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

by SAS Institute

Chapter 10: Creating and Managing Variables

Overview

Introduction

You've learned how to create a SAS data set from raw data that is stored in an external file. You've also learned how to subset observations and how to assign values to variables.

This chapter shows you additional techniques for creating and managing variables. In this chapter, you learn how to retain and accumulate values, assign variable values conditionally, select variables, and assign permanent labels and formats to variables.

Obs	ID	Name	RestHR	MaxHR	RecHR	Tolerance	Total Time	SumSec	TestLength
1	2458	Murray, W	72	185	128	D	758	6,158	Normal
2	2539	LaMance, K	75	168	141	D	706	6,864	Short
3	2572	Oberon, M	74	177	138	D	731	7,595	Short
4	2574	Peterson, V	80	164	137	D	849	8,444	Long
5	2584	Takahashi. Y	76	163	135	D	967	9,411	Long

Figure 10.1: Variables Displayed in a Data Set

Objectives

In this chapter, you learn how to

- create variables that accumulate variable values
- initialize retained variables
- assign values to variables conditionally
- specify an alternative action when a condition is false
- specify lengths for variables
- delete unwanted observations
- select variables
- assign permanent labels and formats.

Creating and Modifying Variables

Accumulating Totals

It is often useful to create a variable that accumulates the values of another variable.

Suppose you want to create the data set Clinic.Stress and to add a new variable, SumSec, to accumulate the total number of elapsed seconds in treadmill stress tests.

SAS Data Set Clinic.Stress (Partial Listing)								
ID	Name	RestHr	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	TotalTime
2458	Murray, W	72	185	128	12	38	D	758
2462	Almers, C	68	171	133	10	5	I	605
2501	Bonaventure, T	78	177	139	11	13	I	673
2523	Johnson, R	69	162	114	9	42	S	582

2539	LaMance. K	75	168	141	11	46	D	706

Figure 10.2: Clinic.Stress Data Set

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

General form, sum statement:
variable+*expression*;

where

- *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.

Caution If the *expression* produces a missing value, the sum statement ignores it. (Remember, however, that assignment statements assign a missing value if the *expression* produces a missing value.)

Additional Note The sum statement is one of the few SAS statements that doesn't 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. At the top of the DATA step, the value of the numeric variable is not set to missing as it usually is when reading raw data. Instead, the variable retains the new value in the program data vector for use in processing the next observation.

Example

To find the total number of elapsed seconds in treadmill stress tests, you need a variable (in this example, 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, you use the sum statement shown below:

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
            RecHR 35-37 TimeMin 39-40 TimeSec 42-43
            Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   SumSec+totaltime;
run;
```

The value of the variable on the left side of the plus sign (here, 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 a previous example, the sum variable SumSec was initialized to 0 by default before the first observation was read. But what if you want to initialize SumSec to a different number, such as the total seconds from previous treadmill stress tests?

You can use the RETAIN statement to assign an initial value other than the default value of 0 to a variable whose value is assigned by a sum statement.

The RETAIN statement

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

General form, simple RETAIN statement for initializing sum variables:
RETAIN *variable <initial;-value>*;

where

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

Additional Note The RETAIN statement

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

Example

Suppose you want to add 5400 seconds (the accumulated total seconds from a previous treadmill stress test) to the variable SumSec in the Clinic.Stress data set when you create the data set. To initialize SumSec with the value `5400`, you use the RETAIN statement shown below.

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
run;
```

Now the value of SumSec begins at `5400` and increases by the value of TotalTime with each observation.

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

Assigning Values Conditionally

Overview

In the previous section, you created the variable SumSec by using a sum statement to add total seconds from treadmill

stress. This time, let's create a variable that categorizes the length of time that a subject spends on the treadmill during a stress test. This new variable, TestLength, will be based on the value of the existing variable TotalTime. The value of TestLength will be assigned conditionally:

If TotalTime is …	then TestLength is …
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.

General form, IF-THEN statement:

```
IF expression THEN statement;
```

where

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

Example

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 clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   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 will be missing.

Comparison and Logical Operators

When writing IF-THEN statements, you can use any of the following comparison operators:

Operator	Comparison Operation
= or `eq`	equal to
^= or `ne`	not equal to
> or `gt`	greater than
< or `lt`	less than
>= or `ge`	greater than or equal to
<= or `le`	less than or equal to
`in`	equal to one of a list

Examples:

```
if test<85 and time<=20
   then Status='RETEST';
```

```
if region in ('NE','NW','SW')
   then Rate=fee-25;
if target gt 300 or sales ge 50000

   then Bonus=salary*.05;
```

You can also use these logical operators:

Operator	Logical Operation
&	and
|	or
^	not

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 parentheses is evaluated before the first expression and the AND operator are evaluated.

