Modifying Variables

Selected Useful Statements

Here are examples of statements that accomplish specific data-manipulation tasks.

Table 9.4 *Manipulating Data Using the DATA Step*

Task	Example Code
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	`retain SumSec 5400;`
Accumulate totals	`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';`

The following topics discuss these tasks.

Accumulating Totals

To add the result of an expression to an accumulator variable, you can use a sum statement in your DATA step.

Syntax, sum statement:

variable+*expression*;

- *variable* specifies the name of the accumulator variable. This variable must be numeric. The variable is automatically set to 0 before the first observation is read. The variable's value is retained from one DATA step execution to the next.
- *expression* is any valid SAS expression.

Note: If the expression produces a missing value, the sum statement ignores it.

The sum statement is one of the few SAS statements that do not begin with a keyword.

The sum statement adds the result of the expression that is on the right side of the plus sign (+) to the numeric variable that is on the left side of the plus sign. The value of the accumulator variable is initialized to 0 instead of missing before the first iteration of the DATA step. Subsequently, the variable's value is retained from one iteration to the next.

Example: Accumulating Totals

To find the total number of elapsed seconds in treadmill stress tests, you need the variable SumSec, whose value begins at 0 and increases by the amount of the total seconds in each observation. To calculate the total number of elapsed seconds in treadmill stress tests, use the sum statement shown below:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  SumSec+totaltime;
run;
```

The value of the variable on the left side of the plus sign, SumSec, begins at 0 and increases by the value of TotalTime with each observation.

SumSec	=	TotalTime	+	Previous total
0				
758	=	758	+	0
1363	=	605	+	758
2036	=	673	+	1363
2618	=	582	+	2036
3324	=	706	+	2618

Initializing Sum Variables

In the previous example, the sum variable SumSec was initialized to 0 before the first observation was read. However, you can initialize SumSec to a different number than 0.

Use the RETAIN statement to assign an initial value, other than 0, to an accumulator variable in a sum statement.

The RETAIN statement has several purposes:

- It assigns an initial value to a retained variable.
- It prevents variables from being initialized each time the DATA step executes.

Syntax, RETAIN statement for initializing sum variables:

RETAIN *variable* <*initial-value*>;

- *variable* is a variable whose values you want to retain.
- *initial-value* specifies an initial value (numeric or character) for the preceding variable.

Note: The following statements are true about the RETAIN statement:

- It is a compile-time-only statement that creates variables if they do not already exist.
- It initializes the retained variable to missing before the first execution of the DATA step if you do not supply an initial value.
- It has no effect on variables that are read with SET, MERGE, or UPDATE statements.

Example: RETAIN Statement

Suppose you want to add 5400 seconds (the accumulated total seconds from a previous treadmill stress test) to the variable SumSec in the StressTest data set when you create the data set. To initialize SumSec with the value 5400, use the RETAIN statement shown below. Now the value of SumSec begins at 5400 and increases by the value of TotalTime with each observation.

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
run;
proc print data=work.stresstest;
run;
```

SumSec	=	TotalTime	+	Previous Total
5400				
6158	=	758	+	5400
6763	=	605	+	6158
7436	=	673	+	6763
8018	=	582	+	7436
8724	=	706	+	8018

Subsetting Data

Using a Subsetting IF Statement

The subsetting IF statement causes the DATA step to continue processing only those observations that meet the condition of the expression specified in the IF statement. The resulting SAS data set or data sets contain a subset of the original external file or SAS data set.

Syntax, subsetting IF statement:

IF *expression*;

expression is any valid SAS expression.

- If the expression is true, the DATA step continues to process that observation.
- If the expression is false, no further statements are processed for that observation, and control returns to the top of the DATA step.

Example: Subsetting IF Statement

The subsetting IF statement below selects only observations whose values for Tolerance are `D`. It is positioned in the DATA step for efficiency: other statements do not need to process unwanted observations.

```
data work.stresstest;
  set cert.tests;
  if tolerance='D';
    TotalTime=(timemin*60)+timesec;
run;
proc print data=work.stresstest;
run;
```

Because Tolerance is a character variable, the value `D` must be enclosed in quotation marks, and it must be the same case as in the data set.

Notice that, in the output below, only the values where Tolerance contains the value of `D` are displayed and TotalTime was calculated.

