5. TRANSFORMING SAS DATA SETS

a. Creating new SAS data sets
b. Creating and transforming variables
c. Subsetting observations
d. Subsetting variables

Reading Assignment: Selected SAS Documentation for Bios111
Part 3: Transforming SAS Data Sets
Creating New SAS Data Sets

It will often be desirable to modify an existing SAS data set in some way--selecting only a subset of the original observations, transforming variables, creating new variables, etc. These kinds of modifications are accomplished within a **DATA step**.

- **A DATA Step**
  - Reads one or more input data sets (SAS and/or non-SAS)
  - Performs processing (transformations, selections, etc.), if specified
  - Creates one or more output data sets (SAS or non-SAS)

- In this chapter we will only discuss reading a single input SAS data set and creating a single output SAS data set. The other possibilities will be covered in subsequent chapters.

- All of the modification statements we will discuss can be used with any combination of input and output sources.

**Structure of A DATA Step**

A DATA step that creates a single output SAS data set by modifying a single input SAS data set has a five part structure:

1. A **DATA** statement to start the step and name the output data set
2. A **SET** statement to read an observation from the input data set
3. Programming statements to perform the processing required for this observation
4. An **OUTPUT** statement to write the observation to the output data set
5. A **RETURN** statement to end processing of this observation and return to the top of the step
The DATA Statement

The DATA statement has two functions:

- It defines the start of a DATA step
- It names the SAS data sets to be created

Syntax:

```
DATA Libref.Dataset;
```

Where

- `Dataset` is the name of the SAS data set to be created
- `Libref` is the libref for a SAS data library in which the data set will be stored

The SET Statement

- The SET statement reads an observation from an input SAS data set each time it is executed.
- All variables in the input SAS data set are automatically passed to the new SAS data set (unless otherwise directed with programming statements).
- All observations in the input SAS data set are automatically passed to the new SAS data set (unless otherwise directed with programming statements).
- New variables may be added with assignment statements.
- Note that reading a data set does not modify it in any way.

Syntax:

```
SET Libref.Dataset;
```

Where

- `Dataset` is the name of an existing SAS data set to be read
- `Libref` is the libref for a SAS data library in which the data set is
The OUTPUT Statement

- The OUTPUT statement controls when the values in the program data vector (PDV) are written to the output SAS data.
- The OUTPUT statement is optional.
- When the OUTPUT statement appears in the data step, there is no automatic output at the end of a data step.
- When the OUTPUT statement does not appear in the data step, SAS outputs the values of the PDV at the end of the data step.
- When an OUTPUT statement is executed, SAS immediately outputs the current PDV values to a SAS dataset.
- Execution of the OUTPUT statement does not return control to the beginning of the DATA step.

Syntax:

```plaintext
OUTPUT;  or
OUTPUT  SASdataset(s) ;
```

The RETURN Statement

- The RETURN statement is usually the last statement in the DATA step. It indicates that processing of the current observation is finished. SAS then returns to the DATA statement at the beginning of the step and processes the next observation.
- The RETURN statement is optional. If the RETURN statement is omitted, execution returns to the top of the data step when a RUN or a PROC statement is encountered.

Syntax:

```plaintext
RETURN;
```
**Processing of a DATA Step**

The processing of every DATA step involves two distinct phases.

- First, SAS compiles the statements within the step, creating a program to perform the processing requested
- Second, the program created is executed, processing the data and creating the new data set

- An Example DATA Step:

```sas
DATA WORK.MYCLASS;
SET CLASSLIB класс;
OUTPUT;
RETURN;
RUN;
```
The Compilation Phase

During the compilation phase, the DATA compiler:

- The SET statement reads the descriptor portion of the existing SAS data set
- Creates the descriptor part of the output data set
- Creates the program data vector which will contain all of the variables found in the existing SAS data set plus any new variables created with assignment statements
- Creates a machine language program to perform the processing
- Detects syntax errors

The Execution Phase

During the execution phase:

- The SET statement is executed once for each observation in the existing SAS data set
- Each time the SET statement is executed, it reads an observation from the existing SAS data set and writes the current observation to the PDV
- Any program statements in the DATA step are executed once for each observation in the input data set
- The values in the PDV are written to the new SAS data set after the last executable statement in the DATA step or when an OUTPUT statement is executed
Flowchart of Execution:

1. Initialize PDV to Missing
2. Data Work.MYCLASS;
3. END of Input
4. Yes
5. No
6. Read Next Observation into PVD
7. Set CLASSLIB.CLASS;
8. Modify Data Values in PDV
9. OUTPUT;
10. Write Values From PDV to Output Data Set
11. RETURN;
12. First Output Data Set and Go to Next Step
### SAS data set
**CLASSLIB.CLASS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>NUM</td>
<td>NUM</td>
<td>NUM</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
</tr>
<tr>
<td>HOSKING J</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
</tr>
<tr>
<td>HELMS R</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
</tr>
<tr>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>GONZO</td>
<td></td>
<td>14</td>
<td>25</td>
<td>45</td>
</tr>
</tbody>
</table>

### Program data vector

<table>
<thead>
<tr>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
</table>

### SAS data set
**WORK.MYCLASS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>NUM</td>
<td>NUM</td>
<td>NUM</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Summary--Creating New SAS Data Sets

The four statements just described (DATA, SET, OUTPUT, RETURN) are used whenever we want to create a new SAS data set from an existing one. Other statements are added to the step in order to make the output data set a modified version of the input data set, rather than an exact copy.