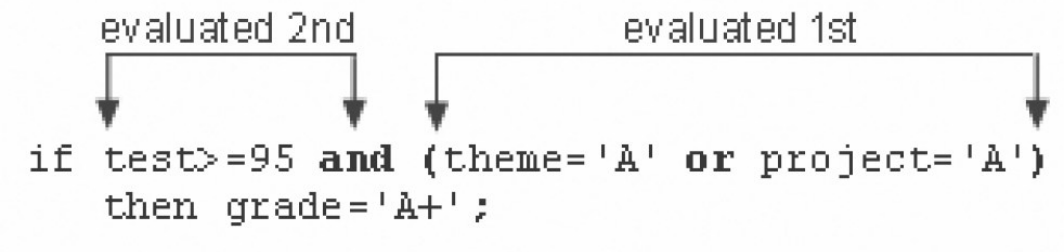


Figure 10.3: Example of a Logical Comparison

Caution In SAS, any numeric value other than 0 or missing is true, and a value of 0 or missing is false. Therefore, a numeric variable or expression can stand alone in a condition. If its value is a number other than 0 or missing, the condition is true. If its value is 0 or missing, the condition is false.

- 0 = False
- . = False
- 1 = True

As a result, you need to 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, nonmissing 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 nonmissing (true), the entire expression is true. In this statement, however, the condition is not necessarily true because either comparison can evaluate as true or false:

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

Providing an Alternative Action

Now 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, as shown below.

```
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 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.

General form, ELSE statement:

ELSE *statement*;

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

So, 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, as shown below:

```
data clinic.stress;
     infile tests;
     input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
           RecHR 35-37 TimeMin 39-40 TimeSec 42-43
           Tolerance $ 45;
     TotalTime=(timemin*60)+timesec;
     retain SumSec 5400;
     sumsec+totaltime;
     if totaltime>800 then TestLeng h='Long';
     else if 750<=totaltime<=800 then TestLength='Normal';
     else if totaltime<750 then TestLength='Short';
run;
```

Additional Note Using ELSE statements with IF-THEN statements can save resources:

- Using IF-THEN statements *without* the ELSE statement causes SAS to evaluate all IF-THEN statements.

- Using IF-THEN statements *with* the ELSE statement causes SAS to execute IF-THEN statements until it encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

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

Additional Note Remember that you can use PUT statements to test your conditional logic.

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   if totaltime>800 then TestLength='Long';
   else if 750<=totaltime<=800 then TestLength='Normal';
   else put 'NOTE: Check this Length:' totaltime=;
run;
```

Specifying Lengths for Variables

Overview

Previously, you added IF-THEN and ELSE statements to a DATA step in order to create the variable TestLength. Values for TestLength were assigned conditionally, based on the value for TotalTime.

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   if totaltime>800 then TestLength='Long';
   else if 750<=totaltime<=800 then TestLength='Normal';
   else if totaltime<750 then TestLength='Short';
run;
```

But look what happens when you submit this program.

During compilation, when creating a new character variable in an assignment statement, SAS allocates as many bytes of storage space as there are characters in the first value that it encounters for that variable. In this case, the first value for TestLength occurs in the IF-THEN statement, which specifies a four-character value (`Long`). So TestLength is assigned a length of 4, and any longer values (`Normal` and `Short`) are truncated.

Variable TestLength
(Partial Listing)

TestLength
Norm
Shor
Shor
Shor
Norm
Shor
Long
...

Figure 10.4: Truncated Variable Values

Additional Note The example above assigns a character constant as the value of the new variable. You can also see other examples of the default type and length that SAS assigns when the type and length of a variable are not explicitly set.

You can use a LENGTH statement to specify a length (the number of bytes) for TestLength before the first value is referenced elsewhere in the DATA step.

General form, LENGTH statement:

```
LENGTH variable(s) <$> length;
```

where

- *variable(s)* names the variable(s) to be assigned a length
- $ is specified if the variable is a character variable
- *length* is an integer that specifies the length of the variable.

Examples:

```
length Type $ 8;
length Address1 Address2 Address3
$200;
length FirstName $12 LastName
$16;
```

Within your program, you include a LENGTH statement to assign a length to accommodate the longest value of the variable TestLength. The longest value is `Normal`, which has six characters. Because TestLength is a character variable, you must follow the variable name with a dollar sign ($).

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   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;
```

Additional Note Make sure the LENGTH statement appears before any other reference to the variable in the DATA step. If the variable has been created by another statement, then a later use of the LENGTH statement will not change its length.

