Chapter 5: Creating and Managing Variables

<SAS Certification Prep Guide - Base Programming for SAS 9>

Objectives : learning about

- create variables that accumulate variable values
- initialize values of accumulator 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. To add the result of an expression to an accumulator variable, you can use a Sum statement in your DATA step.

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 beginning 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.

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 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 ncreases by the value of TotalTime with each observation.

■ Initializing Accumulator Variables

In a previous example, the accumulator variable SumSec was initialized to 0 by default before the irst observation was read. But what if you want to initialize SumSec to a different number, such s the total seconds from previous treadmill stress tests?

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

```
data 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.

■ Assigning Values Conditionally

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.

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 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;
```

- Examples

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;
```

■ Providing an Alternative Action

Now suppose you want to assign a value to TestLength that is based on the other possible values of TotalTime. One way to do this is to add IF-THEN statements to 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.

```
data 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;
```

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

Specifying Lengths for Variables

■ Group Processing Using the CLASS Statement

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.

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.

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.

```
length TestLength $ 6;
```

cf) 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 size.

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 from the data set that SAS is creating as it reads raw data.

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

```
data 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 with the DROP= and KEEP= Data Set Options or Statements

Sometimes you might need to read and process fields 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.

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 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 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;
```

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 significantly smaller than the number to drop.

cf) There are some differences between Drop= (Keep=) statement and data set option, which we will not deal with in this chapter.

Grouping Statements Using DO Groups

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.

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 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;
```

You can nest DO groups to any level, just like you nest IF-THEN/ELSE statements. (The memory capabilities of your system might 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.)

```
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.

cf)
- 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.

Generating Data with DO Loops

You can execute SAS statements repeatedly by placing them in a DO loop. DO loops can execute any number of times in a single iteration of the DATA step.

This example calculates how much interest was earned each month for a one-yearinvestment.

```
data earnings;
        Amount=1000;
        Rate=.075/12;
        do month=1 to 12;
                Earned+(amount+earned)*(rate);
        end;
run;
```

■ Counting Iterations of DO Loops

In some cases, it is useful to create an index variable to count and store the

number of iterations in the DO loop. Then you can drop the index variable from the data set.

```
data earn (drop=counter);
        Value=2000;
        do counter=1 to 20;
                Interest=value*.075;
                value+interest;
                Year+1;
        end;
run;
```

The Sum statement Year+1 accumulates the number of iterations of the DO loop and stores the total in the new variable Year. The final value of Year is then stored in the data set, whereas the index variable counter is dropped. The data set has one observation.

■ **Explicit OUTPUT Statements**

To create an observation for each iteration of the DO loop, place an OUTPUT statement inside the loop. By default, every DATA step contains an implicit OUTPUT statement at the end of the step. But placing an explicit OUTPUT statement in a DATA step overrides automatic output, causing SAS to add an observation to the data set only when the explicit OUTPUT statement is executed.

```
data earn;
        Value=2000;
        do Year=1 to 20;
                Interest=value*.075;
                value+interest;
        Year+1;
        output;
        end;
run;
```

cf) You can decrement a DO loop's index variable by specifying a negative value for the BY clause. For example, the specification in this iterative DO statement decreases the index variable by 1, resulting in values of 5, 4, 3, 2, and 1.

```
DO index-variable=5 to 1 by -1;
        SAS statements
END;
```

■ Nesting DO Loops

Iterative DO statements can be executed within a DO loop. Putting a DO loop within a DO loop is called nesting.

The following DATA step computes the value of a one-year investment that earns 7.5% annual interest, compounded monthly.

```
data earn;
        Capital=2000;
        do month=1 to 12;
                Interest=capital*(.075/12);
                capital+interest;
        end;
run;
```

Let's assume the same amount of capital is to be added to the investment each year for 20 years. The new program must perform the calculation for each month during each of the 20 years. To do this, you can include the monthly calculations within another DO loop that executes 20 times.

```
data earn;
        do year=1 to 20;
                Capital+2000;
                do month=1 to 12;
                        Interest=capital*(.075/12);
                        capital+interest;
                end;
        end;
run;
```

During each iteration of the outside DO loop, an additional 2,000 is added to the capital, and the nested DO loop executes 12 times.

■ Using the DO UNTIL Statement

The DO UNTIL statement executes a DO loop until the expression is true.

The expression is not evaluated until the bottom of the loop, so a DO UNTIL loop always executes at least once. When the expression is evaluated as true, the DO loop is not executed again.

Assume you want to know how many years it will take to earn $50,000 if you deposit $2,000 each year into an account that earns 10% interest. The DATA step that follows uses a DO UNTIL statement to perform the calculation until the value is reached. Each iteration of the DO loop represents one year of earning.

