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

by SAS Institute

Chapter 11: Reading SAS Data Sets

Overview

Introduction

You've learned about creating a SAS data set from raw data. However, you might often want to create a new data set from an existing SAS data set. To create the new data set, you can read a data set using the DATA step. As you read the data set, you can use all the programming features of the DATA step to manipulate your data.

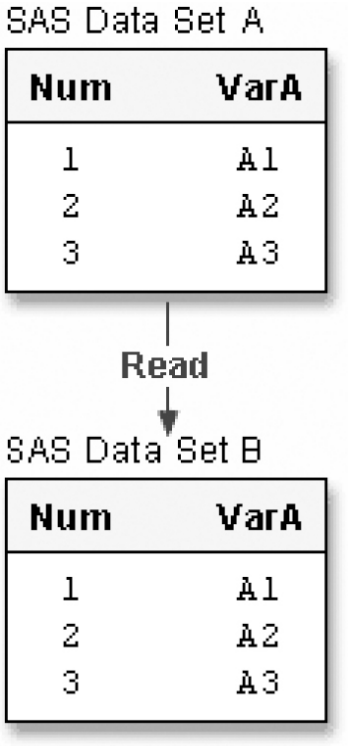


Figure 11.1: Data Set Diagram

This chapter shows you how to use the DATA step to read an existing SAS data set. When you create your new data set, you can choose variables, select observations based on one or more conditions, and assign values conditionally. You can also assign variable attributes such as formats and labels.

Additional Note You can also merge, concatenate, or interleave two or more data sets. For details, see Chapter 5, "Creating SAS Data Sets from External Files," on page 151 and "Combining SAS Data Sets" on page 360.

Objectives

In this chapter, you learn to

- create a new data set from an existing data set
- use BY groups to process observations
- read observations by observation number
- stop processing when necessary
- explicitly write observations to an output data set
- detect the last observation in a data set

- identify differences in DATA step processing for raw data and SAS data sets.

Reading a Single Data Set

The data set `sasuser.admit` contains health information about patients in a clinic, their activity level, height and weight. Suppose you want to create a small data set containing all the men in the group who are older than fifty.

To create the data set, you must first reference the library in which `admit` is stored and then the library in which you want to store the `males` data set. Then you write a DATA step to read your data and create a new data set.

General form, basic DATA step for reading a single data set:

```
DATA SAS-data-set;
        SET SAS-data-set;
    <more SAS statements>
RUN;
```

where

- *SAS-data-set* in the DATA statement is the name (**`libref.filename`**) of the SAS data set to be created
- *SAS-data-set* in the SET statement is the name (**`libref.filename`**) of the SAS data set to be read.

After you write a DATA step to name the SAS data set to be created, you specify the data set that will be read in the SET statement. The DATA step below reads all observations and variables from the existing data set **`admit`** into the new data set **`males`**. The DATA statement creates the permanent SAS data set `males`, which is stored in the SAS library **`Men50`**. The SET statement reads the permanent SAS data set **`admit`** and subsets the data using a WHERE statement. The new data set, **`males`**, contains all males in **`sasuser.admit`** who are older than 50.

```
libname sasuser "C:\Users\name\sasuser\";
libname Men50 "C:\Users\name\sasuser\Men50";
data Men50.males;
set sasuser.admit;

      where sex='M' and age>50;
run;
```

When you submit this DATA step, the following messages appear in the log, confirming that the new data set was created:

```
134  data Men50.males;
135     set sasuser.admit;
136     where sex="M" and age>50;
137  run;

NOTE: There were 3 observations read from the data set SASUSER.ADMIT.
      WHERE (sex='M') and (age>50);
NOTE: The data set MEN50.MALES has 3 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Figure 11.2: SAS Log Output

You can add a PROC PRINT statement to this same example to see the output of `Men50.males`.

```
proc print data=Men50.males;
   title "Men Over 50";
run;
```

Men Over 50

Obs	ID	Name	Sex	Age	Date	Height	Weight	ActLevel	Fee
1	2539	LaMance, K	M	51	4	71	158	LOW	124.80
2	2579	Underwood, K	M	60	22	71	191	LOW	149.75

3	2595	Warren, C	M	54	7	71	183	MOD	149.75

Figure 11.3: PROC PRINT Output For Data Set males

Manipulating Data

Overview

In the previous example, you read in the data set `admit` and used the WHERE statement in the DATA step to subset the data. For any data set you read in, you can use any of the programming features of the DATA step to manipulate your data.

For example, you can use any of the statements and data set options that you learned in previous chapters.

Table 11.1: Manipulating Data Using the DATA Step

To do this…	Use this type of statement…
Subset data	`if resthr<70 then delete;` `if tolerance='D';`
Drop unwanted variables	`drop timemin timesec;`
Create or modify a variable	`TotalTime=(timemin*60)+timesec;`
Initialize and retain a variable Accumulate values	`retain SumSec 5400;` `sumsec+totaltime;`
Specify a variable's length	`length TestLength $ 6;`

Execute statements conditionally	`if totaltime>800 then TestLength='Long';`
	`else if 750<=totaltime<=800`
	`then TestLength='Normal';`
	`else if totaltime<750`
	`then TestLength='Short';`

Label a variableFormat a variable	`label sumsec='Cumulative Total Seconds';` `format sumsec comma6.;`

Example

The following DATA step reads the data set lab23.drug1h, selects observations and variables, and creates new variables.