Now that you have added the LENGTH statement to your program, the values of TestLength are no longer truncated.

Variable TestLength
(Partial Listing)

TestLength
Normal
Short
Short
Short
Normal
Short
Long
...

Figure 10.5: Variable Values That Are Not Truncated

Subsetting Data

Deleting Unwanted Observations

So far in this chapter, you've learned to use IF-THEN statements to execute assignment statements conditionally. But 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.

- The IF-THEN statement executes a SAS statement when the condition in the IF clause is true.
- The DELETE statement stops the processing of the current observation.

General form, DELETE statement:

```
DELETE;
```

To conditionally execute a DELETE statement, you submit a statement in the following general form:

```
IF expression THEN DELETE;
```

If the expression is

- *true*, the DELETE statement executes, and control returns to the top of the DATA step (the observation is deleted).
- *false*, the DELETE statement does not execute, and processing continues with the next statement in the DATA step.

Example

The IF-THEN and DELETE statements below omit any observations whose values for RestHR are lower than `70`.

```
data clinic.stress;
   infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
          RecHR 35-37 TimeMin 39-40 TimeSec 42-43
          Tolerance $ 45;
   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;
```

Selecting Variables

Sometimes you might need to read and process variables that you don't want to keep in your data set. In this case, you can use the DROP= and KEEP= data set options to specify the variables that you want to drop or keep.

Use the KEEP= option instead of the DROP= option if more variables are dropped than kept. You specify data set options in parentheses after a SAS data set name.

General form, DROP= and KEEP= data set options:

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

where

- the `DROP=KEEP=` or option, in parentheses, follows the name of the data set that contains the variables to be dropped or kept
- *variable(s)* identifies the variables to drop or keep.

Example

Suppose you are interested in keeping only the new variable TotalTime and not the original variables TimeMin and TimeSec. You can drop TimeMin and TimeSec when you create the Stress data set.

```
data clinic.stress(drop=timemin timesec);
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   if tolerance='D';
   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;
```

SAS Data Set Clinic.Stress (Partial Listing)

ID	Name	RestHR	MaxHR	RecHR	Tolerance	TotalTime	SumSec	TestLength
2458	Murray, W	72	185	128	D	758	6158	Normal
2539	LaMance, K	75	168	141	D	706	6864	Short
2552	Reberson, P	69	158	139	D	941	7805	Long
2572	Oberon. M	74	177	138	D	731	8536	Short
2574	Peterson, V	80	164	137	D	849	9385	Long
2584	Takahashi, Y	76	163	135	D	967	10352	Long

Figure 10.6: Stress Data Set with Dropped Variables

Another way to exclude variables from your 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.

General form, DROP and KEEP statements:

```
DROP variable(s);
KEEP variable(s);
```

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

Example

The two programs below produce the same results. The first example uses the DROP= data set option. The second example uses the DROP statement.

```
data clinic.stress(drop=timemin timesec);
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   if tolerance='D';
   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;

data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   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;
```

Assigning Permanent Labels and Formats

Overview

At this point, you've seen raw data read and manipulated to obtain the observations, variables, and variable values that you want. Our final task in this chapter is to permanently assign labels and formats to variables.

If you read "Creating List Reports" on page 112, you learned to temporarily assign labels and formats within a PROC step. These temporary labels and formats are applicable only for the duration of the step. To permanently assign labels and formats, you use LABEL and FORMAT statements in DATA steps.

Additional Note Remember that labels and formats do not affect how data is stored in the data set, but only how it appears in output.

To do this…	Use this type of statement…
Reference a SAS data libraryReference an external file	`libname clinic 'c:\users\may\data';` `filename tests 'c:\users\tmill.dat';`

Name a SAS data setIdentify an external fileDescribe raw data	`data clinic.stress;`
	`  infile tests obs=10;`
	`  input ID $ 1-4 Name $ 6-25 …;`

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 variable	`retain SumSec 5400;`
Accumulate values	`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 variable	LABEL statement
Format a variable	FORMAT statement
Execute the DATA step	`run;`
List the data	`proc print data=clinic.stress label;`
Execute the final program step	`run;`

Example

To specify the label Cumulative Total Seconds (+5,400) and the format COMMA6. for the variable SumSec, you can

submit the following program:

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   if resthr<70 then delete;
   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';
   label sumsec='Cumulative Total Seconds (+5,400)';
   format sumsec comma6.;
run;
```

You're done! When we print the new data set, SumSec is labeled and formatted as specified. (Don't forget the LABEL option in the PROC PRINT statement.)