In this chapter, we only discuss creating SAS data sets from other, already existing SAS data sets. Creating a SAS data set from a non-SAS data set (e.g., ascii or Dbase file) is a more complex task, which will be covered in detail later in the course.

Creating a new data set does not delete or modify the input data set; it is still available for use in subsequent steps.
Creating and Transforming Variables

In many cases, the reason for creating a new SAS data set will be to create new variables that are some combination of existing variables, or to transform an existing variable.

For example, we might want to add a new variable to the class data set called RELWT (for relative weight) whose value for each observation is defined by the algebraic formula:

\[
\text{RELWT} = \text{WT}/\text{HT} ;
\]

that is, the person’s weight divided by their height.

An example of transforming an existing variable would be recoding the values of height from English units (inches) to metric units (centimeters). The formula in this case is:

\[
\text{HT} = 2.54 \times \text{ht} ;
\]

that is, take each person’s current value of weight, multiply it by 2.54 and use that result to replace the original value.

These kinds of operations are performed in a DATA step using assignment statements.

The Assignment Statement

- The assignment statement is used to create new variables or to transform existing variables.

  Syntax:

  \[\text{variable} = \text{expression};\]

  where

  \text{variable} is the name of a variable in (or to be added to) the data set

  \text{expression} is an arithmetic expression, as defined below

- Examples:

  \[
  \text{RELWT} = \text{WT}/\text{HT} ;
  \]

  \[
  \text{HT} = 2.54 \times \text{ht} ;
  \]
Notes:

- The assignment is one (of two) exceptions to the rule that every SAS statement begins with a keyword.
- If "variable" is the name of an already existing variable, the value of "expression" replaces the previous value; if "variable" is a new name, the assignment statement creates a new variable, which is added to the output data set.

Expressions

- An expression consists of one or more constants, variables, and functions, combined by operators.
- A constant is a number (e.g., 1, -23.6,.00/) or a character string (e.g., ‘JOHN’, ‘MALE’, ‘X#!’); character constants must be enclosed in single quotes (apostrophes). (SAS also allows other, specialized types of constants; we will discuss some of them later in the course.)
- A function is a program "built in" to SAS that performs some computation on character or numeric values.
- An operator is a mathematical, logical, or character operation or manipulation that combines, compares, or transforms numeric or character values.

Arithmetic Operators perform basic arithmetic calculations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
</tbody>
</table>
Comparison operators look at the relationship between two quantities

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Mnemonic Equivalent</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>EQ</td>
<td>equal to</td>
</tr>
<tr>
<td>^=</td>
<td>NE</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>GT</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>LT</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>GE</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>LE</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>equal to one of a list</td>
</tr>
</tbody>
</table>

Logical or Boolean operators are used to link sequences of comparisons.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Mnemonic Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>NOT</td>
</tr>
</tbody>
</table>

Other operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;</td>
<td>Maximum</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples:

Assigning constants:

\[ \begin{align*}
N &= 4; \quad \text{numeric constant} \\
\text{SEX} &= \text{'FEMALE'}; \quad \text{character constant}
\end{align*} \]

Basic arithmetic operators:

\[ \begin{align*}
X2 &= X; \quad \text{copy the value} \\
\text{SUM} &= X + Y; \quad \text{addition} \\
\text{DIF} &= X - Y; \quad \text{subtraction} \\
\text{TWICE} &= X \times 2; \quad \text{multiplication} \\
\text{HALF} &= X / 2; \quad \text{division} \\
\text{CUBIC} &= X^{**3}; \quad \text{exponentiation} \\
Y &= -X; \quad \text{change sign}
\end{align*} \]

Comparison operators:

\[ \begin{align*}
X < Y \text{ THEN } C &= 5; \\
(X < Y); \\
\text{NAME} &= \text{'PAT'}; \\
\text{IF } 5 &\leq \text{AGE} \leq 20; \\
\text{IF } \text{AGE IN}(10,20,30) \text{ THEN } X &= 5; \\
\text{IF } \text{SEX IN}('M', 'F') \text{ THEN } S &= 1;
\end{align*} \]

Logical operators:

\[ \begin{align*}
\text{IF } A < B \text{ AND } C &= 0; \\
\text{IF } X &= 2 \text{ OR } X = 4; \\
\text{IF NOT}(X = 2); \\
\text{IF NOT} (\text{NAME} = \text{'SMITH'});
\end{align*} \]

Other operators:

\[ \begin{align*}
X &= A > < B; \\
\text{NAME} &= \text{FIRST} || \text{LAST};
\end{align*} \]
Complex expressions

Priority of evaluation: ( ) ** | * / | + - |

A=X+Y+Z; left to right
A=X+Y*Z; operator precedence
A=X/Y/Z; left to right
A=X/(Y/Z); parenthetical
Z=ABS(SQRT(X)-2); parenthetical

Examples of Assignment Statement

1 DATA WORK.NEWCLASS;
2   SET CLASSLIB.CLASS;
3       AGE=AGE*12;
4       QUETELET=((WT/2.2)/((HT*2.54)**1))*10000;
5       OUTPUT;
6       RETURN;
7       RUN;

NOTE: Missing values were generated as a result of performing an operation on missing values.
Each place is given by: (Number of times) at (Line) : (Column).
1 at 16:6
1 at 17:13
2 at 17:23

NOTE: The data set WORK.NEWCLASS has 6 observations and 6 variables.
NOTE: The DATA statement used 1.00 seconds.