Output 9.4 *Subsetted Data of Work.StressTest*

Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
1	2458	Murray, W	72	185	128	12	38	D	758
2	2539	LaMance, K	75	168	141	11	46	D	706
3	2552	Reberson, P	69	158	139	15	41	D	941
4	2572	Oberon, M	74	177	138	12	11	D	731
5	2574	Peterson, V	80	164	137	14	9	D	849
6	2584	Takahashi, Y	76	163	135	16	7	D	967

Categorizing Values

Suppose you want to create a variable that categorizes the length of time that a subject spends on the treadmill during a stress test. This new variable, TestLength, is based on the value of the existing variable TotalTime. The value of TestLength is assigned conditionally:

Value for TotalTime	Resulting Value for TestLength
greater than 800	Long
750 - 800	Normal
less than 750	Short

To perform an action conditionally, use an IF-THEN statement. The IF-THEN statement executes a SAS statement when the condition in the IF clause is true.

Syntax, IF-THEN statement:

IF *expression* **THEN** *statement*;

- *expression* is any valid SAS expression.
- *statement* is any executable SAS statement.

Example: IF-THEN Statement

To assign the value **Long** to the variable TestLength when the value of TotalTime is greater than 800, add the following IF-THEN statement to your DATA step:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  if totaltime>800 then TestLength='Long';
run;
```

SAS executes the assignment statement only when the condition (TotalTime>800) is true. If the condition is false, the value of TestLength is missing.

Examples: Logical Operators

The following examples use IF-THEN statements with logical operators:

- Use the AND operator to execute the THEN statement if both expressions that are linked by AND are true.

```
if status='OK' and type=3
    then Count+1;
if (age^=agecheck | time^=3)
    & error=1 then Test=1;
```

- Use the OR operator to execute the THEN statement if either expression that is linked by OR is true.

```
if (age^=agecheck | time^=3)
    & error=1 then Test=1;
if status='S' or cond='E'
    then Control='Stop';
```

- Use the NOT operator with other operators to reverse the logic of a comparison.

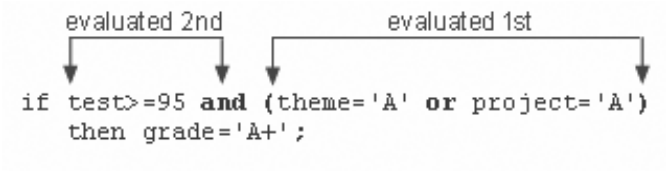
```
if not(loghours<7500)
    then Schedule='Quarterly';
if region not in ('NE','SE')
    then Bonus=200;
```

- Character values must be specified in the same case in which they appear in the data set and must be enclosed in quotation marks.

```
if status='OK' and type=3
    then Count+1;
if status='S' or cond='E'
    then Control='Stop';
if not(loghours<7500)
    then Schedule='Quarterly';
if region not in ('NE','SE')
    then Bonus=200;
```

Logical comparisons that are enclosed in parentheses are evaluated as true or false before they are compared to other expressions. In the example below, the OR comparison in parenthesis is evaluated before the first expression and the AND operator are evaluated.

Figure 9.3 *Example of a Logical Comparison*



Therefore, be careful when using the OR operator with a series of comparisons. Remember that only one comparison in a series of OR comparisons must be true to make a condition true, and any nonzero, not missing constant is always evaluated as true. Therefore, the following subsetting IF statement is always true:

```
if x=1 or 2;
```

SAS first evaluates x=1, and the result can be either true or false. However, since the 2 is evaluated as nonzero and not missing (true), the entire expression is true. In this statement, however, the condition is not necessarily true because either comparison can be evaluated as true or false:

```
if x=1 or x=2;
```

Note: Both sides of the OR must contain complete expressions.

Providing an Alternative Action

Suppose you want to assign a value to TestLength based on the other possible values of TotalTime. One way to do this is to add IF-THEN statements for the other two conditions.

```
if totaltime>800 then TestLength='Long';
if 750<=totaltime<=800 then TestLength='Normal';
if totaltime<750 then TestLength='Short';
```

However, when the DATA step executes, each IF statement is evaluated in order, even if the first condition is true. This wastes system resources and slows the processing of your program.

Instead of using a series of IF-THEN statements, you can use the ELSE statement to specify an alternative action to be performed when the condition in an IF-THEN statement is false. As shown below, you can write multiple IF-THEN/ELSE statements to specify a series of mutually exclusive conditions.

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

The ELSE statement must immediately follow the IF-THEN statement in your program. An ELSE statement executes only if the previous IF-THEN/ELSE statement is false.

Syntax, ELSE statement:

ELSE *statement***;**

statement is any executable SAS statement, including another IF-THEN statement.

To assign a value to TestLength when the condition in your IF-THEN statement is false, you can add the ELSE statement to your DATA step:

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
      else if 750<=totaltime<=800 then TestLength='Normal';
      else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

For greater efficiency, construct your IF-THEN/ELSE statements with conditions of decreasing probability.

TIP You can use PUT statements to test your conditional logic.