```
proc print data=clinic.stress label;
run;
```

Specifying the Label Comulative Total Seconds

Obs	ID	Name	RestHR	MaxHR	RecHR	Tolerance	TotalTime	Cumulative Total Seconds (+5,400)	TestLength
1	2458	Murray, W	72	185	128	D	758	6,158	Normal
2	2539	LaMance, K	75	168	141	D	706	6,864	Short
3	2572	Oberon, M	74	177	138	D	731	7,595	Short
4	2574	Peterson, V	80	164	137	D	849	8,444	Long
5	2584	Takahashi, Y	76	163	135	D	967	9,411	Long

Figure 10.7: Completed Clinic.Stress Data Set

Additional Note Remember that most SAS procedures automatically use permanent labels and formats in output, without requiring additional statements or options.

Additional Note If you assign temporary labels or formats within a PROC step, they override any permanent labels or formats that were assigned during the DATA step.

Assigning Values Conditionally Using SELECT Groups

Overview

Earlier in this chapter, you learned to assign values conditionally using IF-THEN/ELSE statements. You can also use SELECT groups in DATA steps to perform conditional processing. A SELECT group contains these statements:

This statement…	Performs this action…
SELECT	begins a SELECT group.
WHEN	identifies SAS statements that are executed when a particular condition is true.
OTHERWISE (optional)	specifies a statement to be executed if no WHEN condition is met.
END	ends a SELECT group.

You can decide whether to use IF-THEN/ELSE statements or SELECT groups based on personal preference.

General form, SELECT group:

```
SELECT <(select-expression)>;
     WHEN-1 (when-expression-1<…, when-expression-n>) statement;
     WHEN-n (when-expression-1 <…, when-expression-n>) statement;
     <OTHERWISEstatement;>
END;
```

where

- **SELECT** begins a SELECT group.
- the optional *select-expression* specifies any SAS expression that evaluates to a single value.
- **WHEN** identifies SAS statements that are executed when a particular condition is true.
- *when-expression* specifies any SAS expression, including a compound expression. You must specify at least one *when-expression*.
- *statement* is any executable SAS statement. You must specify the *statement* argument.
- the optional **OTHERWISE** statement specifies a statement to be executed if no WHEN condition is met.
- **END** ends a SELECT group.

Example: Basic SELECT Group

The following code is a simple example of a SELECT group. Notice that the variable a is specified in the SELECT statement, and various values to compare to a are specified in the WHEN statements. When the value of the variable a is

- `1, x` is multiplied by 10
- `3, 4`, or `5, x` is multiplied by 100
- `2` or any other value, nothing happens.

```
select a);
   when 1) x=x*10;
   when 3,4,5) x=x*100;
   otherwise;
end
```

Example: SELECT Group in a DATA Step

Now let's look at a SELECT group in context. In the DATA step below, the SELECT group assign values to the variable Group based on values of the variable JobCode. Most of the assignments are one-to-one correspondences, but ticket agents (the JobCode values `TA1, TA2`, and `TA3`) are grouped together, as are values in the Other category.

```
data emps(keep=salary group);
   set sasuser.payrollmaster;
   length Group $ 20;
   select(jobcode);
      when ("FA1") group="Flight Attendant I";
      when ("FA2") group="Flight Attendant II";
      when ("FA3") group="Flight Attendant III";
      when ("ME1") group="Mechanic I";
      when ("ME2") group="Mechanic II";
      when ("ME3") group="Mechanic III";
      when ("NA1") group="Navigator I";
      when ("NA2") group="Navigator II";
      when ("NA3") group="Navigator III";
      when ("NA1") group="Navigator I";
      when ("NA2") group="Navigator II";
```

```
      when ("NA3") group="Navigator III";
      when ("PT1") group="Pilot I";
      when ("PT2") group="Pilot II";
      when ("PT3") group="Pilot III";
      when ("TA1","TA2","TA3") group="Ticket Agents";
      otherwise group="Other";
   end;
run;
```

Notice that in this case the SELECT statement contains a select-expression. You are checking values of a single variable by using select(jobcode), which is more concise than eliminating the select-expression and repeating the variable in each when-expression, as in when(jobcode-'FA1').

Additional Note Notice that the LENGTH statement in the DATA step above specifies a length of 20 for Group. Remember that without the LENGTH statement, values for Group might be truncated, as the first value for Group (`Flight Attendant I`) is not the longest possible value.

Caution When you are comparing values in the when-expression, be sure to express the values exactly as they are coded in the data. For example, the when-expression below would be evaluated as false because the values for JobCode in Sasuser.Payrollmaster are stored in uppercase letters.