```
data lab23.drug1h(drop=placebo uric);
   set research.cltrials(drop=triglyc);
      if sex='M' then delete;
      if placebo='YES';
      retain TestData='22MAY1999'd;
      retain Days 30;
      days+1;
      length Retest $ 5;
      if cholesterol>190 then retest='YES';
        else if 150<=cholesterol<=190 then retest='CHECK';
        else if cholesterol<150 then retest='NO';
   label retest='Perform Cholesterol Test 2?';
   format enddate mmddyy10.;
run;
```

Where to Specify the DROP= and KEEP= Data Set Options

You've learned that you can specify the DROP= and KEEP= data set options anywhere you name a SAS data set. You can specify DROP= and KEEP= in either the DATA statement or the SET statement, depending on whether you want to

drop variables onto output or input:

- If you never reference certain variables and you don't want them to appear in the new data set, use a DROP= option in the SET statement.

  In the DATA step shown below, the DROP= or KEEP= option in the SET statement prevents the variables Triglycerides and UricAcid from being read. These variables won't appear in the Lab23.Drug1H data set.

```
data lab23.drug1h(drop=placebo);
   set research.cltrials(drop=triglycerides uricacid);
   if placebo='YES';
run;
```

- If you do need to reference a variable in the original data set (in a subsetting IF statement, for example), you can specify the variable in the DROP= or KEEP= option in the DATA statement. Otherwise, the statement that references the variable uses a missing value for that variable.

  This DATA step uses the variable Placebo to select observations. To drop Placebo from the new data set, the DROP= option must appear in the DATA statement.

```
data lab23.drug1h(drop=placebo);
   set research.cltrials(drop=triglycerides uricacid);
   if placebo='YES';
run;
```

When used in the DATA statement, the DROP= option simply drops the variables from the new data set. However, they are still read from the original data set and are available within the DATA step.

Using BY-Group Processing

Finding the First and Last Observations in a Group

"Creating List Reports" on page 112 explained how to use a BY statement in PROC SORT to sort observations and in PROC PRINT to group observations for subtotals. You can also use the BY statement in the DATA step to group observations for processing.

```
data temp;
   set salary;
   by dept;
run;
```

When you use the BY statement with the SET statement,

- the data sets that are listed in the SET statement must be sorted by the values of the BY variable(s), or they must have an appropriate index.
- the DATA step creates two temporary variables for each BY variable. One is named FIRST.variable, where variable is the name of the BY variable, and the other is named LAST.variable. Their values are either `1` or `0`. FIRST.variable and LAST.variable. identify the first and last observation in each BY group.

Table 11.2: Finding the First and Last Observations in a Group

This variable …	Equals …
FIRST.variable	1 for the first observation in a BY group 0 for any other observation in a BY group
LAST.variable	1 for the last observation in a BY group 0 for any other observation in a BY group

Example

To work with FIRST.variable and LAST.variable, let's look at a different set of data. The Company.USA data set contains payroll information for individual employees. Suppose you want to compute the annual payroll by department. Assume 2,000 work hours per year for hourly employees.

Before computing the annual payroll, you need to group observations by values of the variable Dept.

SAS Data Set Company.USA (Partial Listing)		
Dept	**WageCat**	**WageRate**
ADM20	S	3392.50
ADM30	S	5093.75
CAM 10	S	1813.30
CAM 10	S	1572.50
CAM 10	H	13.48
ADM30	S	2192.25

Figure 11.4: Partial Listing of Data Set

The following program computes the annual payroll by department. Notice that the variable name Dept has been appended to FIRST. and LAST.

```
proc sort data=company.usa out=work.temp;
   by dept;
run;
data company.budget(keep=dept payroll);
   set work.temp;
   by dept;
   if wagecat='S' then Yearly=wagerate*12;
   else if wagecat='H' then Yearly=wagerate*2000;
   if first.dept then Payroll=0;
   payroll+yearly;
   if last.dept;
run;
```

If you could look behind the scenes at the program data vector (PDV) as the Company.Budget data set is created, you would see the following. Notice the values for FIRST.Dept and LAST.Dept.

Selected PDV Variables				
_N_	**Dept**	**Payroll**	**FIRST.Dept**	**LAST.Dept**
1	ADM10	70929.0	**1**	0
2	ADM10	119479.2	0	0
3	ADM10	173245.2	0	0
4	ADM10	255516.0	0	0
5	ADM10	293472.0	0	**1**
1	ADM20	40710.0	**1**	0
2	ADM20	68010.0	0	0
3	ADM20	94980.0	0	0
4	ADM20	136020.0	0	0
5	ADM20	177330.0	0	**1**
1	ADM30	61125.0	**1**	0

Figure 11.5: Program Data Vector

When you print the new data set, you can now list and sum the annual payroll by department.

```
proc print data=company.budget noobs;
   sum payroll;
   format payroll dollar12.2;
run;
```

Dept	Payroll
ADM10	$293,472.00
ADM20	$177,330.00
ADM30	$173,388.00
CAM10	$130,709.60
CAM20	$156,731.20
	$931,630.80

Figure 11.6: Payroll Sum

Finding the First and Last Observations in Subgroups

When you specify multiple BY variables,

- FIRST.variable for each variable is set to 1 at the first occurrence of a new value for the primary variable
- a change in the value of a primary BY variable forces LAST.variable to equal *1* for the secondary BY variables.

Example

Suppose you now want to compute the annual payroll by job type for each manager. In your program, you specify two BY variables, Manager and JobType.

