# Chapter 10

# Creating and Managing Variables

# Overview

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

# Creating and Modifying Variables

## Accumulating Totals: the sum statement

*variable +expression;*

where

- variable specifies the name of the **accumulator variable**, which 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.

# Creating and Modifying Variables

## Accumulating Totals: the sum statement

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. For example,

```
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;
```

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

# Creating and Modifying Variables

## Initializing Sum Variables

By default the sum variable is initialized to 0 before the first observation was read, but you can initialize it to a different value by using the RETAIN statement. For example,

```
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;
```

**SumSec** is initialized to 5400; however, it is initialized to missing before the first execution of the DATA step if you do not supply an initial value.

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

# Assigning Values Conditionally

Use the IF-THEN statement to assign values conditionally.  For example,

```
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'; /*the value of TestLength is
        missing if TotalTime <= 800*/
run;
```

# Comparison and Logical Operators (Review)

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

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 |

Logical operators

| Operator | Logical Operation |
|---|---|
| & | and |
| \| | or |
| ^ | not |

# Comparison and Logical Operators

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; otherwise, 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, non-missing constant is always evaluated as true. Therefore, the following IF statement is always true:

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

However, the following condition is not necessarily true because either comparison can evaluate as true or false:

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

# Providing an Alternative Action

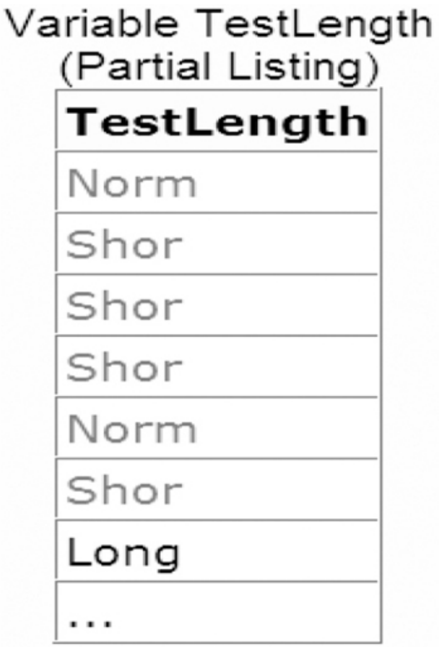
```
/*Each IF statement is evaluated in order, even if the first condition is true.
This wastes system resources and slows the processing of your program.*/
if totaltime>800 then TestLength='Long';
if 750<=totaltime<=800 then TestLength='Normal';
if totaltime<750 then TestLength='Short';

/*Use the ELSE statement to specify an alternative action to be performed
when the condition in an IF-THEN statement is false.*/
if totaltime>800 then TestLength='Long';
else if 750<=totaltime<=800 then TestLength='Normal';
else if totaltime<750 then TestLength='Short';
```

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

# Specifying Lengths for Variables

SAS determines a variable's **type** and **length** based on the value it sees **first time**. During compilation, the first value for TestLength occurs in the IF-THEN statement, which is **Long**, a 4-character value. Later on, any longer values (Normal and Shorter) will be truncated.

```
data clinic.stress;
     infile tests;
     input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32
          RecHR 34-36 TimeMin 38-39 TimeSec 41-42
          Tolerance $ 44;
     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;
```



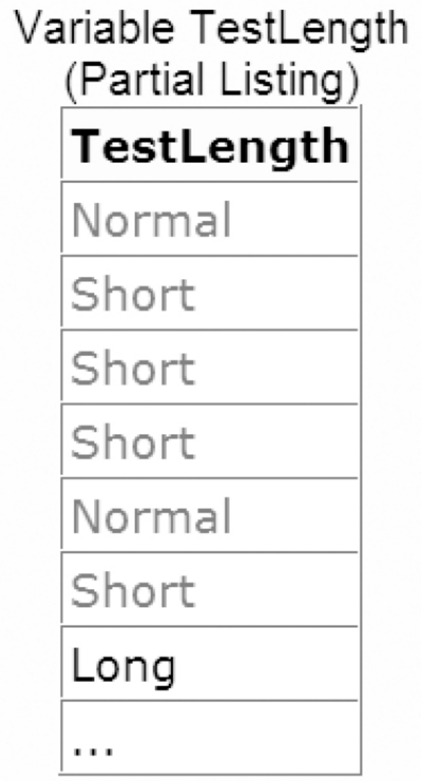
Variable TestLength
(Partial Listing)

| TestLength |
|---|
| Norm |
| Shor |
| Shor |
| Shor |
| Norm |
| Shor |
| Long |
| ... |