```
when ('fa1") group="Flight Attendant I";
```

In this case, given the SELECT group above, Group would be assigned the value `Other`.

As you saw in the general form for SELECT groups, you can optionally specify a select-expression in the SELECT statement. The way SAS evaluates a when-expression depends on whether you specify a select-expression.

Specifying SELECT Statements with Expressions

If you *do* specify a select-expression in the SELECT statement, SAS compares the value of the select-expression with the value of each when-expression. That is, SAS evaluates the select-expression and when-expression, compares the two for equality, and returns a value of true or false.

- If the comparison is *true*, SAS executes the statement in the WHEN statement.
- If the comparison is *false*, SAS proceeds either to the next when-expression in the current WHEN statement, or to the next WHEN statement if no more expressions are present. If no WHEN statements remain, execution proceeds to the OTHERWISE statement, if one is present.

Additional Note If the result of all SELECT-WHEN comparisons is false and no OTHERWISE statement is present, SAS issues an error message and stops executing the DATA step.

In the following SELECT group, SAS determines the value of toy and compares it to values in each WHEN statement in turn. If a WHEN statement is true compared to the toy value, SAS assigns the related price and continues processing the rest of the DATA step. If none of the comparisons is true, SAS executes the OTHERWISE statement and writes a debugging message to the SAS log.

```
select (toy);
   when ("Bear") price=35.00;
   when ("Violin") price=139.00;
   when ("Top","Whistle","Duck") price=7.99;
   otherwise put "Check unknown toy:" toy=;
end;
```

Specifying SELECT Statements without Expressions

If you *don't* specify a select-expression, SAS evaluates each when-expression to produce a result of true or false.

- If the result is *true*, SAS executes the statement in the WHEN statement.
- If the result is *false*, SAS proceeds either to the next when-expression in the current WHEN statement, or to the next WHEN statement if no more expressions are present, or to the OTHERWISE statement if one is present. (That is, SAS performs the action that is indicated in the first true WHEN statement.)

If more than one WHEN statement has a true when-expression, only the *first* WHEN statement is used. Once a when-expression is true, no other when-expressions are evaluated.

Additional Note If the result of all when-expressions is false and no OTHERWISE statement is present, SAS issues an error message.

In the example below, the SELECT statement does not specify a select-expression. The WHEN statements are evaluated in order, and only one is used. For example, if the value of toy is `Bear` and the value of month is `FEB`, only the second WHEN statement is used, even though the condition in the third WHEN statement is also met. In this case, the variable price is assigned the value `25.00`:

```
select;
   when (toy="Bear" and month in ('OCT', 'NOV', 'DEC')) price=45.00;
   when (toy="Bear" and month in ('JAN', 'FEB')) price=25.00;
   when (toy="Bear") price=35.00;
   otherwise;
end;
```

Grouping Statements Using DO Groups

Overview

So far in this chapter, you've seen examples of conditional processing (IF-THEN/ELSE statements and SELECT groups) that execute only a single SAS statement when a condition is true. However, you can also execute a group of statements as a unit by using DO groups.

To construct a DO group, you use the DO and END statements along with other SAS statements.

General form, simple DO group:

```
DO;
   SAS statements
END;
```

where

- the `DO` statement begins DO group processing
- *SAS statements* between the DO and END statements are called a DO group and execute as a unit
- the `END` statement terminates DO group processing.

Additional Note You can nest DO statements within DO groups.

You can use DO groups in IF-THEN/ELSE statements and SELECT groups to execute many statements as part of the conditional action.

Examples

In this simple DO group, the statements between DO and END are performed only when TotalTime is greater than 800. If TotalTime is less than or equal to 800, statements in the DO group do not execute, and the program continues with the assignment statement that follows the appropriate ELSE statement.

```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   retain SumSec 5400;
   sumsec+totaltime;
   length TestLength $ 6 Message $ 20;
   if totaltime>800 then
      do;
```

```
         testlength='Long';
         message='Run blood panel';
      end;
   else if 750<=totaltime<=800 then testlength='Normal';
   else if totaltime<750 then TestLength='Short';
run;
```

In the SELECT group below, the statements between DO and END are performed only when the value of Payclass is `hourly`. Notice that an IF-THEN statement appears in the DO group; the PUT statement executes only when Hours is greater than `40`. The second END statement in the program closes the SELECT group.

