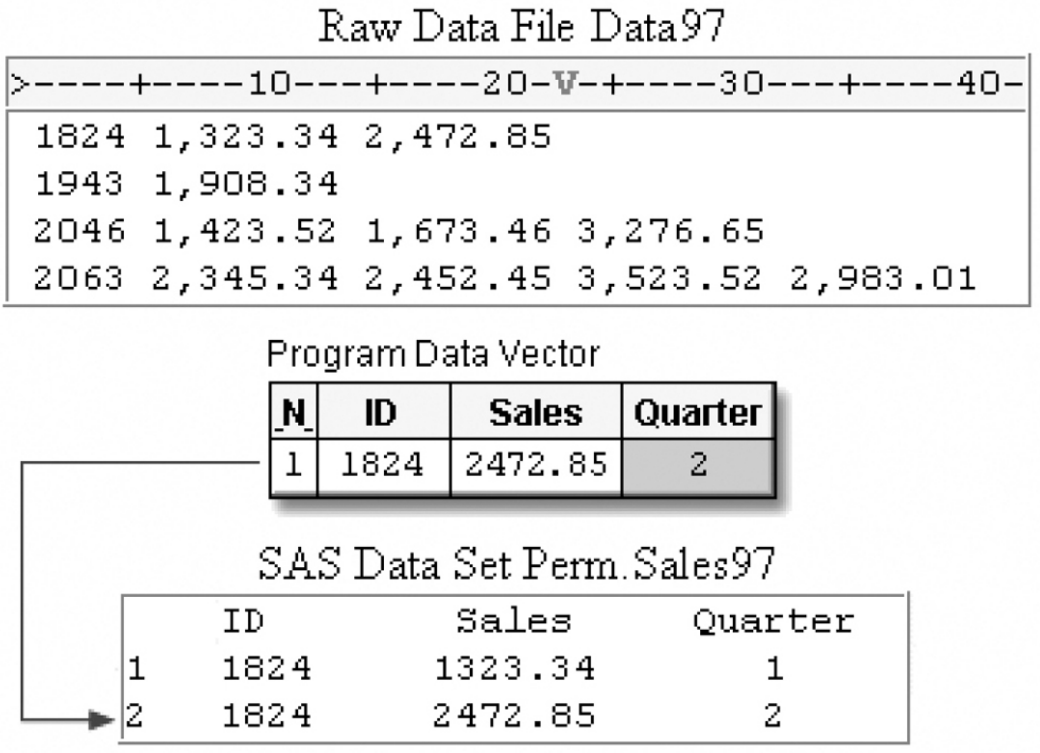


Figure 20.41: Executing the Loop Again

Quarter is incremented to 2, and the values in the program data vector are written as the second observation.



The INPUT statement executes again. The MISSOVER option prevents the input pointer from moving to the next record in search of another value for Sales. Therefore, SALEs receives a missing value.

Raw Data File Data97

```
>----+----10---+----20-V-+----30---+----40-
 1824 1,323.34 2,472.85
 1943 1,908.34
 2046 1,423.52 1,673.46 3,276.65
 2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Program Data Vector

_N_	ID	Sales	Quarter
1	1824	.	2

SAS Data Set Perm.Sales97

	ID	Sales	Quarter
1	1824	1323.34	1
2	1824	2472.85	2

The end of the loop is reached, and control returns to the DO WHILE statement. Because the condition is now false, the statements in the loop are not executed and the values in the PDV are not output.

Raw Data File Data97

```
>----+----10---+----20-V-+----30---+----40-
 1824 1,323.34 2,472.85
 1943 1,908.34
 2046 1,423.52 1,673.46 3,276.65
 2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Program Data Vector

_N_	ID	Sales	Quarter
1	1824	.	2

SAS Data Set Perm.Sales97

	ID	Sales	Quarter
1	1824	1323.34	1
2	1824	2472.85	2

Figure 20.42: Returning Control the DO WHILE Statement

Instead, control returns to the top of the DATA step, the values in the program data vector are reset to missing, and the input statement reads the next record. The DATA step continues executing until the end of the file.