```
proc sort data=company.usa out=work.temp2;
   by manager jobtype;
   data company.budget2(keep=manager jobtype payroll);
   set work.temp2;
   by manager jobtype;
   if wagecat='S' then Yearly=wagerate*12;
   else if wagecat='H' then Yearly=wagerate*2000;
   if first.jobtype then Payroll=0;
   payroll+yearly;
   if last.jobtype;
run;
```

If you could look at the PDV now, you would see the following. Notice that the values for FIRST.JobType and LAST.JobType change according to values of FIRST.Manager and LAST.Manager.

Selected PDV Variables							
_N_	Manager	JobType	Payroll	FIRST.Manager	LAST.Manager	FIRST.JobType	LAST.Jobtype
1	Coxe	3	40710.0	**1**	0	**1**	**1**
2	Coxe	50	41040.0	0	0	**1**	0
3	Coxe	50	82350.0	0	0	0	**1**
4	Coxe	240	27300.0	0	0	**1**	0
5	Coxe	240	54270.0	0	**1**	0	**1**

6	Delgado	240	35520.0	**1**	0	**1**	0
7	Delgado	240	63120.0	0	0	0	**1**
8	Delgado	420	18870.0	0	0	**1**	0
9	Delgado	420	45830.0	0	0	0	**1**
10	Delgado	440	21759.6	0	**1**	**1**	**1**
11	Overby	1	82270.8	**1**	0	**1**	**1**
12	Overby	5	48550.2	0	0	**1**	**1**
13	Overby	10	53766.0	0	0	**1**	**1**
14	Overby	20	70929.0	0	0	**1**	0
15	Overby	20	108885.0	0	**1**	0	**1**

Figure 11.7: PDV

Now you can sum the annual payroll by job type for each manager. Here, the payroll for only two managers (Coxe and Delgado) is listed.

```
proc print data=company.budget2 noobs;;
   by manager;
   var jobtype;
   sum payroll;
   where manager in ('Coxe', 'Delgado');
   format payroll dollar12.2;
run;
```

Manager=Coxe

JobType	Payroll
3	$40,710.00
50	$123,390.00
240	$81,570.00
Manager	**$245,670.00**

Manager=Delgado

JobType	Payroll
240	$98,640.00
420	$64,700.00
440	$21,759.00
Manager	**$185,099.00**

Figure 11.8: Payroll Sum by Job Type and Manager

Reading Observations Using Direct Access

So far we have read the observations in an input data set sequentially. That is, we have accessed observations in the order in which they appear in the physical file. However, you can also access observations directly, by going straight to an observation in a SAS data set without having to process each observation that precedes it.

To access observations directly by their observation number, you use the POINT= option in the SET statement.

General form, POINT= option:

POINT=*variable*;

where *variable*

- specifies a temporary numeric variable that contains the observation number of the observation to read
- must be given a value before the SET statement is executed.

Example

Let's suppose you want to read only the fifth observation from a data set. In the following DATA step, the value 5 is assigned to the variable ObsNum. The POINT= option reads the value of ObsNum to determine which observation to read from the data set Company.USA.

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
run;
```

But let's see what would happen if you submitted this program.

As you learned in a previous chapter, the DATA step continues to read observations until it reaches the end-of-file marker in the input data. However, because the POINT= option reads only specified observations, SAS cannot read an end-of-file indicator as it would if the file were being read sequentially. So submitting the following program would cause continuous looping.

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
run;
```

Preventing Continuous Looping with POINT=

Because there is no end-of-file condition when you use direct access to read data, you must use a STOP statement to prevent continuous looping. The STOP statement causes SAS to stop processing the current DATA step immediately and to resume processing statements after the end of the current DATA step.

General form, STOP statement:

STOP;

So, if you add a STOP statement, your program no longer loops continuously.

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
   stop;
run;
```

But it doesn't write any observations to output, either! Remember from "Understanding DATA Step Processing" on page 200 that the DATA step writes observations to output at the end of the DATA step. However, in this program, the STOP statement immediately stops processing before the end of the DATA step.

Let's see how you can write the observation to output before processing stops.

Writing Observations Explicitly

To override the default way in which the DATA step writes observations to output, you can use an OUTPUT statement in the DATA step. Placing an explicit OUTPUT statement in a DATA step overrides the automatic output, so that observations are added to a data set only when the explicit OUTPUT statement is executed.

General form, OUTPUT statement:

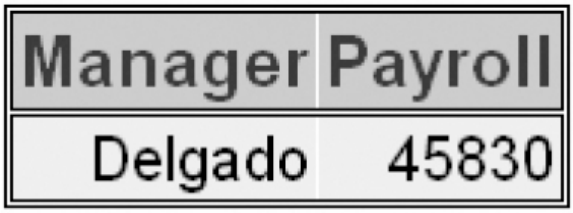
OUTPUT *<SAS-data-set(s)>;*

where *SAS-data-set(s)* names the data set(s) to which the observation is written. All data set names that are specified in the OUTPUT statement *must* also appear in the DATA statement.

Using an OUTPUT statement without a following data set name causes the current observation to be written to all data sets that are named in the DATA statement.

With an OUTPUT statement, your program now writes a single observation to output—observation 5.

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
   output;
   stop;
run;
proc print data=work.getobs5 noobs;
run;
```



Manager	Payroll
Delgado	45830

Figure 11.9: Single Observation

Suppose your DATA statement contains two data set names, and you include an OUTPUT statement that references only one of the data sets. The DATA step will create both data sets, but only the data set that is named in the OUTPUT statement will contain output. For example, the program below creates two temporary data sets, Empty and Full. The result of this DATA step is that the data set Empty is created but contains no observations, and the data set Full contains all of the observations from Company.Usa.