```
data payroll;
   set salaries;
   select(payclass);
   when ('monthly') atm=salary;
   when ('hourly')
      do;
         amt=hrlywage*min(hrs,40);
         if hrs>40 then put 'CHECK TIMECARD';
      end;
   otherwise put 'PROBLEM OBSERVATION';
   end;
run;
```

Indenting and Nesting DO Groups

You can nest DO groups to any level, just like you nest IF-THEN/ELSE statements. (The memory capabilities of your system may limit the number of nested DO statements that you can use. For details, see the SAS documentation about how many levels of nested DO statements your system's memory can support.)

The following is an example of nested DO groups:

```
do;
    statements;
       do;
                statements ;
                   do;
                           statements;
                   end;
       end;
end;
```

It is good practice to indent the statements in DO groups, as shown in the preceding statements, so that their position indicates the levels of nesting.

Additional Note There are three other forms of the DO statement:

- The iterative DO statement executes statements between DO and END statements repetitively based on the value of an index variable. The iterative DO statement can contain a WHILE or UNTIL clause.
- The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true, checking the condition after each iteration of the DO loop.
- The DO WHILE statement executes statements in a DO loop repetitively while a condition is true, checking the condition before each iteration of the DO loop.

You can learn about these forms of the DO statement in "Generating Data with DO Loops" on page 463.

Chapter Summary

Text Summary

Accumulating Totals

Use a sum statement to add the result of an expression to an accumulator variable.

Initializing and Retaining Variables

You can use the RETAIN statement to assign an initial value to a variable whose value is assigned by a sum statement.

Assigning Values Conditionally

To perform an action conditionally, use an IF-THEN statement. The IF-THEN statement executes a SAS statement when the condition in the IF expression is true. You can include comparison and logical operators; logical comparisons that are enclosed in parentheses are evaluated as true or false before other expressions are evaluated. Use the ELSE statement to specify an alternative action when the condition in an IF-THEN statement is false.

Specifying Lengths for Variables

When creating a new character variable, SAS allocates as many bytes of storage space as there are characters in the first value that it encounters for that variable at compile time. This can result in truncated values. You can use the LENGTH statement to specify a length before the variable's first value is referenced in the DATA step.

Subsetting Data

To omit observations as you read data, include the DELETE statement in an IF-THEN statement. If you need to read and process variables that you don't want to keep in the data set, use the DROP= and KEEP= data set options or the DROP and KEEP statements.

Assigning Permanent Labels and Formats

You can use LABEL and FORMAT statements in DATA steps to permanently assign labels and formats. These do not affect how data is stored in the data set, only how it appears in output.

Assigning Values Conditionally Using SELECT Groups

As an alternative to IF-THEN/ELSE statements, you can use SELECT groups in DATA steps to perform conditional processing.

Grouping Statements Using DO Groups

You can execute a group of statements as a unit with DO groups in DATA steps. You can use DO groups in IF-THEN/ELSE statements and SELECT groups to perform many statements as part of the conditional action.

Syntax

```
LIBNAME libref 'SAS-data-library';
FILENAME fileref 'filename';
DATA SAS-data-set(DROP=variable(s)|KEEP=variable(s));
      INFILE file-specification <OBS=n>;
      INPUT variable <$> startcol-endcol…;
      DROP variable(s);
      KEEP variable(s);
      RETAIN variable initial-value;
      variable+expression;
      LENGTH variable(s) <$> length;
      IF expressionTHEN statement;
      ELSE statement;
      IF expressionTHEN DELETE;
      LABEL variable1='label1' variable2='label2' …;
      FORMAT variable(s) format-name;
      SELECT <(select-expression)>;
        WHEN-1 (when-expression-1 <…, when-expression-n>)statement;
        WHEN-n (when-expression-1 <…, when-expression-n>)statement;
        <OTHERWISEstatement;>
      END;
RUN;
PROC PRINT DATA=SAS-data set LABEL;
RUN;
```

Sample Program

```
data clinic.stress;
   infile tests;
```

```
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   if tolerance='D'and resthr ge 70 then delete;
   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';
   label sumsec='Cumulative Total Seconds (+5,400)';
   format sumsec comma6.;
run;
```

Points to Remember

- Like the assignment statement, the sum statement does not contain a keyword.
- If the expression in a sum statement produces a missing value, the sum statement ignores it. (Remember, however, that assignment statements assign a missing value if the expression produces a missing value.)
- Using ELSE statements with IF-THEN statements can save resources. For greater efficiency, construct your IF-THEN/ELSE statements with conditions of decreasing probability.
- Make sure the LENGTH statement appears before any other reference to the variable in the DATA step. If the variable has been created by another statement, a later use of the LENGTH statement will not change its length.
- Labels and formats do not affect how data is stored in the data set, only how it appears in output. You assign labels and formats temporarily in PROC steps and permanently in DATA steps.

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. Which program creates the output shown below? ?

Raw Data File Furnture