Raw Data File Data97

```
>V---+----10---+----20---+----30---+----40-
 1824 1,323.34 2,472.85
 1943 1,908.34
 2046 1,423.52 1,673.46 3,276.65
 2063 2,345.34 2,452.45 3,523.52 2,983.01
```

Program Data Vector

_N_	ID	Sales	Quarter
2	•	•	•

SAS Data Set Perm.Sales97

	ID	Sales	Quarter
1	1824	1323.34	1
2	1824	2472.85	2

Figure 20.43: Returning Control to the Top of the DATA Step

PROC PRINT output for the data set shows a varying number of observations for each employee.

```
proc print data=perm.sales97;
run;
```

Obs	ID	Sales	Quarter
1	1824	1323.34	1
2	1824	2472.85	2
3	1943	2199.23	1
4	2046	3598.48	1
5	2046	4697.98	2
6	2046	4598.45	3
7	2063	4963.87	1
8	2063	3434.42	2
9	2063	2241.64	3
10	2063	2759.11	4

Figure 20.44: Output of PROC PRINT

Chapter Summary

Text Summary

File Formats

One raw data record might contain data for several observations. Data might be stored in this manner in order to reduce the size of the entire file. The data can be organized into

- repeating blocks of data
- an ID field followed by the same number of repeating fields

- an ID field followed by a varying number of repeating fields.

Reading Repeating Blocks of Data

To create multiple observations from a record that contains repeating blocks of data, the INPUT statement needs to hold the current record until each block of data has been read and written to the data set as an observation. The DATA step should include statements that

1. read the first block of values and hold the current record with the double trailing at sign (@@) line-hold specifier
2. optionally add a FORMAT statement to display date or time values with a specified format
3. write the first block of values as an observation
4. execute the DATA step until all repeating blocks have been read.

Reading the Same Number of Repeating Fields

To create multiple observations from a record that contains an ID field and the same number of repeating fields, you must execute the DATA step once for each record, repetitively reading and writing values in one iteration. The DATA step should include statements that

1. read the ID field and hold the current record with the single trailing at sign (@) linehold specifier
2. execute SAS statements using an iterative DO loop. The iterative DO loop repetitively processes statements that
   - read the next value of the repeating field and hold the record with the @ linehold specifier
   - explicitly write an observation to the data set by using an OUTPUT statement.
3. complete the iterative DO loop with an END statement.

Reading a Varying Number of Repeating Fields

To create multiple observations from a record that contains an ID field and a varying number of repeating fields, you must execute the DATA step once for each record, repetitively reading and writing values in one iteration while the value of the repeating field is nonmissing. The DATA step should include statements that

1. prevent SAS from reading the next record if missing values were encountered in the current record. The MISSOVER option on the INFILE statement prevents reading the next record.
2. read the ID field and the first repeating field, and then hold the record with the single trailing at sign (@) line-hold specifier
3. optionally create a counter variable
4. execute SAS statements while a condition is true, using a DO WHILE loop. A DO WHILE loop repetitively processes statements that
   - optionally increment the value of the counter variable by using a sum statement
   - explicitly add an observation to the data set by using an OUTPUT statement
   - read the next value of the repeating field and hold the record with the single trailing at sign (@) line-hold specifier.
5. complete the DO WHILE loop with an END statement.

Syntax

Repeating Blocks of Data

```
LIBNAME libref 'SAS-data-library';
FILENAME fileref 'filename';
DATA SAS-data-set;
     INFILE file-specification;
     INPUT variables @@;
     FORMAT date/time-variable format;
```

```
RUN;
```

An ID Field Followed by the Same Number of Repeating Fields

```
LIBNAME libref 'SAS-data-library';
FILENAME fileref 'filename';
DATA SAS-data-set;
    INFILE file-specification;
    INPUT id-variable @;
    DO index-variable specification;
        INPUT repeating-variable @;
        OUTPUT;
    END;
RUN;
```

An ID Field Followed by a Varying Number of Repeating Fields

```
LIBNAME libref 'SAS-data-library';
FILENAME fileref 'filename';
DATA SAS-data-set;
     INFILE file-specification MISSOVER;
     INPUT id-variable repeating-variable @;
     counter-variable=0;
     DO WHILE (expression);
         counter-variable+1;
         OUTPUT;
         INPUT repeating-variable @;
     END;
RUN;
```

Sample Programs

Repeating Blocks of Data

```
libname perm 'c:\records\weather';
filename tempdata 'c:\records\weather\tempdata';
data perm.april90;
   infile tempdata;
   input Date : date. HighTemp @@;
   format date date9.;
run;
```

An ID Field Followed by the Same Varying Number of Repeating Fields

```
libname perm 'c:\records\sales';
filename data97 'c:\records\sales\1997.dat';
data perm.sales97;
   infile data97;
   input ID $ @;
   do Quarter=1 to 4;
      input Sales : comma. @;
      output;
   end;
run;
```