```
data empty full;
   set company.usa;
   output full;
run;
```

More Complex Ways of Using Direct Access

To convey concepts clearly, the examples in this section have been as simple as possible. However, most uses of the POINT= option are more complex. For example, POINT= is commonly used to efficiently select a random sample from a data set.

You can see more complex examples of using POINT= in "Generating Data with DO Loops" on page 463.

Detecting the End of a Data Set

Overview

Instead of reading specific observations, you might want to determine when the last observation in an input data set has been read, so that you can perform specific processing. For example, you might want to output only one observation that contains grand totals for numeric variables.

To create a temporary numeric variable whose value is used to detect the last observation, you can use the END= option

in the SET statement.

General form, END= option:
END=*variable*

where *variable* creates and names a temporary variable that contains an end-of-file marker. The variable, which is initialized to 0, is set to 1 when the SET statement reads the last observation of the data set.

This variable is *not* added to the data set.

Additional Note Do not specify END= with POINT=. POINT= reads only a specific observation, so the last observation in the data set is not encountered.

Example

Suppose you want to sum the number of seconds for treadmill stress tests. If you submit the following program, you produce a new data set that contains cumulative totals for each of the values of TotalTime.

```
data work.addtoend(drop=timemin timesec);
   set sasuser.stress2(keep=timemin timesec);
   TotalMin+timemin;
   TotalSec+timesec;
   TotalTime=totalmin*60+totalsec;
run;
proc print data=work.addtoend noobs;
run;
```

TotalMin	TotalSec	TotalTime
12	38	758
22	43	1363
33	56	2036
42	98	2618
53	144	3324
65	170	4070
80	211	5011
93	224	5804
103	246	6426
119	295	7435
134	297	8337
146	308	9068
160	317	9917
171	343	10603
185	370	11470
198	389	12269
214	396	13236
231	431	14291
246	472	15232
260	529	16129
272	539	16859

Figure 11.10: Data Set with Cumulative Totals for Each of the Values of TotalTime

But what if you want only the *final* total (the last observation) in the new data set? The following program uses the END= variable last to select only the last observation of the data set. You specify END= in the SET statement and last wherever you need it in processing (here, in the subsetting IF statement).