```
if totaltime>800 then TestLength='Long';
   else if 750<=totaltime<=800 then TestLength='Normal';
      else put 'NOTE: Check this Length: ' totaltime=;
run;
```

Deleting Unwanted Observations

You can specify any executable SAS statement in an IF-THEN statement. For example, you can use an IF-THEN statement with a DELETE statement to determine which observations to omit as you read data.

Syntax, DELETE statement:

DELETE;

To conditionally execute a DELETE statement, use the following syntax for an IF statement:

IF *expression* **THEN DELETE;**

The expression is evaluated as follows:

- If it is true, execution stops for that observation. The DELETE statement deletes the observation from the output data set, and control returns to the top of the DATA step.
- If it is false, the DELETE statement does not execute, and processing continues with the next statement in the DATA step.

Example: IF-THEN and DELETE Statements

In the following example, the IF-THEN and DELETE statements omit any observations whose values for RestHR are below 70.

```
data work.stresstest;
  set cert.tests;
  if resthr<70 then delete;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

Output 9.5 *Values for RestHR Less Than 70 Are Not in the Output (partial output)*

Obs	ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime	SumSec	TestLength
1	2458	Murray, W	72	185	128	12	38	D	758	6158	Normal
2	2501	Bonaventure, T	78	177	139	11	13	I	673	6831	Short
3	2539	LaMance, K	75	168	141	11	46	D	706	7537	Short
4	2544	Jones, M	79	187	136	12	26	N	746	8283	Short
5	2555	King, E	70	167	122	13	13	I	793	9076	Normal
			. . . more observations. . .								
12	2579	Underwood, K	72	165	127	13	19	S	799	14639	Normal
13	2584	Takahashi, Y	76	163	135	16	7	D	967	15606	Long
14	2588	Ivan, H	70	182	126	15	41	N	941	16547	Long
15	2589	Wilcox, E	78	189	138	14	57	I	897	17444	Long
16	2595	Warren, C	77	170	136	12	10	S	730	18174	Short

Selecting Variables

You might want to read and process variables that you do not want to keep in your output data set. In this case, use the DROP= and KEEP= data set options to specify the variables to drop or keep.

Use the KEEP= option instead of the DROP= option if more variables are dropped than kept.

Syntax, DROP=, and KEEP= data set options:

(DROP=*variable(s)***)**

(KEEP=*variable(s)***)**

- The DROP= or KEEP= options, in parentheses, follow the names of the data sets that contain the variables to be dropped or kept.
- *variable(s)* identifies the variables to drop or keep.

Example: DROP Data Set Option

Suppose you want to use theTimeMin and TimeSec variables to calculate the total time in the TotalTime variable, but you do not want to keep them in the output data set. You want to keep only the TotalTime variable. When you use the DROP data set option, the TimeMin and TimeSec variables are not written to the output data set:

```
data work.stresstest (drop=timemin timesec);
  set cert.tests;
  if resthr<70 then delete;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
      else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

Output 9.6 *StressTest Data Set with Dropped Variables (partial output)*

Obs	ID	Name	RestHR	MaxHR	RecHR	Tolerance	TotalTime	SumSec	TestLength
1	2458	Murray, W	72	185	128	D	758	6158	Normal
2	2501	Bonaventure, T	78	177	139	I	673	6831	Short
3	2539	LaMance, K	75	168	141	D	706	7537	Short
4	2544	Jones, M	79	187	136	N	746	8283	Short
5	2555	King, E	70	167	122	I	793	9076	Normal
. . . more observations. . .									
12	2579	Underwood, K	72	165	127	S	799	14639	Normal
13	2584	Takahashi, Y	76	163	135	D	967	15606	Long
14	2588	Ivan, H	70	182	126	N	941	16547	Long
15	2589	Wilcox, E	78	189	138	I	897	17444	Long
16	2595	Warren, C	77	170	136	S	730	18174	Short

Another way to exclude variables from a data set is to use the DROP statement or the KEEP statement. Like the DROP= and KEEP= data set options, these statements drop or keep variables. However, the DROP and KEEP statements differ from the DROP= and KEEP= data set options in the following ways:

- You cannot use the DROP and KEEP statements in SAS procedure steps.
- The DROP and KEEP statements apply to all output data sets that are named in the DATA statement. To exclude variables from some data sets but not from others, use the DROP= and KEEP= data set options in the DATA statement.

The KEEP statement is similar to the DROP statement, except that the KEEP statement specifies a list of variables to write to output data sets. Use the KEEP statement instead of the DROP statement if the number of variables to keep is smaller than the number to drop.

Syntax, DROP, and KEEP statements:

DROP *variable(s);*

KEEP *variable(s);*

variable(s) identifies the variables to drop or keep.

Example: Using the DROP Statement

The following example uses the DROP statement to drop unwanted variables.