```
1---+----10---+----20---+----30---+
310 oak   pedestal table 329.99
311 maple pedestal table 369.99
312 brass floor    lamp   79.99
313 glass table    lamp   59.99
314 oak   rocking  chair 153.99
```

Figure 10.8: Output 1

StockNum	Finish	Style	Item	TotalPrice
310	oak	pedestal	table	329.99
311	maple	pedestal	table	699.98
312	brass	floor	lamp	779.97
313	glass	table	lamp	839.96

Figure 10.9: Output 2

a.
```
data test2;
   infile furnture;
   input StockNum $ 1-3 Finish $ 5-9 Style $ 11-18
         Item $ 20-24 Price 26-31;
   if finish='oak' then delete;
   retain TotPrice 100;
   totalprice+price;
   drop price;
run;
proc print data=test2 noobs;
run;
```

b.
```
data test2;
   infile furnture;
   input StockNum $ 1-3 Finish $ 5-9 Style $ 11-18
         Item $ 20-24 Price 26-31;
   if finish='oak' and price<200 then delete;
   TotalPrice+price;
run;
proc print data=test2 noobs;
run;
```

c.
```
data test2(drop=price);
   infile furnture;
   input StockNum $ 1-3 Finish $ 5-9 Style $ 11-18
         Item $ 20-24 Price 26-31;
   if finish='oak' and price<200 then delete;
   TotalPrice+price;
run;
proc print data=test2 noobs;
run;
```

d.
```
data test2;
   infile furnture;
   input StockNum $ 1-3 Finish $ 5-9 Style $ 11-18
         Item $ 20-24 Price 26-31;
   if finish=oak and price<200 then delete price;
   TotalPrice+price;
run;
proc print data=test2 noobs;
run;
```

2. How is the variable Amount labeled and formatted in the PROC PRINT output? ?

```
data credit;
   infile creddata;
   input Account $ 1-5 Name $ 7-25 Type $ 27
            Transact $ 29-35 Amount 37-50;
   label amount='Amount of Loan';
```

```
   format amount dollar12.2;
run;
proc print data=credit label;
   label amount='Total Amount Loaned';
   format amount comma10.;
run;
```

a. label Amount of Loan, format DOLLAR12.2

b. label Total Amount Loaned, format COMMA10.

c. label Amount, default format

d. The PROC PRINT step does not execute because two labels and two formats are assigned to the same variable.

3. Consider the IF-THEN statement shown below. When the statement is executed, which expression is evaluated first? ?

```
if finlexam>=95
   and (research='A' or
       (project='A' and present='A'))
   then Grade='A+';
```

a. `finlexam>=95`

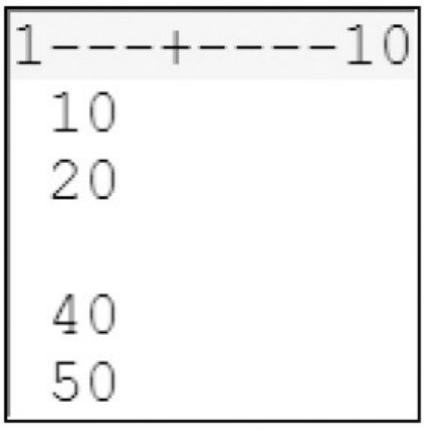
b. `research='A'`

c. `project='A' and present='A'`

d.
```
 research='A' or
(project='A' and present='A')
```

4. Consider the small raw data file and program shown below. What is the value of Count after the fourth record is read? ?

```
data work.newnums;
   infile numbers;
   input Tens 2-3;
   Count+tens;
run;
```



a. missing

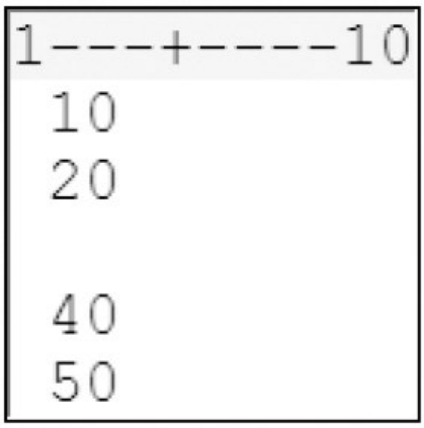
b. 0

c. 30

d. 70

5. Now consider the revised program below. What is the value of Count after the third observation is read? ?