```
data work.addtoend(drop=timemin timesec);
   set sasuser.stress2(keep=timemin timesec) end=last;
   TotalMin+timemin;
   TotalSec+timesec;
   TotalTime=totalmin*60+totalsec;
   if last;
run;
proc print data=work.addtoend noobs;
run;
```

Now the new data set has one observation:

TotalMin	TotalSec	TotalTime
272	539	16859

Figure 11.11: Data Set with One Observation

Understanding How Data Sets Are Read

In a previous chapter, you learned about the compilation and execution phases of the DATA step as they pertain to reading raw data. DATA step processing for reading existing SAS data sets is very similar. The main difference is that while reading an existing data set with the SET statement, SAS retains the values of existing variables from one observation to the next.

Let's briefly look at the compilation and execution phases of DATA steps that use a SET statement. In this example, the DATA step reads the data set Finance.Loans, creates the variable Interest, and creates the new data set Finance.DueJan.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans				
Account	Amount	Rate	Months	Payment
101-1092	22000	0.100	60	467.43
101-1731	114000	0.095	360	958.57
101-1289	10000	0.105	36	325.02
101-3144	3500	0.105	12	308.52

Figure 11.12: New Data Set Finance.DueJan

Compilation Phase

1. The program data vector is created and contains the automatic variables _N_ and _ERROR_.

Program Data Vector

_N_	_ERROR_	
1	0	

Figure 11.13: program data vector with Automatic Variables

2. SAS also scans each statement in the DATA step, looking for syntax errors.
3. When the SET statement is compiled, a slot is added to the program data vector for each variable in the input data set. The input data set supplies the variable names, as well as attributes such as type and length.

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment

4. Any variables that are created in the DATA step are also added to the program data vector. The attributes of each of these variables are determined by the expression in the statement.

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest

5. At the bottom of the DATA step, the compilation phase is complete, and the descriptor portion of the new SAS data set is created. There are no observations because the DATA step has not yet executed.

When the compilation phase is complete, the execution phase begins.

Execution Phase

1. The DATA step executes once for each observation in the input data set. For example, this DATA step will execute four times because there are four observations in the input data set Finance.Loans.

2. At the beginning of the execution phase, the value of _N_ is 1. Because there are no data errors, the value of _ERROR_ is 0. The remaining variables are initialized to missing. Missing numeric values are represented by a period, and missing character values are represented by a blank.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

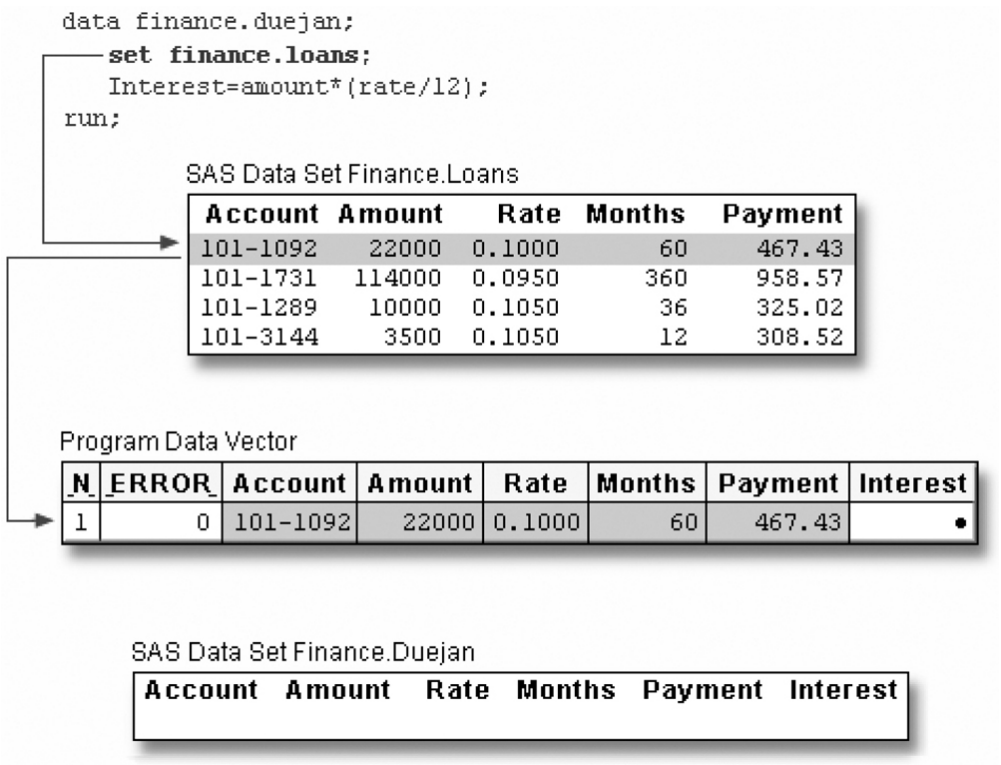
_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
1	0		.	.	.	.	.

Character Value | Numeric Values

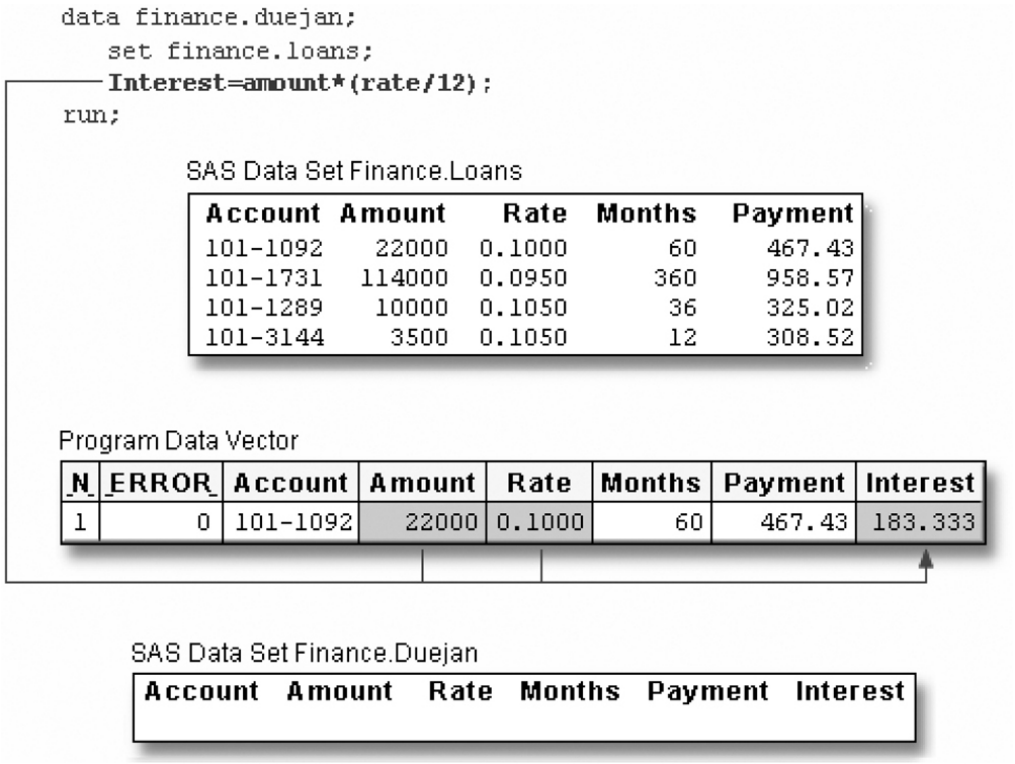
SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest

3. The SET statement reads the first observation from the input data set into the program data vector.



4. Then, the assignment statement executes to compute the value for Interest.



5. At the end of the first iteration of the DATA step, the values in the program data vector are written to the new data set as the first observation.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
1	0	101-1092	22000	0.1000	60	467.43	183.333

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

6. The value of _N_ increments from 1 to 2, and control returns to the top of the DATA step. Remember, the automatic variable _N_ keeps track of how many times the DATA step has begun to execute.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	183.333

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

7. SAS retains the values of variables that were read from a SAS data set with the SET statement, or that were created by a sum statement. All other variable values, such as the values of the variable Interest, are set to missing.

```
data finance.duejan;
    set finance.loans;
    Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	.

Values Retained — Reset to Missing

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

Additional Note When SAS reads raw data, the situation is different. In that case, SAS sets the value of each variable in the DATA step to missing at the beginning of each iteration, with these exceptions:

- variables named in a RETAIN statement
- variables created in a sum statement
- elements in a _TEMPORARY_ array
- any variables created by using options in the FILE or INFILE statements
- automatic variables.