# Specifying Lengths for Variables

You can use a **LENGTH statement** to specify a length for TestLength before the first value is referenced elsewhere in the DATA step.

```
filename tests “G:\STSCI5010_fall_2017\fall_2017\Module
    1\Base_SAS\stress.dat”;
data clinic.stress;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32
        RecHR 34-36 TimeMin 38-39 TimeSec 41-42
        Tolerance $ 44;
    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;
```



Variable TestLength
(Partial Listing)

| TestLength |
|---|
| Normal |
| Short |
| Short |
| Short |
| Normal |
| Short |
| Long |
| ... |

# More About the Lengths Statement

- You may specify the lengths of many variables in the same LENGTH statement.

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

- If the variable has been created by another statement, then a later use of the LENGTH statement will not change its length.

```
…
length TestLength $ 3;
…
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';
…
```

| TestLength |
|---|
| Nor |
| Sho |
| Sho |
| Sho |
| Sho |
| Sho |
| Lon |
| … |

(Partial)

# Subsetting Data

## Deleting Unwanted Observations (a review):

You can use the DELETE and IF-THEN statements to conditionally stop processing of the current observation.

**IF** *expression* **THEN DELETE;**

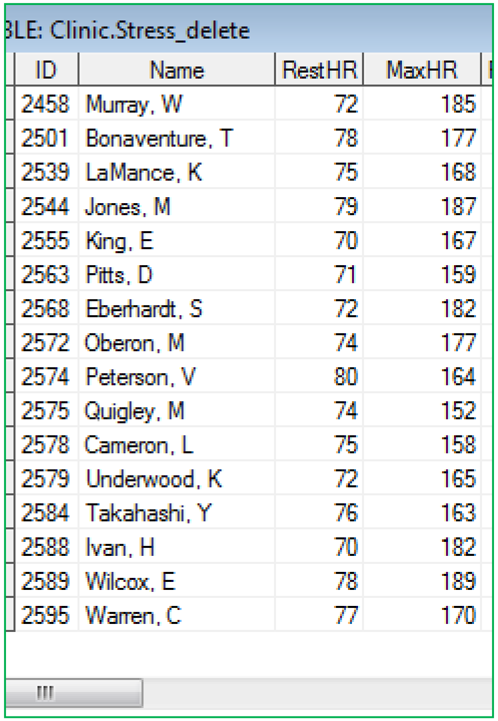
If the *expression* is

- ***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.

# Subsetting Data

## Example: Deleting Unwanted Observations

```
filename tests “G:\STSCI5010_fall_2017\fall_2017\Module
    1\Base_SAS\stress.dat”;
data clinic.stress_delete;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32
        RecHR 34-36 TimeMin 38-39 TimeSec 41-42 Tolerance
        $ 44;
    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;
```



3LE: Clinic.Stress_delete

| ID | Name | RestHR | MaxHR |
|---|---|---|---|
| 2458 | Murray, W | 72 | 185 |
| 2501 | Bonaventure, T | 78 | 177 |
| 2539 | LaMance, K | 75 | 168 |
| 2544 | Jones, M | 79 | 187 |
| 2555 | King, E | 70 | 167 |
| 2563 | Pitts, D | 71 | 159 |
| 2568 | Eberhardt, S | 72 | 182 |
| 2572 | Oberon, M | 74 | 177 |
| 2574 | Peterson, V | 80 | 164 |
| 2575 | Quigley, M | 74 | 152 |
| 2578 | Cameron, L | 75 | 158 |
| 2579 | Underwood, K | 72 | 165 |
| 2584 | Takahashi, Y | 76 | 163 |
| 2588 | Ivan, H | 70 | 182 |
| 2589 | Wilcox, E | 78 | 189 |
| 2595 | Warren, C | 77 | 170 |