```
data work.stresstest;
  set cert.tests;
  if tolerance='D';
  drop timemin timesec;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6;
  if totaltime>800 then TestLength='Long';
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

BY-Group Processing

Definitions

BY-group processing
: is a method of processing observations from one or more SAS data sets that are grouped or ordered by values of one or more common variables.

BY variable
: names a variable or variables by which the data set is sorted. All data sets must be ordered by the values of the BY variable.

BY value
: is the value of the BY variable.

BY group
: includes all observations with the same BY value. If you use more than one variable in a BY statement, a BY group is a group of observations with the same combination of values for these variables. Each BY group has a unique combination of values for the variables.

FIRST.*variable* and LAST.*variable*
: are variables that SAS creates for each BY variable. SAS sets FIRST.*variable* when it is processing the first observation in a BY group, and sets LAST.*variable* when it is processing the last observation in a BY group. These assignments enable you to take different actions, based on whether processing is starting for a new BY group or ending for a BY group.

Preprocessing Data

Determine Whether the Data Requires Preprocessing

Before you perform BY-group processing on one or more data sets using the SET, MERGE, and UPDATE statements, you must check the data to determine whether it requires preprocessing. The data requires no preprocessing if the observations in all of the data sets occur in one of the following patterns:

- ascending or descending numeric order
- ascending or descending character order
- not alphabetical or numerical order, but grouped in some way, such as by calendar month

If the observations are not in the order that you want, sort the data set before using BY-group processing.

Example: Sorting Observations for BY-Group Processing

You can use the SORT procedure to change the physical order of the observations in the data set. You can either replace the original data set, or create a new, sorted data set by using the OUT= option of the SORT procedure. In this example, PROC SORT rearranges the observations in the data set Cert.Usa in ascending order based on the values of the variable Manager. Then, the sorted data is created as a new, sorted data set Work.Usa.

Note: The default sort order for the SORT procedure is ascending.

```
proc sort data=cert.usa out=work.usa;
  by manager;
run;
proc print data=work.usa;
run;
```

Specify the variables in the PROC SORT BY statement in the same order that you intend to specify them in subsequent DATA or PROC steps.

The following output shows the Work.Usa data set sorted by the variable Manager in ascending order.

Output 8.1 *Sorted Work.Usa Data Set*

Obs	Dept	WageCat	WageRate	Manager	JobType
1	ADM10	S	3392.50	Coxe	3
2	ADM10	S	3420.00	Coxe	50
3	ADM10	S	6862.50	Coxe	50
4	ADM10	H	13.65	Coxe	240
5	ADM20	S	4522.50	Coxe	240
6	ADM20	S	2960.00	Delgado	240
7	ADM20	S	5260.00	Delgado	240
8	ADM20	S	1572.50	Delgado	420
9	ADM30	S	3819.20	Delgado	420
10	ADM30	S	1813.30	Delgado	440
11	CAM10	S	6855.90	Overby	1
12	CAM10	S	4045.80	Overby	5
13	CAM20	S	4480.50	Overby	10
14	ADM10	S	5910.80	Overby	20
15	CAM10	S	9073.80	Overby	20

FIRST. and LAST. DATA Step Variables

How the DATA Step Identifies BY Groups

In the DATA step, SAS identifies the beginning and end of each BY group by creating the following two temporary variables:

- FIRST.*variable*
- LAST.*variable*

The temporary variables are available for DATA step programming, but they are not added to the output data set. Their values indicate whether an observation is one of the following positions:

- the first one in a BY group
- the last one in a BY group
- neither the first nor the last one in a BY group
- both first and last, as is the case when there is only one observation in a BY group

How SAS Determines FIRST.variable and LAST.variable

- When an observation is the first in a BY group, SAS sets the value of the FIRST.*variable* to 1. This happens when the value of the variable changed from the previous observation.
- For all other observations in the BY group, the value of FIRST.*variable* is 0.
- When an observation is the last in a BY group, SAS sets the value of LAST.*variable* to 1. This happens when the value of the variable changes in the next observation.
- For all other observations in the BY group, the value of LAST.*variable* is 0.
- For the last observation in a data set, the value of all LAST.*variable* variables are set to 1.

Example: Grouping Observations Using One BY Variable

In this example, the Cert.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 the values of the variable Dept.

Output 8.2 *Sample Data Set: Cert.Usa*

Obs	Dept	WageCat	WageRate	Manager	JobType
1	ADM10	S	3392.50	Coxe	3
2	ADM10	S	3420.00	Coxe	50
3	ADM10	S	6862.50	Coxe	50
4	ADM10	H	13.65	Coxe	240
5	ADM20	S	4522.50	Coxe	240
6	ADM20	S	2960.00	Delgado	240
7	ADM20	S	5260.00	Delgado	240
8	ADM20	S	1572.50	Delgado	420
9	ADM30	S	3819.20	Delgado	420
10	ADM30	S	1813.30	Delgado	440
11	CAM10	S	6855.90	Overby	1
12	CAM10	S	4045.80	Overby	5
13	CAM20	S	4480.50	Overby	10
14	ADM10	S	5910.80	Overby	20
15	CAM10	S	9073.80	Overby	20

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=cert.usa out=work.temp;                /*#1*/
  by dept;
run;
data work.budget(keep=dept payroll);                  /*#2*/
  set work.temp;
  by dept;                                            /*#3*/
  if wagecat='S' then Yearly=wagerate*12;             /*#4*/
    else if wagecat='H' then Yearly=wagerate*2000;
  if first.dept then Payroll=0;                       /*#5*/
  payroll+yearly;                                     /*#6*/
  if last.dept;                                       /*#7*/
run;
```

1 The SORT procedure sorts the data in Cert.Usa by the variable Dept. The results of the SORT procedure are stored in Work.Temp.

2 The KEEP= data set option keeps the variables Dept and Payroll in the output data set, Work.Budget.

3 The BY statement in a DATA step applies only to the SET statement. The data set Work.Temp must be sorted by the Dept variable for the BY statement to set up grouping variables. By specifying Dept as the variable, you can identify the first and last observations for each Dept group. The Dept groups are **ADM10**, **ADM20**, **ADM30**, **CAM10**, and **CAM20**.

4 The IF statement executes the statements conditionally. If the value for WageCat is **S**, then the variable Yearly contains the value of WageRate multiplied by 12. If the value of WageCat is **H**, then the variable Yearly contains the value of WageRate multiplied by 2000.

5 If the observation is the first observation for the variable Dept, initialize Payroll to **0**.

Note: FIRST.Dept variable is not written to the data set and does not appear in the output.

6 Add the value of Yearly to the value of Payroll.

7 If this observation is the last in the variable, Dept, then end. If not, then read the next observation.

Note: LAST.Dept variable is not written to the data set and does not appear in the output.

The following figure illustrates how SAS processes FIRST.Dept and LAST.Dept. Notice that the values of FIRST.Dept and LAST.Dept change as the value for Dept changes.

Figure 8.1 *BY Group for Dept*

_N_	Dept	Yearly	Payroll	FIRST.Dept	LAST.Dept
1	ADM10	40710.0	40710.0	1	0
2	ADM10	41040.0	81750.0	0	0
3	ADM10	82350.0	164100.0	0	0
4	ADM10	27300.0	191400.0	0	0
5	ADM10	70929.6	262329.6	0	1
6	ADM20	54270.0	54270.0	1	0
7	ADM20	35520.0	89790.0	0	0
8	ADM20	63120.0	152910.0	0	0
9	ADM20	18870.0	171780.0	0	1
10	ADM30	45830.4	45830.4	1	0
11	ADM30	21759.6	67590.0	0	1
12	CAM10	82270.8	82270.8	1	0
13	CAM10	48549.6	130820.4	0	0
14	CAM10	108885.6	239706.0	0	1
15	CAM20	53766.0	53766.0	1	1

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

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

Output 8.3 *PROC PRINT Output of Work.Budget: Sum of Payroll*

The SAS System

Dept	Payroll
ADM10	$262,329.60
ADM20	$171,780.00
ADM30	$67,590.00
CAM10	$239,706.00
CAM20	$53,766.00
	$795,171.60

Example: Grouping Observations Using Multiple BY Variables

Suppose you now want to compute the annual payroll by job type for each manager. In the following example, you specify two BY variables, Manager and JobType, creating two groups. The Manager group contains three subgroups: **Coxe**, **Delgado**, and **Overby**. The JobType subgroup contains nine subgroups: **1**, **3**, **5**, **10**, **20**, **50**, **240**, **420**, and **440**. Within these subgroups, you can identify the first and last observations for each of these subgroups.