8. At the beginning of the second iteration, the value of _ERROR_ is reset to `0`.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1092	22000	0.1000	60	467.43	.

SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

9. As the SET statement executes, the values from the second observation are read into the program data vector.

```
data finance.duejan;
   set finance.loans;
   Interest=amount*(rate/12);
run;
```

SAS Data Set Finance.Loans

Account	Amount	Rate	Months	Payment
101-1092	22000	0.1000	60	467.43
101-1731	114000	0.0950	360	958.57
101-1289	10000	0.1050	36	325.02
101-3144	3500	0.1050	12	308.52

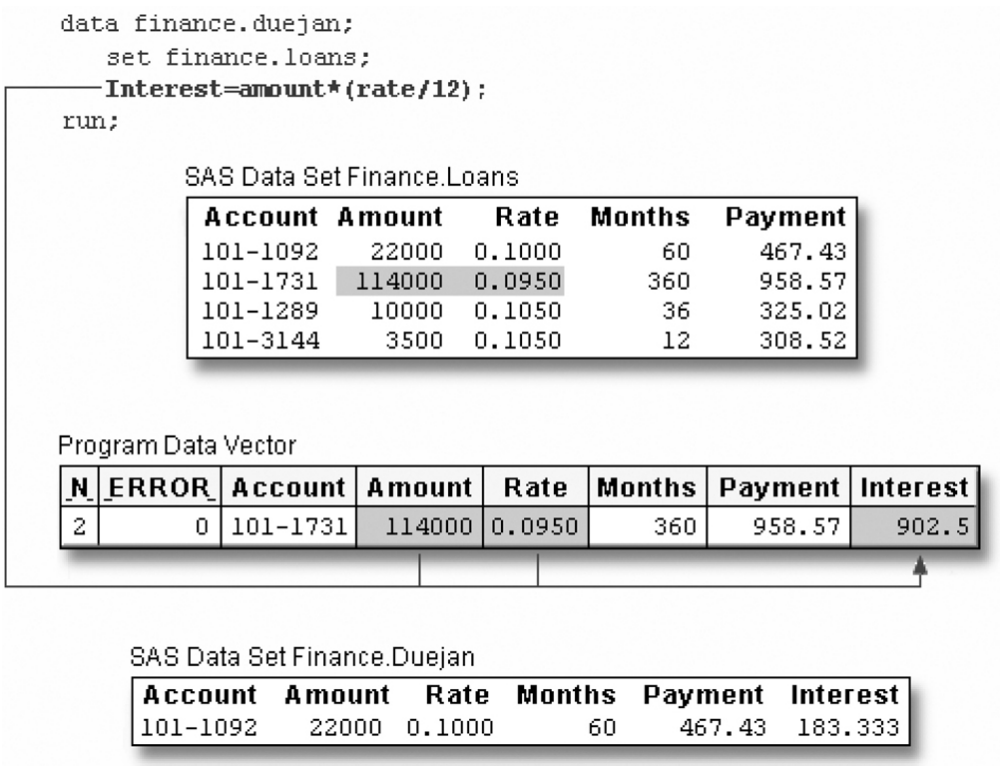
Program Data Vector

_N_	_ERROR_	Account	Amount	Rate	Months	Payment	Interest
2	0	101-1731	114000	0.0950	360	958.57	.

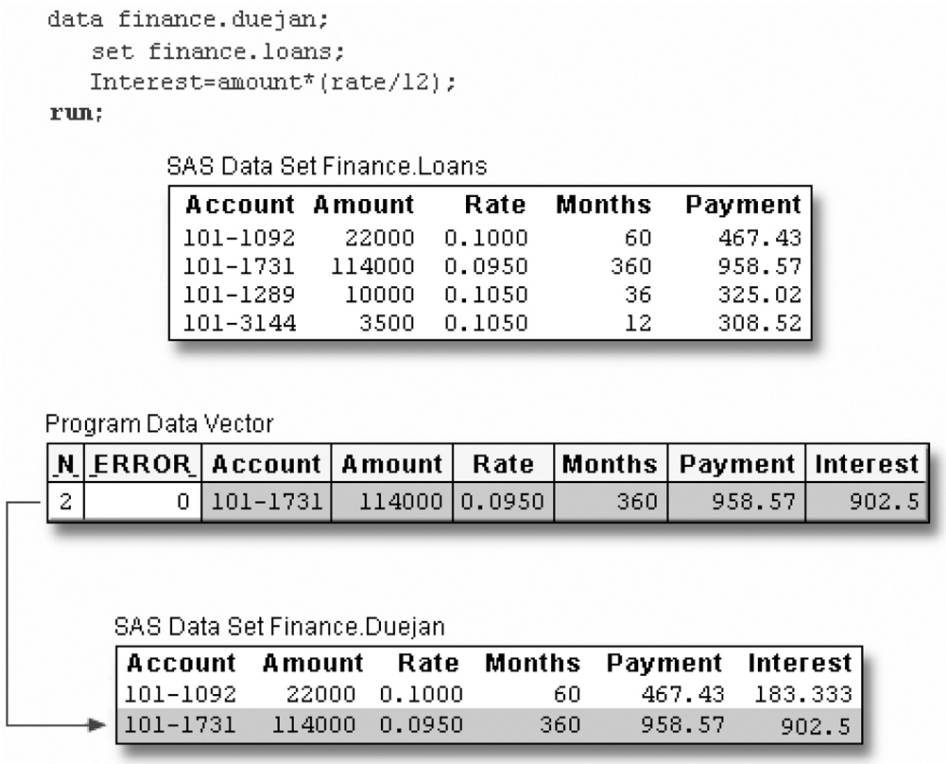
SAS Data Set Finance.Duejan

Account	Amount	Rate	Months	Payment	Interest
101-1092	22000	0.1000	60	467.43	183.333