# Subsetting Data

## Selecting Variables: Using the Drop= or KEEP= *Data Set Option*

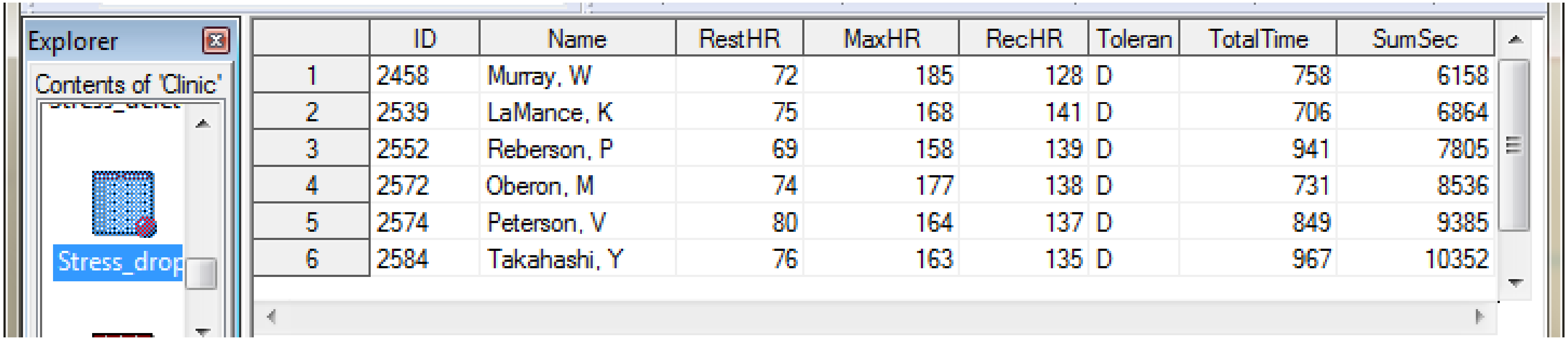
- You can use the DROP= or KEEP= data set options to specify the variables that you want to drop or keep.
- Use the KEEP= option instead of the DROP= option if more variables are dropped than kept.
- General form: (**DROP=***variable(s)*)

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

# Subsetting Data

## Example: Using the Drop= or KEEP= Data Set Option

```
filename tests “G:\STSCI5010_fall_2017\fall_2017\Module 1\Base_SAS\stress.dat”;
data clinic.stress (drop=timemin timesec);
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32 RecHR 34-36 TimeMin
       38-39 TimeSec 41-42 Tolerance $ 44;
    if tolerance=‘D’;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
run;
```



| | ID | Name | RestHR | MaxHR | RecHR | Toleran | TotalTime | SumSec |
|---|---|---|---|---|---|---|---|---|
| 1 | 2458 | Murray, W | 72 | 185 | 128 | D | 758 | 6158 |
| 2 | 2539 | LaMance, K | 75 | 168 | 141 | D | 706 | 6864 |
| 3 | 2552 | Reberson, P | 69 | 158 | 139 | D | 941 | 7805 |
| 4 | 2572 | Oberon, M | 74 | 177 | 138 | D | 731 | 8536 |
| 5 | 2574 | Peterson, V | 80 | 164 | 137 | D | 849 | 9385 |
| 6 | 2584 | Takahashi, Y | 76 | 163 | 135 | D | 967 | 10352 |