```
proc sort data=cert.usa out=work.temp2;                /*#1*/
  by manager jobtype;
run;
data work.budget2 (keep=manager jobtype payroll);      /*#2*/
  set work.temp2;
  by manager jobtype;                                  /*#3*/
  if wagecat='S' then Yearly=wagerate*12;              /*#4*/
    else if wagecat='H' then Yearly=wagerate*2000;
  if first.jobtype then Payroll=0;                     /*#5*/
  payroll+yearly;                                      /*#6*/
  if last.jobtype;                                     /*#7*/
run;
```

1 The SORT procedure sorts the data in Cert.Usa by the variables Manager and JobType. The results of the SORT procedure are stored in Work.Temp2.

2 The KEEP= data set option specifies the variables Manager, JobType, and Payroll and writes the variables to the new data set, Work.Budget.

3 The BY statement in a DATA step applies only to the SET statement. The data set Work.Temp2 must be sorted by the Manager and JobType variables in order for the BY statement to set up grouping variables. The data set is sorted by the variable Manager first and then by JobType.

4 The IF statement executes the statements conditionally. If the value for WageCat is **S**, then the variable Yearly contains the value of WageRate multiplied by 12. If the value of WageCat is **H**, then the variable Yearly contains the value of WageRate multiplied by 2000.

5 If the observation is the first for JobType, then initialize Payroll to 0.

6 Add the value of Yearly to the value of Payroll.

7 If this observation is the last in the variable, JobType, then end. If not, then read the next observation.

The following figure illustrates how SAS processes FIRST.Manager, FIRST.JobType, LAST.Manager, and LAST.JobType. Notice how the values of FIRST.Manager and LAST.Manager change only when the Manager value changes. However, the values for FIRST.JobType and LAST.JobType values change multiple times even when the Manager value remains the same.

Figure 8.2 *Multiple BY Group Variables: Manager and JobType*

_N_	Manager	JobType	WageRate	Yearly	Payroll	FIRST.Manager	LAST.Manager	FIRST.JobType	LAST.JobType
1	Coxe	3	3392.50	40710.00	40710.00	1	0	1	1
2	Coxe	50	3420.00	41040.00	41040.00	0	0	1	0
3	Coxe	50	6862.50	82350.00	123390.00	0	0	0	1
4	Coxe	240	13.65	27300.00	27300.00	0	0	1	0
5	Coxe	240	4522.50	54270.00	81570.00	0	1	0	1
6	Delgado	240	2960.00	35520.00	35520.00	1	0	1	0
7	Delgado	240	5260.00	63120.00	98640.00	0	0	0	1
8	Delgado	420	1572.50	18870.00	18870.00	0	0	1	0
9	Delgado	420	3819.20	45830.40	64700.40	0	0	0	1
10	Delgado	440	1813.30	21759.60	21759.60	0	1	1	1
11	Overby	1	6855.90	82270.80	82270.80	1	0	1	1
12	Overby	5	4045.80	48549.60	48549.60	0	0	1	1
13	Overby	10	4480.50	53766.00	53766.00	0	0	1	1
14	Overby	20	5910.80	70929.60	70929.60	0	0	1	0
15	Overby	20	9073.80	108885.60	179815.20	0	1	0	1

You can generate a sum for the annual payroll by job type for each manager. The example below shows the payroll sum for only two managers, Coxe and Delgado.

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

Figure 8.3 *Payroll Sum by Job Type and Manager*

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	$64,700.40
440	$21,759.60
Manager	**$185,100.00**
	$430,770.00