An ID Field Followed by a Varying Number of Repeating Fields

```
libname perm 'c:\records\sales';
filename data97 'c:\records\sales\1997.dat';
data perm.sales97;
   infile data97 missover;
   input ID $ Sales : comma. @;
   Quarter=0;
   do while (sales ne .);
      quarter+1;
      output;
      input sales : comma. @;
   end;
run;
```

Points to Remember

- The double trailing at sign (@@) holds a record across multiple iterations of the DATA step until the end of the record is reached.
- The single trailing at sign (@) releases a record when control returns to the top of the DATA step.
- Use an END statement to complete DO loops and DO WHILE loops.

Chapter Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which is true for the double trailing at sign (@@)? ?

a. It enables the next INPUT statement to read from the current record across multiple iterations of the DATA step.

b. It must be the last item specified in the INPUT statement.

c. It is released when the input pointer moves past the end of the record.

d. all of the above

2. A record that is being held by a single trailing at sign (@) is automatically released when ?

a. the input pointer moves past the end of the record.

b. the next iteration of the DATA step begins.

c. another INPUT statement that has an @ executes.

d. another value is read from the observation.

3. Which SAS program correctly creates a separate observation for each block of data? ?

```
1---+----10---+----20---+----30---+----40---+
1001 apple 1002 banana 1003 cherry
1004 guava 1005 kiwi 1006 papaya
1007 pineapple 1008 raspberry 1009 strawberry
```

Figure 20.45: Raw Data File

a.
```
data perm.produce;
   infile fruit;
   input Item $ Variety : $10.;
run;
```

b.
```
data perm.produce;
   infile fruit;
   input Item $ Variety : $10. @;
run;
```

c.
```
data perm.produce;
   infile fruit;
   input Item $ Variety : $10. @@;
run;
```

d.
```
data perm.produce;
   infile fruit @@;
   input Item $ Variety : $10.;
run;
```

4. Which SAS program reads the values for ID and holds the record for each value of Quantity, so that three observations are created for each record? ?

```
1---+----10---+----20---+----30
2101 21,208 19,047 22,890
2102 18,775 20,214 22,654
2103 19,763 22,927 21,862
```

Figure 20.46: Raw Data File

a.
```
data work.sales;
   infile unitsold;
   input ID $;
   do week=1 to 3;
      input Quantity : comma.;
      output;
   end;
run;
```

b.
```
data work.sales;
   infile unitsold;
   input ID $ @@;
   do week=1 to 3;
      input Quantity : comma.;
      output;
   end;
run;
```

c.
```
data work.sales;
   infile unitsold;
   input ID $ @;
   do week=1 to 3;
      input Quantity : comma.;
      output;
   end;
run;
```

d.
```
data work.sales;
   infile unitsold;
   input ID $ @;
   do week=1 to 3;
      input Quantity : comma. @;
      output;
   end;
run;
```

5. Which SAS statement repetitively executes several statements when the value of an index variable named count ranges from **1** to **50**, incremented by 5? ?

a. `do count=1 to 50 by 5;`

b. `do while count=1 to 5 0 by 5;`

c. `do count=1 to 50 + 5;`

d. `do while (count=1 to 50 + 5);`

6. Which option below, when used in a DATA step, writes an observation to the data set after each value for Activity has been read? ?

a.
```
do choice=1 to 3;
   input Activity : $10. @;
   output;

end;
run;
```

b.
```
do choice=1 to 3;
   input Activity : $10. @;
end;
output;
run;
```

c.
```
do choice=1 to 3;
   input Activity : $10. @;
end;
run;
```

d. both a and b

7. Which SAS statement repetitively executes several statements while the value of Cholesterol is greater than **200?** ?

a. `do cholesterol > 200;`

b. `do cholesterol gt 200;`

c. `do while (cholesterol > 200);`

d. `do while cholesterol > 200;`

8. Which choice below is an example of a sum statement? ?

a. `totalpay=1;`

b. `totalpay+1;`

c. `totalpay*1;`

d. `totalpay by 1;`

9. Which program creates the SAS data set Perm.Topstore from the raw data file shown below? ?

```
1---+----10---+----20---+----30---+
1001 77,163.19 76,804.75 74,384.27
1002 76,612.93 81,456.34 82,063.97
1003 82,185.16 79,742.33
```