# Subsetting Data

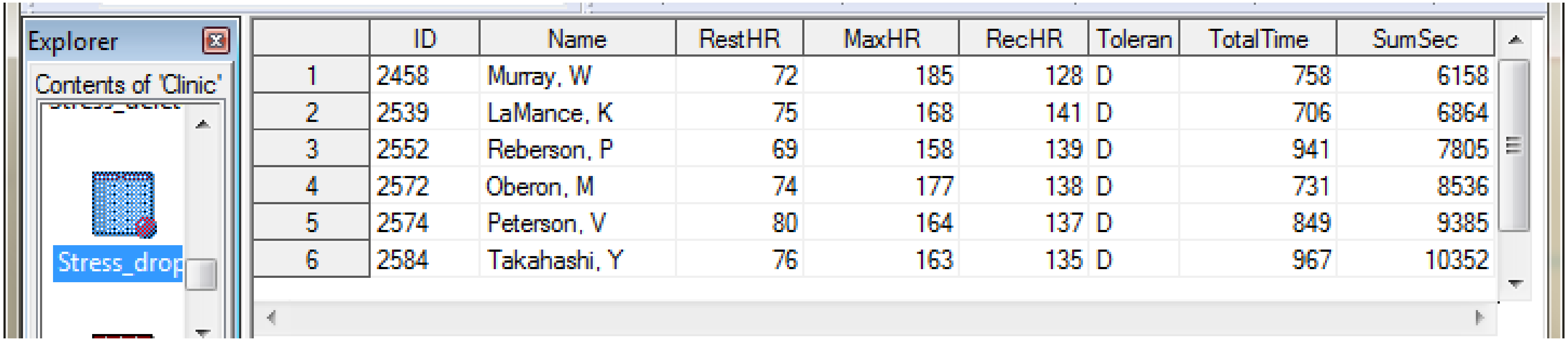
## Selecting Variables: Using the Drop or KEEP ***Statement***

- Another way to exclude variables from your data set is to use the DROP statement or the KEEP statement.
- The DROP and KEEP statements apply to **all output datasets** that are named in the DATA statement.
- You cannot use the DROP and KEEP statements in SAS procedure steps.
- Use the KEEP statement instead of the DROP statement if more variables are dropped than kept.
- General form: **DROP** *variable(s);*
  **KEEP** *variable(s);*

# Subsetting Data

## Example: Using the Drop or KEEP Statement

```
filename tests “G:\STSCI5010_fall_2016\fall_2016\Module 1\Base_SAS\stress.dat”;
data clinic.stress_drop_stmt;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32 RecHR 34-36 TimeMin
        38-39 TimeSec 41-42 Tolerance $ 44;
    if tolerance=‘D’;
    drop timemin timesec;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
run;
```



| | ID | Name | RestHR | MaxHR | RecHR | Toleran | TotalTime | SumSec |
|---|---|---|---|---|---|---|---|---|
| 1 | 2458 | Murray, W | 72 | 185 | 128 | D | 758 | 6158 |
| 2 | 2539 | LaMance, K | 75 | 168 | 141 | D | 706 | 6864 |
| 3 | 2552 | Reberson, P | 69 | 158 | 139 | D | 941 | 7805 |
| 4 | 2572 | Oberon, M | 74 | 177 | 138 | D | 731 | 8536 |
| 5 | 2574 | Peterson, V | 80 | 164 | 137 | D | 849 | 9385 |
| 6 | 2584 | Takahashi, Y | 76 | 163 | 135 | D | 967 | 10352 |