```
data nvest;
        do until(Capital>=50000);
                capital+2000;
                capital+capital*.10;
                Year+1;
        end;
run;
```

Because there is no index variable in the DO UNTIL statement, the variable Year is created in a Sum statement to count the number of iterations of the DO loop. This program produces a data set that contains the single observation shown below. To accumulate more than $50,000 in capital requires 13 years (and 13 iterations of the DO loop).

■ Using the DO WHILE Statement

Like the DO UNTIL statement, the DO WHILE statement executes DO loops conditionally. You can use the DO WHILE statement to execute a DO loop while the expression is true.

An important difference between the DO UNTIL and DO WHILE statements is that the DO WHILE expression is evaluated at the top of the DO loop. If the expression is false the first time it is evaluated, then the DO loop never executes. For example, in the following program, if the value of Capital is less than 50,000, the DO loop does not execute.

```
data invest;
        do while(Capital>=50000);
                capital+2000;
                capital+capital*.10;
                Year+1;
        end;
run;
```

Because there is no index variable in the DO UNTIL statement, the variable Year is created in a Sum statement to count the number of iterations of the DO loop. This program produces a data set that contains the single observation shown below. To accumulate more than $50,000 in capital requires 13 years (and 13 iterations of the DO loop).

■ Using Conditional Clauses with the Iterative DO Statement

Like the DO UNTIL statement, the DO WHILE statement executes DO loops conditionally. You can use the DO WHILE statement to execute a DO loop while the expression is true.

Now let's look at a form of the iterative DO statement that combines features of both conditional and unconditional execution of DO loops.

In this DATA step, the DO UNTIL statement determines how many years it takes (13) for an investment to reach $50,000.

```
data invest;
        do until(Capital>=50000);
                Year+1;
                capital+2000;
                capital+capital*.10;
        end;
run;
```

Suppose you also want to limit the number of years that you invest your capital to 10 years. You can add the UNTIL or WHILE expression to an iterative DO statement to further control the number of iterations. This iterative DO statement enables you to execute the DO loop until Capital is greater than or equal to 50000 or until the DO loop executes 10 times, whichever occurs first.

```
data invest(drop=i);
        do i=1 to 10 until(Capital>=50000);
                Year+1;
                capital+2000;
                capital+capital*.10;
        end;
run;
```

In this case, the DO loop stops executing after 10 iterations, and the value of Capital never reaches 50000. If you increase the amount added to Capital each year to 4000, the DO loop stops executing after the eighth iteration when the value of Capital exceeds 50000.

```
data invest(drop=i);
        do i=1 to 10 until(Capital>=50000);
                Year+1;
                capital+4000;
                capital+capital*.10;
        end;
run;
```

sas-code

```
filename tests 'c:\users\stress.dat';
data 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;

data 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;

data 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;

data 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;

data 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;


data 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 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;

data 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;

data earnings;
        Amount=1000;
        Rate=.075/12;
        do month=1 to 12;
                Earned+(amount+earned)*(rate);
        end;
run;

data earn (drop=counter);
        Value=2000;
        do counter=1 to 20;
                Interest=value*.075;
                value+interest;
                Year+1;
        end;
run;

data earn;
        Value=2000;
```

```
        do Year=1 to 20;
                Interest=value*.075;
                value+interest;
        Year+1;
        output;
        end;
run;

data earn;
        Capital=2000;
        do month=1 to 12;
                Interest=capital*(.075/12);
                capital+interest;
        end;
run;

data earn;
        do year=1 to 20;
                Capital+2000;
                do month=1 to 12;
                        Interest=capital*(.075/12);
                        capital+interest;
                end;
        end;
run;

data invest;
        do until(Capital>=50000);
                capital+2000;
                capital+capital*.10;
                Year+1;
        end;
run;

data invest;
        do while(Capital>=50000);
                capital+2000;
                capital+capital*.10;
                Year+1;
        end;
run;

data invest;
        do until(Capital>=50000);
                Year+1;
                capital+2000;
                capital+capital*.10;
        end;
run;

data invest(drop=i);
        do i=1 to 10 until(Capital>=50000);
                Year+1;
                capital+2000;
                capital+capital*.10;
        end;
run;

data invest(drop=i);
        do i=1 to 10 until(Capital>=50000);
                Year+1;
                capital+4000;
                capital+capital*.10;
        end;
run;
```