PROC PRINT DATA=WORK.NEWCLASS;
TITLE1 ‘CREATING VARIABLES WITH ASSIGNMENT STATEMENTS’;
RUN;
NOTE: The PROCEDURE PRINT used 1.00 seconds.

CREATING VARIABLES WITH ASSIGNMENT STATEMENTS

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
<th>QUETELET</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>444</td>
<td>71</td>
<td>195</td>
<td>27.2538</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>M</td>
<td>372</td>
<td>70</td>
<td>160</td>
<td>23.0056</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>M</td>
<td>492</td>
<td>74</td>
<td>195</td>
<td>25.0889</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>36</td>
<td>12</td>
<td>1</td>
<td>4.8927</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td></td>
<td>168</td>
<td>25</td>
<td>45</td>
<td>50.7274</td>
</tr>
</tbody>
</table>
Functions:

- General form of a SAS function:

  \[ \text{variable} = \text{function-name}(\text{argument1}, \text{argument2}, \ldots); \]

- Each argument is separated from the others by a comma. Must functions accept arguments that are constants, variables, expressions, or functions.

- Examples:

  
  \[
  \begin{align*}
  S &= \text{SQRT}(X); \\
  A &= \text{ABS}(X); \\
  B &= \text{MAX}(2,7); \\
  C &= \text{SUBST}(\text{``INSIDE''},3,4); \\
  D &= \text{MIN}(X,7,A+B);
  \end{align*}
  \]

- Types of functions

  - Arithmetic (absolute value, square root, mean, variance…..)
  - Trigonometric (cosine, sine, arc cosine…..)
  - Other mathematical and statistical (natural logarithm, exponential…..)
  - Pseudo-random number generators
  - Character string functions

- Selected functions that compute simple statistics

  - Sum       sum
  - Mean      mean
  - Var       variance
  - Min       minimum
  - Max       maximum
  - Std       standard deviation
  - N         number non-missing
  - Nmiss     number missing

- Simple statistics functions compute statistics for each observation (row) in the SAS data set (functions operate across rows)

- Procedures produce statistics for variables (columns) in the SAS data set (procedures operate down columns)
Subsetting Observations

A common type of transformation is subsetting observations, creating a new SAS data set with the same variables as the input data set, but only those observations that satisfy some selection criterion. The subsetting IF statement can be used to accomplish this.

Syntax:

IF logical expression;

where logical expression is given by one of the following:

- GT
- LT
1. expression • GE • expression
- LE
- EQ
- NE

2. logical expression1 • OR • logical expression2
   • AND •

where "expression" can be any of the forms discussed for assignment statements.

- If the expression is true, execution of the step continues for this observation.
- If the expression is false:
  - SAS stops executing statements for this observation immediately, and
  - returns to the top of the data step and begins processing the next observation.

Examples:

IF AGE GT 35;
IF DEPT EQ ‘FURS’;

Complex logical expressions can be constructed by combining simple logical expression with the operators AND and/or OR.

Examples:

IF (HT GT 70) AND (WT GT 180);
IF (DEPT EQ ‘FURS’) OR (CLERK EQ ‘ABLE’);
Execution of a Data Step With Subsetting IFs

DATA WORK.MALES;
SET CLASSLIB.CLASS;
IF SEX EQ ‘M’;
OUTPUT;
RETURN;
RUN;

CLASSLIB.CLASS

<table>
<thead>
<tr>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIGGY</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>FROG</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>GONZO</td>
<td></td>
<td>14</td>
<td>25</td>
<td>45</td>
</tr>
</tbody>
</table>

NAME   | SEX | AGE | HT  | WT |
-------|-----|-----|-----|----|
FROG   | M   | 3   | 12  | 1  |

WORK.MALES
Examples of Subsetting Observations

```
1 2 DATA WORK.FURS;
3   SET CLASSLIB.SALES;
4   IF DEPT EQ 'FURS';
5   OUTPUT;
6   RETURN
7   RUN;

NOTE: The data set WORK.FURS has 6 observations and 6 variables.
NOTE: The DATA statement used 2.00 seconds.

8 9  PROC PRINT DATA=WORK.FURS;
10 TITLE1 'SELECTING OBSERVATIONS USING SUBSETTING IF';
11  RUN;

NOTE: The PROCEDURE PRINT used 1.00 seconds.
```