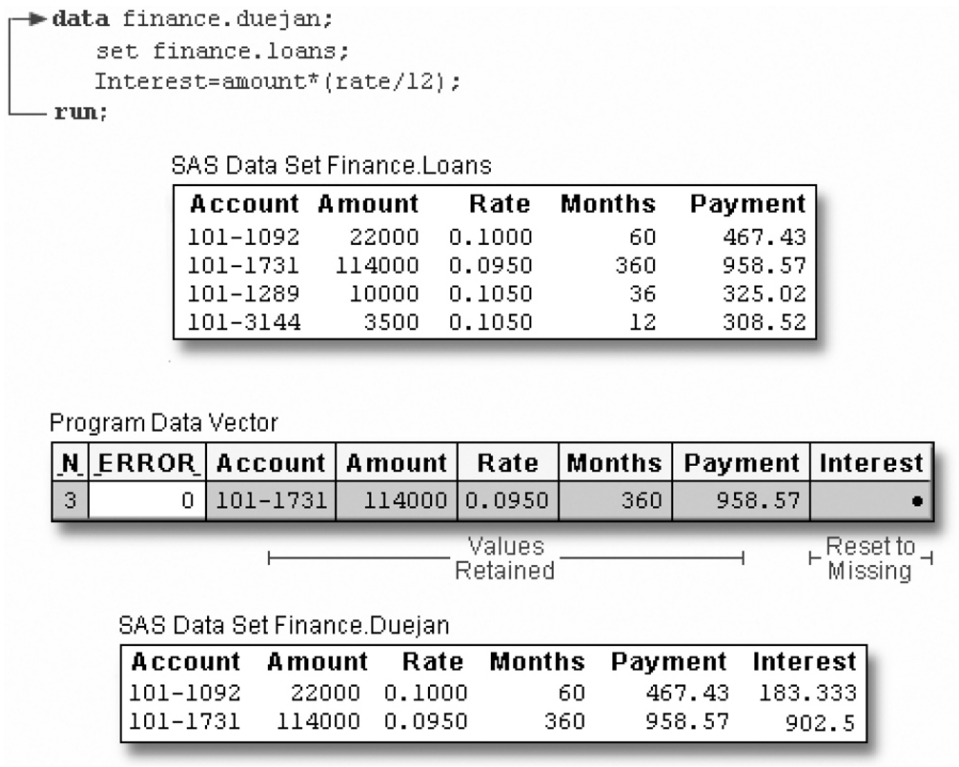
10. The assignment statement executes again to compute the value for Interest for the second observation.



11. At the bottom of the DATA step, the values in the program data vector are written to the data set as the second observation.



12. The value of _N_ increments from 2 to 3, and control returns to the top of the DATA step. SAS retains the values of variables that were read from a SAS data set with the SET statement, or that were created by a sum statement. All other variable values, such as the values of the variable Interest, are set to missing.



This process continues until all of the observations are read.

Additional Features

The DATA step provides many other programming features for manipulating data sets. For example, you can

- use IF-THEN/ELSE logic with DO groups and DO loops to control processing based on one or more conditions
- specify additional data set options
- process variables in arrays
- use SAS functions.

You can also combine SAS data sets in several ways, including match merging, interleaving, one-to-one merging, and updating.

Chapter Summary

Text Summary

Setting Up

Before you can create a new data set, you must assign a libref to the SAS library that will store the data set.

Reading a Single Data Set

After you have referenced the library in which your data set is stored, you can write a DATA step to name the SAS data set to be created. You then specify the data set to be read in the SET statement.

Selecting Variables

You can select the variables that you want to drop or keep by using the DROP= or KEEP= data set options in parentheses after a SAS data set name. For convenience, use DROP= if more variables are kept than dropped.

BY-Group Processing

Use the BY statement in the DATA step to group observations for processing. When you use the BY statement with the

SET statement, the DATA step automatically creates two temporary variables, FIRST. and LAST. When you specify multiple BY variables, a change in the value of a primary BY variable forces LAST.variable to equal 1 for the secondary BY variables.

Reading Observations Using Direct Access

In addition to reading input data sequentially, you can access observations directly by using the POINT= option to go directly to a specified observation. There is no end-of-file condition when you use direct access, so include an explicit OUTPUT statement and then the STOP statement to prevent continuous looping.

Detecting the End of a Data Set

To determine when the last observation in an input data set has been read, use the END= option in the SET statement. The specified variable is initialized to 0, then set to 1 when the SET statement reads the last observation of the data set.