Figure 20.47: Raw Data File

SAS Date Set Perm. Topstore

Store	Sales	Month
1001	77163.19	1
1001	76804.75	2
1001	74384.27	3

1002	76612.93	1
1002	81456.34	2
1002	82063.97	3
1003	82185.16	1
1003	79742.33	2

Figure 20.48: Output from PROC PRINT

a.
```
data perm.topstores;
   infile sales98 missover;
   input Store Sales : comma. @;
   do while (sales ne .);
      month + 1;
      output;
      input sales : comma. @;
   end;
run;
```

b.
```
data perm.topstores;
   infile sales98 missover;
   input Store Sales : comma. @;
   do while (sales ne .);
      Month=0;
      month + 1;
      output;
      input sales : comma. @;
   end;
run;
```

c.
```
data perm.topstores;
   infile sales98 missover;
   input Store Sales : comma.
   Month @;
   do while (sales ne .);
       month + 1;
      input sales : comma. @;
   end;
   output;
run;
```

d.
```
data perm.topstores;
   infile sales98 missover;
   input Store Sales : comma. @;
   Month=0;
   do while (sales ne .);
      month + 1;
      output;
      input sales : comma. @;

   end;
run;
```

10. How many observations are produced by the DATA step that reads this external file? ?

```
1---+----10---+----20---+----30---+----40
01 CHOCOLATE VANILLA RASPBERRY
02 VANILLA PEACH
03 CHOCOLATE
04 RASPBERRY PEACH CHOCOLATE
05 STRAWBERRY VANILLA CHOCOLATE
```

Figure 20.49: Raw Data File

```
data perm.choices;
   infile icecream missover;
   input Day $ Flavor : $10. @;
   do while (flavor ne ' ');
       output; input flavor : $10. @;
   end;
run;
```

a. 3

b. 5

c. 12

d. 15

Answers

1. Correct answer: d

   The double trailing at sign (@@) enables the next INPUT statement to read from the current record across multiple iterations of the DATA step. It must be the last item specified in the INPUT statement. A record that is being held by the double trailing at sign (@@) is not released until the input pointer moves past the end of the record, or until an INPUT statement that has no line-hold specifier executes

2. Correct answer: b

   Unlike the double trailing at sign (@@), the single trailing at sign (@) is automatically released when control returns to the top of the DATA step for the next iteration. The trailing @ does not toggle on and off. If another INPUT statement that has a trailing @ executes, the holding effect is still on.

3. Correct answer: c

   Each record in this file contains three repeating blocks of data values for Item and Variety. The INPUT statement reads a block of values for Item and Variety, and then holds the current record by using the double-trailing at sign (@@). The values in the program data vector are written to the data set as the first observation. In the next iteration, the INPUT statement reads the next block of values for Item and Variety from the same record.

4. Correct answer: d

   This raw data file contains an ID field followed by repeating fields. The first INPUT statement reads the values for ID and uses the @ line-hold specifier to hold the current record for the next INPUT statement in the DATA step. The second INPUT statement reads the values for Quantity. When all of the repeating fields have been read, control returns to the top of the DATA step, and the record is released.

5. Correct answer: a

The iterative DO statement begins the execution of a loop based on the value of an index variable. Here, the loop executes when the value of count ranges from **`1`** to **`50`**, incremented by 5.

6. Correct answer: a

The OUTPUT statement must be in the loop so that each time a value for Activity is read, an observation is immediately written to the data set.

7. Correct answer: c

The DO WHILE statement checks for the condition that Cholesterol is greater than 200. The expression must be enclosed in parentheses. The expression is evaluated at the top of the loop before the loop executes. If the condition is true, the DO WHILE loop executes. If the expression is false the first time it is evaluated, the loop does not execute.

8. Correct answer: b

The sum statement adds the result of an expression to an accumulator variable. The + sign is an essential part of the sum statement. Here, the value of TotalPay is incremented by 1.

9. Correct answer: d

The first input statement reads STORE and SALES and holds the record. Month is initialized to 0. The DO loop executes while SALES is not missing. Inside the loop, month increments, an observation is output, the next SALES value is read, and the record is held. The loop stops when a missing value of SALES is read. Control returns to the top of the DATA step, the held record is released, and the input statement reads the next record.

10. Correct answer: c

This DATA step produces one observation for each repeating field. The MISSOVER option in the INFILE statement prevents SAS from reading the next record when missing values occur at the end of a record. Every observation contains one value for Flavor, paired with the corresponding value for ID. Because there are 12 values for Flavor, there are 12 observations in the data set.