SELECTING OBSERVATIONS USING SUBSETTING IF

<table>
<thead>
<tr>
<th>OBS</th>
<th>DEPT</th>
<th>CLERK</th>
<th>PRICE</th>
<th>COST</th>
<th>WEEKDAY</th>
<th>DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FURS</td>
<td>BURLEY</td>
<td>599.95</td>
<td>180.01</td>
<td>THR</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>FURS</td>
<td>BURLEY</td>
<td>800.00</td>
<td>240.00</td>
<td>MON</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>FURS</td>
<td>AGILE</td>
<td>590.00</td>
<td>182.00</td>
<td>SAT</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>FURS</td>
<td>BURLEY</td>
<td>499.95</td>
<td>200.01</td>
<td>SAT</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>FURS</td>
<td>BURLEY</td>
<td>700.00</td>
<td>210.00</td>
<td>THR</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>FURS</td>
<td>AGILE</td>
<td>700.00</td>
<td>210.00</td>
<td>WED</td>
<td>25</td>
</tr>
</tbody>
</table>
Comparison Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>&gt;</td>
</tr>
<tr>
<td>LT</td>
<td>&lt;</td>
</tr>
<tr>
<td>GE</td>
<td>&gt;=</td>
</tr>
<tr>
<td>LE</td>
<td>&lt;=</td>
</tr>
<tr>
<td>EQ</td>
<td>=</td>
</tr>
<tr>
<td>NE</td>
<td>^=</td>
</tr>
<tr>
<td>IN</td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLES:

if age > 35 ;
if age gt 35 ;

if age < 35 ;
if age lt 35 ;

if age >= 35 ;
if age ge 35 ;

if sex > name ;

if age <= 35 ;
if age le 35 ;

if age=35 ;
if age= 35 ;
if age eq 35 ;

if sex='female' ;
if sex='FEMALE' ;
if sex=''' ;
if sex=''' ;

if age ne 35 ;
if age ^= 35 ;

if ht < wt ;
if ht <=.z. ;
if sex=''' ;

IF sex in('MALE','FEMALE') ;
IF age in(30,34) ;
if 30 <= age <= 40 ;
if .z< age <= 50 ;
if 20< age < 50 ;

LOGICAL BOOLEAN OPERATORS AND EXPRESSIONS

<table>
<thead>
<tr>
<th>&amp;</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>NOT</td>
</tr>
</tbody>
</table>

EXAMPLES:

IF AGE=35 AND HT=40 ;
IF (AGE=35) & (HT=40) ;

IF SEX EQ 'FEMALES' AND AGE IN(30,35) ;

IF AGE>=16 AND AGE<=65 ;
IF 16<= AGE <=65 ;

IF HT>WT OR AGE=40 ;
IF (HT>WT) | (AGE=40) ;

IF AGE=20 OR AGE=30 OR AGE=40 ;
IF AGE IN(20,30,40) ;

IF NOT(SEX='MALE') ;
IF SEX NE 'MALES' ;
**BOOLEAN NUMERIC EXPRESSIONS**

In SAS a numeric value other than 0 or missing is true; a value of 0 or missing is false. Therefore a numeric variable or expression can stand alone as a condition. If its value is a number other than 0 or missing it is true; if its value is 0 or missing, the condition is false.

```sas
IF AGE;
IF (HT > WT);
IF (AGE) & (HT > WT);

NEWVAR=(HT>WT);
NEWVAR=(AGE=40);
```
Example

19 PROC PRINT DATA=CLASSLIB.CLASS ;
20 TITLE 'PRINT OUT CLASS DATA SET' ;
21 RUN;
NOTE: The PROCEDURE PRINT used 1.00 seconds.

22 DATA CLASS2 ;
23 SET CLASSLIB.CLASS ;
24 IF AGE ;
25 OUTPUT ;
26 RETURN ;
27 RUN  ;
NOTE: The data set WORK.CLASS2 has 5 observations and 5 variables.
NOTE: The DATA statement used 2.00 seconds.

29 PROC PRINT DATA=ONE ;
30 TITLE 'PRINT OUT CLASS2 DATA SET' ;
31 RUN;
NOTE: The PROCEDURE PRINT used 1.00 seconds.

PRINT OUT CLASS DATA SET

OBS | NAME           | SEX | AGE | HT | WT
--- | ---------------|-----|-----|----|----
 1  | CHRISTIANSEN    | M   | 37  | 71 | 195
 2  | HOSKING J       | M   | 31  | 70 | 160
 3  | HELMS R         | M   | 41  | 74 | 195
 4  | PIGGY M         | F   | .   | 48 | .
 5  | FROG K          | M   | 3   | 12 | 1
 6  | GONZO           |     | 14  | 25 | 45

PRINT OUT CLASS2 DATA SET

OBS | NAME | SEX | AGE | HT | WT
--- |------|-----|-----|----|----
 1  | CHRISTIANSEN | M   | 37  | 71 | 195
 2  | HOSKING J    | M   | 31  | 70 | 160
 3  | HELMS R      | M   | 41  | 74 | 195
 4  | FROG K       | M   | 3   | 12 | 1
 5  | GONZO        |     | 14  | 25 | 45
THE CONCATENATION OPERATOR

- The concatenation operator (||) concatenates character values.
- The results of a concatenation are usually stored in a variable using an assignment statement.
- The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation.
- The concatenation operator does not trim trailing or leading blanks.
- The TRIM function can be used to trim trailing blanks from values before concatenating them.
- Use the LEFT function to trim leading blanks.
CONCATENATION EXAMPLE

64   data one ;
65   set classlib.class ;
66   c1 = 'dept' ;
67   c2 = 'bios' ;
68   c3 = c1 || c2 ;
70   length c4 $ 8 ;
72   c4 = 'dept' ;
73   c5 = c4 || c2 ;
75   c6 = c1 || ' of ' || c2 ;
77   keep c1-c6 ;
78   run;

NOTE: The data set WORK.ONE has 6 observations and 6 variables.