Syntax

```
LIBNAME libref'SAS-data-library';
DATA SAS-data-set;
       SET SAS-data-set (KEEP= variable-1 <…variable-n>)|(DROP= variable-1
        <…variable-n>)
       POINT=variable | END=variable;
       OUTPUT <SAS-data-set>;
       STOP;
RUN;
```

Sample Program

```
proc sort data=company.usa out=work.temp2;
   by manager jobtype;
run;

data company.budget2(keep=manager jobtype payroll);
   set work.temp2;
   by manager jobtype;
   if wagecat='S' then Yearly=wagerate*12;
   else if wagecat='H' then Yearly=wagerate*2000;
   if first.jobtype then Payroll=0;
   payroll+yearly;
   if last.jobtype;
run;

data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
   output;
   stop;
run;

data work.addtoend(drop=timemin timesec);
   set sasuser.stress2(keep=timemin timesec) end=last;
   TotalMin+timemin;
   TotalSec+timesec;
   TotalTime=totalmin*60+totalsec;
   if last;
run;
```

Points to Remember

- When you perform BY-group processing, the data sets listed in the SET statement must either be sorted by the values of the BY variable(s), or they must have an appropriate index.
- When using direct access to read data, you *must* prevent continuous looping. Add a STOP statement to the DATA step.
- Do not specify the END= option with the POINT= option in a SET statement.

Chapter Quiz

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

1. If you submit the following program, which variables appear in the new data set? ?

```
data work.cardiac(drop=age group);
   set clinic.fitness(keep=age weight group);
   if group=2 and age>40;
run;
```

a. none

b. Weight

c. Age, Group

d. Age, Weight, Group

2. Which of the following programs correctly reads the data set Orders and creates the data set FastOrdr? ?

a.
```
data catalog.fastordr(drop=ordrtime);
   set july.orders(keep=product units price);
   if ordrtime<4;
   Total=units*price;
run;
```

b.
```
data catalog.orders(drop=ordrtime);
   set july.fastordr(keep=product units price);
   if ordrtime<4;
   Total=units*price;
run;
```

c.
```
data catalog.fastordr(drop=ordrtime);
   set july.orders(keep=product units price
                   ordrtime);
   if ordrtime<4;
   Total=units*price;
run;
```

d. none of the above

3. Which of the following statements is *false* about BY-group processing? ?

When you use the BY statement with the SET statement:

a. The data sets listed in the SET statement must be indexed or sorted by the values of the BY variable (s).

b. The DATA step automatically creates two variables, FIRST. and LAST., for each variable in the BY statement.

c. FIRST. and LAST. identify the first and last observation in each BY group, respectively.

d. FIRST. and LAST. are stored in the data set.

4. There are 500 observations in the data set Usa. What is the result of submitting the following program? ?

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
   stop;
run;
```

a. an error

b. an empty data set

c. continuous loop

d. a data set that contains one observation

5. There is no end-of-file condition when you use direct access to read data, so how can your program prevent a continuous loop? ?

a. Do not use a POINT= variable.

b. Check for an invalid value of the POINT= variable.

c. Do not use an END= variable.

d. Include an OUTPUT statement.

6. Assuming that the data set Company.USA has five or more observations, what is the result of submitting the following program? ?

```
data work.getobs5;
   obsnum=5;
   set company.usa(keep=manager payroll) point=obsnum;
   output;
   stop;
run;
```

a. an error

b. an empty data set

c. a continuous loop

d. a data set that contains one observation

7. Which of the following statements is *true* regarding direct access of data sets? ?

a. You cannot specify END= with POINT=.

b. You cannot specify OUTPUT with POINT=.

c. You cannot specify STOP with END=.

d. You cannot specify FIRST. with LAST.

8. What is the result of submitting the following program? ?

```
data work.addtoend;
   set clinic.stress2 end=last;
   if last;
run;
```

a. an error

b. an empty data set

c. a continuous loop

d. a data set that contains one observation

9. At the start of DATA step processing, during the compilation phase, variables are created in the program data vector (PDV), and observations are set to: ?

a. blank

b. missing

c. 0

d. there are no observations.

10. The DATA step executes: ?

a. continuously if you use the POINT= option and the STOP statement.

b. once for each variable in the output data set.

c. once for each observation in the input data set.

d. until it encounters an OUTPUT statement.

Answers

1. Correct answer: b

The variables Age, Weight, and Group are specified using the KEEP= option in the SET statement. After processing, Age and Group are dropped in the DATA statement.

2. Correct answer: c

You specify the data set to be created in the DATA statement. The DROP= data set option prevents variables from being written to the data set. Because you use the variable OrdrTime when processing your data, you cannot drop OrdrTime in the SET statement. If you use the KEEP= option in the SET statement, then you must list OrdrTime as one of the variables to be kept.

3. Correct answer: d

When you use the BY statement with the SET statement, the DATA step creates the temporary variables FIRST. and LAST. They are not stored in the data set.

4. Correct answer: b

The DATA step outputs observations at the end of the DATA step. However, in this program, the STOP statement stops processing before the end of the DATA step. An explicit OUTPUT statement is needed in order to produce an observation.

5. Correct answer: b

To avoid a continuous loop when using direct access, either include a STOP statement or use programming logic that executes a STOP statement when the data step encounters an invalid value of the POINT= variable. If SAS reads an invalid value of the POINT= variable, it sets the automatic variable _ERROR_ to 1. You can use this information to check for conditions that cause continuous looping.

6. Correct answer: d

By combining the POINT= option with the OUTPUT and STOP statements, your program can output a single observation.

7. Correct answer: a

The END= option and POINT= option are incompatible in the same SET statement. Use one or the other in your program.

8. Correct answer: d

This program uses the END= option to name a temporary variable that contains an end-of-file marker. That variable — last — is set to 1 when the SET statement reads the last observation of the data set.

9. Correct answer: d

At the bottom of the DATA step, the compilation phase is complete, and the descriptor portion of the new SAS data set is created. There are no observations because the DATA step has not yet executed.

10. Correct answer: c

The DATA step executes once for each observation in the input data set. You use the POINT= option with the STOP statement to prevent continuous looping.