```
data work.newnums;
   infile numbers;
   input Tens 2-3;
   retain Count 100;
   count+tens;
run;
```



a. missing

b. 0

c. 100

d. 130

6. For the observation shown below, what is the result of the IF-THEN statements? ?

Status	Type	Count	Action	Control
Ok	3	12	E	Go

```
if status='OK' and type=3
   then Count+1;
if status='S' or action='E'
   then Control='Stop';
```

a. `Count = 12  Control = Go`

b. `Count = 13  Control =Stop`

c. `Count = 12  Control =Stop`

d. `Count = 13  Control = Go`

7. Which of the following can determine the length of a new variable? ?

a. the length of the variable's first reference in the DATA step

b. the assignment statement

c. the LENGTH statement

d. all of the above

8. Which set of statements is equivalent to the code shown below? ?

```
if code='1' then Type='Fixed';
if code='2' then Type='Variable';
if code^='1' and code^='2' then Type='Unknown';
```

a.
```
if code='1' then Type='Fixed';
   else if code='2' then Type='Variable';
```

```
   else Type='Unknown';
```

b.
```
 if code='1' then Type='Fixed';
if code='2' then Type='Variable';
else Type='Unknown';
```

c.
```
if code='1' then type='Fixed';
else code='2' and type='Variable';
else type='Unknown';
```

d.
```
if code='1' and type='Fixed';
then code='2' and type='Variable';
else type='Unknown';
```

9. What is the length of the variable Type, as created in the DATA step below? ?

```
data finance.newloan;
   set finance.records;
   TotLoan+payment;
   if code='1' then Type='Fixed';
   else Type='Variable';
   length type $ 10;
run;
```

a. 5

b. 8

c. 10

d. it depends on the first value of Type

10. Which program contains an error? ?

a.
```
data clinic.stress(drop=timemin timesec);
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   SumSec+totaltime;
run;
```

b.
```
proc print data=clinic.stress;
   label totaltime='Total Duration of Test';
   format timemin 5.2;
   drop sumsec;
run;
```

c.
```
proc print data=clinic.stress(keep=totaltime timemin);
   label totaltime='Total Duration of Test';
   format timemin 5.2;
run;
```

d.
```
data clinic.stress;
   infile tests;
   input ID $ 1-4 Name $ 6-25 RestHR 27-29 MaxHR 31-33
         RecHR 35-37 TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
   TotalTime=(timemin*60)+timesec;
   keep id totaltime tolerance;
run;
```

Answers

1. Correct answer: c

   Program c correctly deletes the observation in which the value of Finish is `oak` and the value of Price is less than `200`. It also creates TotalPrice by summing the variable Price down observations, and then drops Price by using the DROP= data set option in the DATA statement.

2. Correct answer: b

   The PROC PRINT output displays the label Total Amount Loaned for the variable Amount and formats this variable using the COMMA10. format. Temporary labels or formats that are assigned in a PROC step override permanent labels or formats that are assigned in a DATA step.

3. Correct answer: c

   Logical comparisons that are enclosed in parentheses are evaluated as true or false before they are compared to other expressions. In the example above, the AND comparison within the nested parentheses is evaluated before being compared to the OR comparison.

4. Correct answer: d

   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. The new value is then retained for subsequent observations. The sum statement ignores the missing value, so the value of Count in the fourth observation would be `10+20+0+40`, or `70`.

5. Correct answer: d

   The RETAIN statement assigns an initial value of `100` to the variable Count, so the value of Count in the third observation would be `100+10+20+0`, or `130`.

6. Correct answer: c

   You must enclose character values in quotation marks, and you must specify them in the same case in which they appear in the data set. The value `Ok` is not identical to OK, so the value of Count is not changed by the IF-THEN statement.

7. Correct answer: d

   The length of a variable is determined by its first reference in the DATA step. When creating a new character variable, SAS allocates as many bytes of storage space as there are characters in the reference to that variable. The first reference to a new variable can also be made with a LENGTH statement or an assignment statement.

8. Correct answer: a

   You can write multiple ELSE statements to specify a series of mutually exclusive conditions. 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.

9. Correct answer: a

   The length of a new variable is determined by the first reference in the DATA step, not by data values. In this case, the length of Type is determined by the value Fixed. The LENGTH statement is in the wrong place; it must occur before any other reference to the variable in the DATA step.

10. Correct answer: b

To select variables, you can use a DROP or KEEP statement in any DATA step. You can also use the DROP= or KEEP= data set options following a data set name in any DATA or PROC step. However, you cannot use DROP or KEEP statements in PROC steps.