NOTE: The DATA statement used 2.00 seconds.

79   title 'concatenation example' ;
81   proc print ;
82   run;

NOTE: The PROCEDURE PRINT used 1.00 seconds.

83   proc contents ;
84   run;

NOTE: The PROCEDURE CONTENTS used 1.00 seconds.

---

CONCATENATION EXAMPLE

<table>
<thead>
<tr>
<th>OBS</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
<tr>
<td>2</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
<tr>
<td>3</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
<tr>
<td>4</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
<tr>
<td>5</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
<tr>
<td>6</td>
<td>dept</td>
<td>bios</td>
<td>deptbios</td>
<td>dept</td>
<td>dept</td>
<td>bios dept of bios</td>
</tr>
</tbody>
</table>

CONTENTS PROCEDURE

Data Set Name: WORK.ONE  Type:               
Observations:    6  Record Len: 52   
Variables:      6

-----Alphabetic List of Variables and Attributes-----

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C1</td>
<td>Char</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C2</td>
<td>Char</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C3</td>
<td>Char</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C4</td>
<td>Char</td>
<td>8</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C5</td>
<td>Char</td>
<td>12</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>C6</td>
<td>Char</td>
<td>12</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
WHERE STATEMENT

- The WHERE statement allows you to select observations from an existing SAS data set that meet a particular condition before the SAS system brings observations into a data set.

- WHERE selection is the first operation the SAS system performs in each execution of a set, merge, or update operation.

- The WHERE statement is not executable; that is, it can’t be used as part of an IF/THEN statement.

- The WHERE statement is not a replacement for the IF statement; the two work differently and can produce different output data sets. A data step can use either statement, both, or neither.

SYNTAX:

```
WHERE where_expression
```

in which

* `where_expression` is an arithmetic or logical expression.

EXAMPLES:

```
where age>50;
where sex='FEMALE' and ht=.;
```

WHERE EXPRESSIONS

- A WHERE expression is a sequence of operands and operators. You cannot use variables created within the data step or variables created in assignment statements.

- A WHERE expression can use the following operators:

<table>
<thead>
<tr>
<th>Arithmetic Operators</th>
<th>Logical Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>&amp;</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>^</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Comparison Operators

```
=  
^=  
>  
<  
>=  
<=  
IN
```
WHERE vs IF

- The WHERE statement works before observation are brought into the data step (that is, the PROGRAM DATA VECTOR).
- The IF statement works on observation that are already in the data step.
- The WHERE statement is not executable, but the IF statement is.
- The WHERE statement operates only on observations in SAS data sets, whereas the IF statement can operate either on observations from existing SAS data sets or on observations created with an input statement.
- If a BY statement does not accompany a SET or MERGE statement, the WHERE and IF statements usually produce the same result.
- In almost all cases a WHERE statement is more efficient than an IF statement (observations do not have to be moved into the PDV).
- The WHERE statement, but not the IF statement can be used in SAS PROCS.

EXAMPLES:

```sas
DATA ONE;
  SET TWO;
    WHERE AGE > 35 ;
RUN;

DATA ONE;
  SET TWO;
  WHERE AGE ;
RUN;

PROC PRINT DATA=CLASSLIB.CLASS ;
  WHERE SEX = 'FEMALE' ;
RUN;

PROC MEANS DATA=CLASSLIB.CLASS ;
  WHERE 25 < AGE <= 35 AND SEX = 'MALE' ;
RUN;
```
USING A WHERE STATEMENT

```sas
27 proc print data=classlib.class ;
28 title1 'print classlib.sales' ;
29 title2 'no WHERE statement' ;
30 run;
NOTE: The PROCEDURE PRINT used 1.00 seconds.
31
32 proc print data=classlib.class ;
33 WHERE SEX='F' ;
34 title1 'print classlib.sales' ;
35 title2 'using a WHERE statement' ;
36 run;
NOTE: The PROCEDURE PRINT used 1.00 seconds.
37
38 proc print data=classlib.class ;
39 WHERE AGE < 20 OR SEX=' ' ;
40 title1 'print classlib.sales' ;
41 title2 'using a WHERE statement' ;
42 run;
NOTE: The PROCEDURE PRINT used 1.00 seconds.
```

---

print classlib.sales
no WHERE statement

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td>14</td>
<td>25</td>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>

print classlib.sales
using a WHERE statement

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
</tbody>
</table>

print classlib.sales
using a WHERE statement

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td>14</td>
<td>25</td>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>
Subsetting Variables

Another type of transformation that can be performed with a data step is to create a data set containing a subset of the variables from the input data set. The DROP or KEEP statement can be used to accomplish this.