# Assigning Permanent Labels and Formats

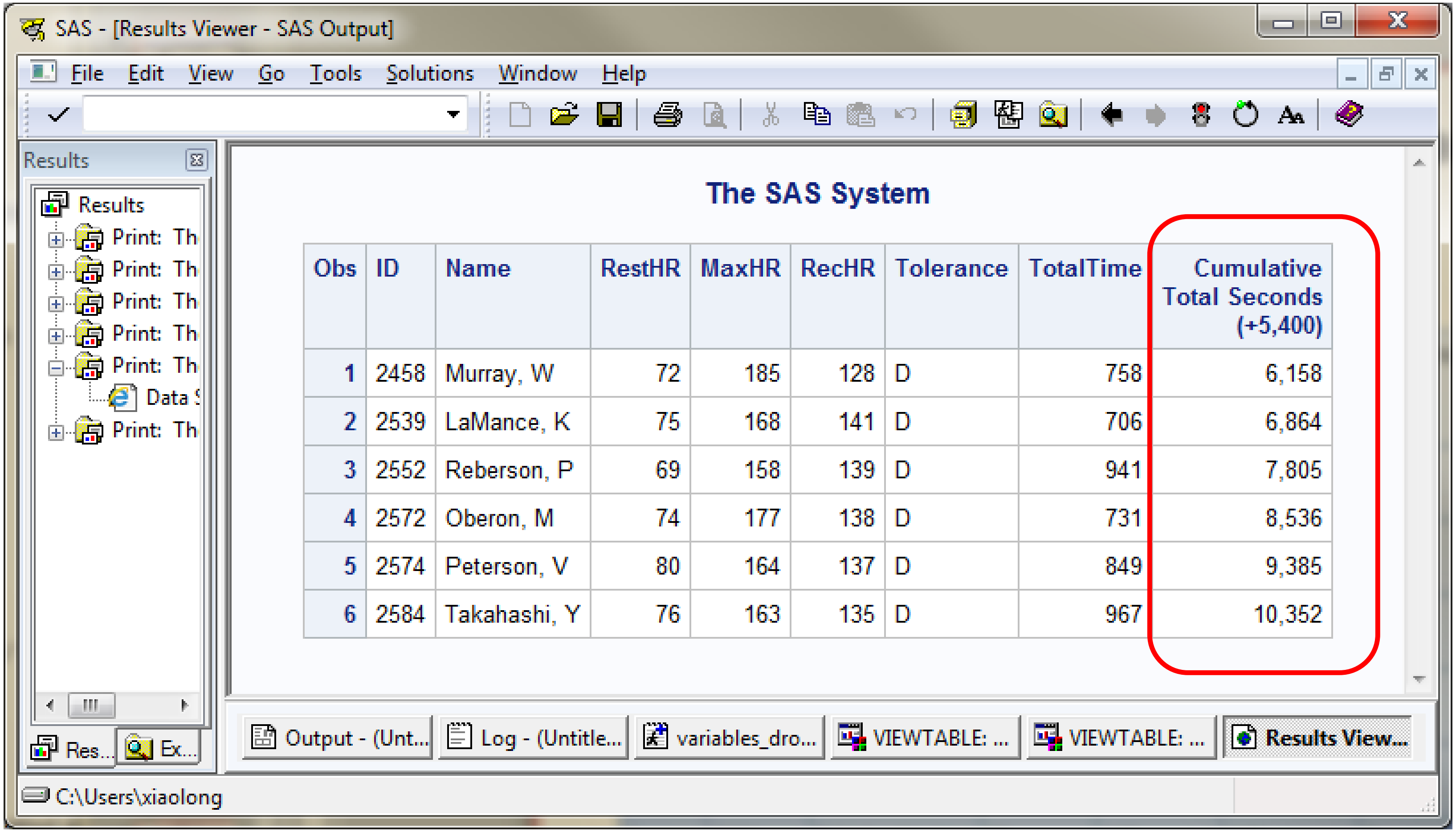
In "Creating List Reports" you learned to temporarily assign labels and formats within a PROC step. To permanently assign labels and formats, you use **LABEL** and **FORMAT** statements in DATA steps.

```
filename tests “G:\STSCI5010_fall_2016\fall_2016\Module 1\Base_SAS\stress.dat”;
data clinic.stress_drop_LF;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32 RecHR 34-36 TimeMin
        38-39 TimeSec 41-42 Tolerance $ 44;
    if tolerance=‘D’;
    drop timemin timesec;
    TotalTime=(timemin*60)+timesec;
    retain SumSec 5400;
    sumsec+totaltime;
    label sumsec='Cumulative Total Seconds (+5,400)';
    format sumsec comma6.;
run;
proc print data=clinic.stress_drop_LF label;
run;
```