- Syntax:
  ```
  DROP variable list;
  or
  KEEP variable list;
  ```

- Notes:
  
  - Only one of the statements can be used in a step:
  
  "if the DROP statement is used, the variables listed are not included in the output data set if the KEEP statement is used, the variables listed are the only ones included in the output data set"

  - The KEEP or DROP statement only defines which values are written from the program data vector to the output data set; all values are available during the execution of the step

Examples of Subsetting Variables

1
2  DATA WORK.NAMELESS;
3  SET CLASSLIB.CLASS;
4  DROP NAME;
5  OUTPUT;
6  RETURN;
7  RUN;

NOTE: The data set WORK.NAMELESS has 6 observations and 4 variables.

8  PROC PRINT DATA=WORK.NAMELESS;
9  TITLE1 'SUBSETTING VARIABLES WITH THE DROP STATEMENT';
10  RUN;
11  NOTE: The PROCEDURE PRINT used 1.00 seconds.

<table>
<thead>
<tr>
<th>OBS</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>.</td>
<td>14</td>
<td>25</td>
<td>45</td>
</tr>
</tbody>
</table>
Examples of Subsetting Variables

```sas
DATA WORK.NAMEONLY;
SET CLASSLIB.CLASS;
OUTPUT;
KEEP NAME;
RETURN;
RUN;
```

NOTE: The data set WORK.NAMEONLY has 6 observations and 1 variable.
NOTE: The DATA statement used 2.00 seconds.

```sas
PROC PRINT DATA=WORK.NAMEONLY;
TITLE1 'SUBSETTING VARIABLES WITH THE KEEP STATEMENT';
RUN
```

NOTE: The PROCEDURE PRINT used 0.00 seconds.

SUBSETTING VARIABLES WITH THE KEEP STATEMENT

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
</tr>
</tbody>
</table>
**The LENGTH statement:**

The LENGTH statement is used to control the number of bytes allocated for a variable. It may also be used to establish whether the variable is character or numeric. When used as described below, the LENGTH statement can be used with numeric variables which take on only integer values, thus decreasing the disk space required to store SAS data sets.

- **SYNTAX:**

  
  ```
  LENGTH var-list ($) N ...;
  ```

  where

  - `var-list` is a list of variables of the same type and length
  - `$` (if present) denotes that the variables in the preceding var-list are character, not numeric
  - `N` is the number of bytes (length) to be assigned to the variables in the preceding var-list.

    - For character variables, `1 <= N <= 200 (0-32,767 starting with version 7.0)`
    - For numeric variables,
      - on the ACS mainframe (MVS), `2 <= N <= 8`.
      - on the PC (DOS), `3 <= N <= 8`.

- **Notes for character variables:**

  - The length of a character variable is determined by the first statement in which the compiler sees the variable. When used, the LENGTH statement should precede any assignment of SET statement involving the variable in question.

  - When character variables of different lengths are compared, the shorter value is padded with blanks on the right to match the length of the longer variable (in memory only).

- **Notes for numeric variables:**

  - The valid length of a numeric variable is 2-8 bytes on the mainframe and 3-8 bytes on the PC.

  - The default length for numeric variables is 8 bytes; you should specify shorter lengths ONLY FOR INTEGERS, being sure to take into account the maximum integer that can be stored in a given number of bytes as specified in the length tables on the next page. Nonintegers stored in less than 8 bytes will lose precision because they will be truncated.

  - In the PDV, all numbers are stored in 8 bytes.
## LENGTH TABLE FOR MVS AND PC ENVIRONMENTS

(Largest Integer by Length for SAS Numeric Variables under MVS and PC)

<table>
<thead>
<tr>
<th>Length in Bytes</th>
<th>Largest Integer Represented Exactly MVS</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>256</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>65,536</td>
<td>8,192</td>
</tr>
<tr>
<td>4</td>
<td>16,777,316</td>
<td>2,097,152</td>
</tr>
<tr>
<td>5</td>
<td>4,294,967,296</td>
<td>536,870,912</td>
</tr>
<tr>
<td>6</td>
<td>1,099,511,627,776</td>
<td>137,438,953,472</td>
</tr>
<tr>
<td>7</td>
<td>281,474,946,710,565</td>
<td>35,184,372,088,832</td>
</tr>
<tr>
<td>8</td>
<td>72,057,594,037,927,936</td>
<td>9,007,199,254,740,992</td>
</tr>
</tbody>
</table>
LENGTH STATEMENT

USAGE NOTES:

- LENGTH’s placement in the data step determines its effectiveness. If placed before the first reference to a variable, it will store it in the indicated number of bytes. If it is placed after the step’s first reference to a variable, it will have no effect, nor will SAS produce an error message.

- There is no correspondence between the number of columns used for a numeric variable and the number of bytes specified in the length statement.

- For numeric variables lengths of less than eight should only be used for integers.

- It is usually a good idea to specify lengths for all calculated or assigned character variables.

EXAMPLES:

Length a b c 3 d 4 c $ 8 ;
Length v1-v5 7 c1-c5 $ 8 ;

Data one ;
  set two ;
  length size $ 6 ;
  if ht<10 then size = 'small' ;
  if ht>=10 then size='medium' ;
run;

Data new;
  length v1 5 region $ 3 ;
  set old;
run;
ASSIGNING CONSTANTS: EXAMPLE

24    data one ;
25      set classlib.class ;
26
27      ** create c1 as a character constant ** ;
28      c1 = 'CSCC' ;
29
30      ** create c2 as a character constant ** ;
31      c2 = 'cscc' ;
32
33      ** create c2/c3 as a character constant ** ;
34      c3=c2 ;
35      c3='cscc' ;
36
37      ** create n1 as a numeric constant ** ;
38      n1=100 ;
39      n2=100.00 ;
40      n3= 100 ;
41      n4=1e2 ;
42      RUN;

NOTE: The data set WORK.ONE has 6 observations and 12 variables.
NOTE: The DATA statement used 2.00 seconds.

43
44      TITLE 'ASSIGNING CONSTANTS' ;
45      PROC PRINT ;
46      RUN;

NOTE: The PROCEDURE PRINT used 1.00 seconds.
47      PROC CONTENTS ;
48      RUN;

NOTE: The PROCEDURE CONTENTS used 2.00 seconds.
## ASSIGNING CONSTANTS

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td></td>
<td>14</td>
<td>25</td>
<td>45</td>
<td>CSCC</td>
<td>cscc</td>
<td>csc</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

## CONTENTS PROCEDURE

Data Set Name: WORK.ONE
Observations: 6
Variables: 12

-----Alphabetic List of Variables and Attributes-----

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>AGE</td>
<td>Num</td>
<td>8</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>C1</td>
<td>Char</td>
<td>4</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>C2</td>
<td>Char</td>
<td>4</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>C3</td>
<td>Char</td>
<td>4</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HT</td>
<td>Num</td>
<td>8</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>N1</td>
<td>Num</td>
<td>8</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>N2</td>
<td>Num</td>
<td>8</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>N3</td>
<td>Num</td>
<td>8</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>N4</td>
<td>Num</td>
<td>8</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NAME</td>
<td>Char</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SEX</td>
<td>Char</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WT</td>
<td>Num</td>
<td>8</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>
SAS Date, Time, and Date-Time Values

- Although dates and times are typically written in a numeric form (12/25/83, 12:15), they are not truly numeric in that we can not directly perform arithmetic operations on them.

- SAS allows date and time data to be stored as numeric variables.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>UNITS</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Days</td>
<td>Days since January 1, 1960</td>
</tr>
<tr>
<td>Time</td>
<td>Seconds</td>
<td>Number of seconds and hundredths of seconds</td>
</tr>
<tr>
<td>Date-time</td>
<td>Seconds</td>
<td>Seconds since midnight, January 1, 1960</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jan 1, 1953</th>
<th>Jan 1, 1960</th>
<th>Nov 24, 1983</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2556</td>
<td>0</td>
<td>8728</td>
</tr>
</tbody>
</table>

SAS Date Values

Notes:

- The baseline of January 1, 1960 is arbitrary

- Any dates from 1582 to 20,000AD are valid

- SAS accounts for leap years, century and fourth-century adjustments

- Although date and date-time values have implied baseline times, differences in these values are directly interpretable. For example, the number of days from January 1, 1953 to November 24, 1983 is:

$$8728 - (-2556) = 11284 \text{ days}$$
**Date Constants and Functions**

- You can use **SAS date constants** to generate a SAS date from a specific date entered in your SAS code.
- Date constants are special constants that SAS converts into date values.
- The syntax of the date constant is:

  `'ddmmmyyyy’d`

  where

  dd        is a one or two digit value for day  
  mmm      is a three letter abbreviation for month(JAN,FEB…)
  yyyy    is a two or four digit value for year(4 is recommended)

  The d at the end of the constant ensures that SAS does not confuse the string with a character constant.

- Examples:

  Date1 = ‘07OCT1999’d;
  If evdate <= ‘21JUL1987’d;
  If ‘01JUL1990’d <= bdate <= ‘30JUL1990’d;

- There are several useful functions available for handling SAS dates

  **YEAR(SAS-date)** extracts the year from a SAS date and returns a 4-digit year value.

  **MONTH(SAS-date)** extracts the month from a SAS date and returns a number between 1 and 12.

  **DAY(SAS-date)** extracts the day from a SAS date and returns a number between 1 and 31.

  **TODAY()** extracts the date from the computer system’s clock and stores the value as a SAS date. This function does not require any arguments.

  **MDY(month,day,year)** creates a SAS date from separate month, day, and year variables. Arguments can be SAS numeric variables or constants. A missing or out of range value creates a missing value.
Simple Calculations

Using SAS date variables you can easily find the time elapsed between two dates. Simply subtract the dates to find the number of elapsed days, then, if necessary, divide the number to scale it to months, years, weeks, or any other unit of interest.