**Review:** 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 Permanent Labels and Formats

## The Result



The SAS System

| Obs | ID | Name | RestHR | MaxHR | RecHR | Tolerance | TotalTime | Cumulative Total Seconds (+5,400) |
|---|---|---|---|---|---|---|---|---|
| 1 | 2458 | Murray, W | 72 | 185 | 128 | D | 758 | 6,158 |
| 2 | 2539 | LaMance, K | 75 | 168 | 141 | D | 706 | 6,864 |
| 3 | 2552 | Reberson, P | 69 | 158 | 139 | D | 941 | 7,805 |
| 4 | 2572 | Oberon, M | 74 | 177 | 138 | D | 731 | 8,536 |
| 5 | 2574 | Peterson, V | 80 | 164 | 137 | D | 849 | 9,385 |
| 6 | 2584 | Takahashi, Y | 76 | 163 | 135 | D | 967 | 10,352 |

# Assigning Values Conditionally Using SELECT Groups

You can use a SELECT group to perform conditional processing the same as the IF-THEN/ELSE statements. The general form is

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

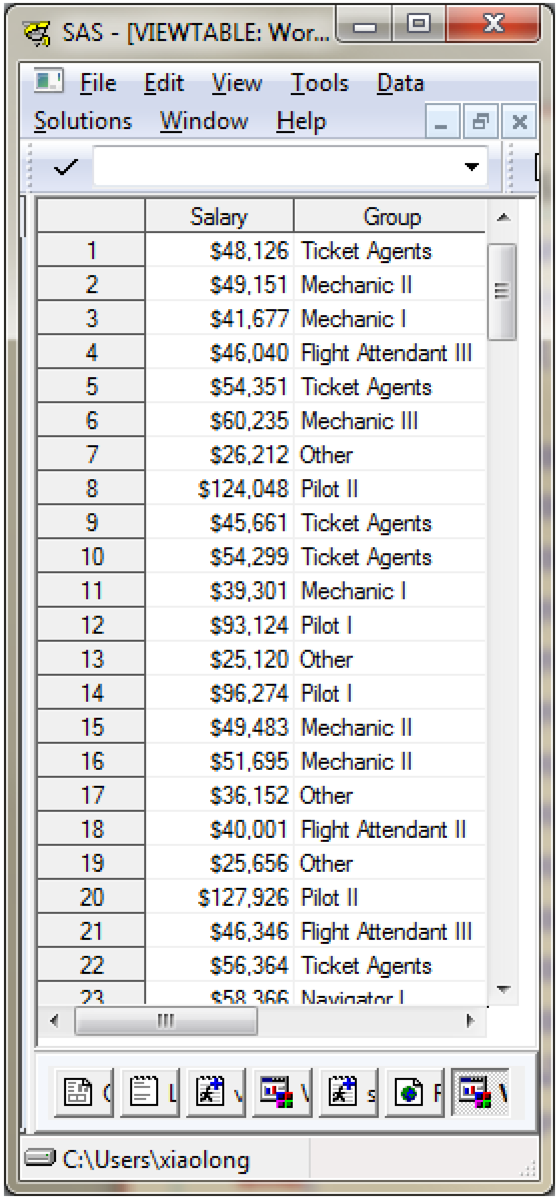
# Examples: Using SELECT Groups

## Basic SELECT Group

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

# Examples: Using SELECT Groups in a DATA Step

```
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 ("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;
```



| | Salary | Group |
|---|---|---|
| 1 | $48,126 | Ticket Agents |
| 2 | $49,151 | Mechanic II |
| 3 | $41,677 | Mechanic I |
| 4 | $46,040 | Flight Attendant III |
| 5 | $54,351 | Ticket Agents |
| 6 | $60,235 | Mechanic III |
| 7 | $26,212 | Other |
| 8 | $124,048 | Pilot II |
| 9 | $45,661 | Ticket Agents |
| 10 | $54,299 | Ticket Agents |
| 11 | $39,301 | Mechanic I |
| 12 | $93,124 | Pilot I |
| 13 | $25,120 | Other |
| 14 | $96,274 | Pilot I |
| 15 | $49,483 | Mechanic II |
| 16 | $51,695 | Mechanic II |
| 17 | $36,152 | Other |
| 18 | $40,001 | Flight Attendant II |
| 19 | $25,656 | Other |
| 20 | $127,926 | Pilot II |
| 21 | $46,346 | Flight Attendant III |
| 22 | $56,364 | Ticket Agents |
| 23 | $58,366 | Navigator I |

# Specifying SELECT Statements with and w/o an Expression

## With an Expression:

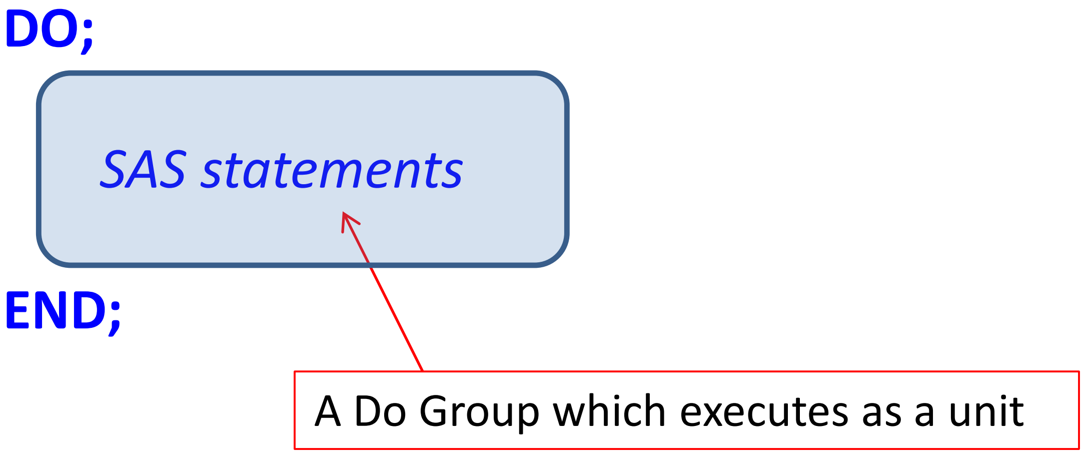
```
select (toy);
      when ("Bear") price=35.00;
      when ("Violin") price=139.00;
      when ("Top","Whistle","Duck") price=7.99;
      otherwise put "Check unknown toy:" toy=;
end;
```

## Without an Expression:

```
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;
end;
```

# Grouping Statements Using DO Groups

- The conditional processing of the IF-THEN/ELSE statements and SELECT groups executes only a **single SAS statement** when a condition is true.
- 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. The general form is



**DO;**

*SAS statements*

**END;**

A Do Group which executes as a unit

# Example 1: Grouping Statements Using DO Groups

```
filename tests “F:\STSCI 5010 Fall 2016\Data\tests.txt”;
data clinic.stress2014_do;
    infile tests;
    input ID $ 1-4 Name $ 6-25 RestHR 27-28 MaxHR 30-32 RecHR
        34-36 TimeMin 38-39 TimeSec 41-42 Tolerance $ 44;
    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;
```

# Example2: Grouping Statements Using DO Groups

```
filename pd "G:\STSCI5010\Lectures\Module1\Ch10\paydata";
data paydata;
    infile pd;
    input payclass $ 1-7 salary 9-13 hrs 15-16 hrlywage 18-19;
       select(payclass);
           when ('monthly') amt=salary;
           when ('hourly')
               do;
                   amt=hrlywage*min(hrs,40);
                   if hrs>40 then put 'CHECK TIMECARD' hrs= _N_=;
               end;
           otherwise put 'PROBLEMATIC 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. It is good practice to indent the statements in DO groups.

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

# Nested DO Groups

```
filename pd "C:\Teaching\STSCI5010\Lectures\Ch10\paydata";
data paydata;
  infile pd;
  input payclass $ 1-7 salary 9-13 hrs 15-16 hrlywage 18-19;
    select(payclass);
       when ('monthly') amt=salary;
       when ('hourly')
          do;
            amt=hrlywage*min(hrs,40);
            if hrs>40 then
               do;
                   put 'CHECK TIMECARD' hrs= _N_=;
                   msg = 'over worked';
               end;
          end;
       otherwise put 'PROBLEMATIC OBSERVATION ' payclass= _N_=;
    end;
run;
```