<table>
<thead>
<tr>
<th>Days</th>
<th>date2 – date1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Months</td>
<td>(date2 – date1)/30.4</td>
</tr>
<tr>
<td>Years</td>
<td>(date2 – date1)/365.25</td>
</tr>
</tbody>
</table>
ASSIGNING DATE CONSTANTS: EXAMPLE

102 data one;
103 set classlib.class;
104
105 ** create date1 as a date constant **;
106 date1 = '01jul1993'd;
107
108 ** create date2 as a date constant **;
109 date2 = mdy(07,01,1993);
110
111 ** create date3 as a date constant **;
112 date3 = '01jul1943'd;
113
114 KEEP NAME DATE1-DATE3;
115 run;

NOTE: The data set WORK.ONE has 6 observations and 4 variables.
NOTE: The DATA statement used 2.00 seconds.

116 title 'ASSIGNING DATE CONSTANTS';
117 proc print;
118 run;

NOTE: The PROCEDURE PRINT used 1.00 seconds.

120 proc contents;
121 run;

NOTE: The PROCEDURE CONTENTS used 1.00 seconds.

ASSIGNING DATE CONSTANTS

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>DATE1</th>
<th>DATE2</th>
<th>DATE3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td>12235</td>
<td>12235</td>
<td>-6028</td>
</tr>
</tbody>
</table>

CONTENTS PROCEDURE

Data Set Name: WORK.ONE
Observations: 6
Variables: 4

----Alphabetic List of Variables and Attributes----

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NAME</td>
<td>Char</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DATE1</td>
<td>Num</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DATE2</td>
<td>Num</td>
<td>8</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DATE3</td>
<td>Num</td>
<td>8</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

5-38
**ARITHMETIC OPERATIONS: EXAMPLE**

192 data one;
193 set classlib.class;
194
195 n1 = min(ht,wt);
196 n2 = min(age,35);
197 n3 = min(ht,sex);
198 n4 = mean(sex,name);
199 RUN;

NOTE: Invalid numeric data, SEX='M', at line 197 column 15.
NOTE: Invalid numeric data, SEX='M', at line 198 column 17.
NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 N1=71 N2=35 N3=71 N4=. _ERROR_=1 _N_=1
NOTE: Invalid numeric data, SEX='M', at line 197 column 15.
NOTE: Invalid numeric data, SEX='M', at line 198 column 17.
NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 N1=71 N2=35 N3=71 N4=. _ERROR_=1 _N_=1

NAME=HOSKING J SEX=M AGE=31 HT=70 WT=160 N1=70 N2=31 N3=70 N4=. _ERROR_=1 _N_=2
NAME=HELMS R SEX=M AGE=41 HT=74 WT=195 N1=74 N2=35 N3=74 N4=. _ERROR_=1 _N_=3
NAME=PIGGY M SEX=F AGE=. HT=48 WT=. N1=48 N2=35 N3=48 N4=. _ERROR_=1 _N_=4
NAME=FROG K SEX=M AGE=3 HT=12 WT=1 N1=1 N2=3 N3=12 N4=. _ERROR_=1 _N_=5
NAME=GONZO SEX= AGE=14 HT=25 WT=45 N1=25 N2=14 N3=25 N4=. _ERROR_=1 _N_=6

NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 N1=71 N2=35 N3=71 N4=. _ERROR_=1 _N_=1
NAME=HOSKING J SEX=M AGE=31 HT=70 WT=160 N1=70 N2=31 N3=70 N4=. _ERROR_=1 _N_=2
NAME=HELMS R SEX=M AGE=41 HT=74 WT=195 N1=74 N2=35 N3=74 N4=. _ERROR_=1 _N_=3
NAME=PIGGY M SEX=F AGE=. HT=48 WT=. N1=48 N2=35 N3=48 N4=. _ERROR_=1 _N_=4
NAME=FROG K SEX=M AGE=3 HT=12 WT=1 N1=1 N2=3 N3=12 N4=. _ERROR_=1 _N_=5
NAME=GONZO SEX= AGE=14 HT=25 WT=45 N1=25 N2=14 N3=25 N4=. _ERROR_=1 _N_=6

NOTE: Character values have been converted to numeric values at the places given by: (Number of times) at (Line):(Column).

6 at 197:15
12 at 198:17

NOTE: Missing values were generated as a result of performing an operation on missing values.
   Each place is given by: (Number of times) at (Line):(Column).
6 at 198:17

NOTE: The data set WORK.ONE has 6 observations and 9 variables.
NOTE: The DATA statement used 3.00 seconds.

202 203 TITLE 'ARITHMETIC OPERATIONS';
204 PROC PRINT LABEL;
205 RUN;
NOTE: The PROCEDURE PRINT used 1.00 seconds.

---

**ARITHMETIC OPERATIONS**

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
<td>71</td>
<td>35</td>
<td>71</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
<td>70</td>
<td>31</td>
<td>70</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
<td>74</td>
<td>35</td>
<td>74</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
<td>48</td>
<td>35</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>.</td>
</tr>
</tbody>
</table>
ARITHMETIC OPERATIONS: EXAMPLE

35    data one ;
36      set classlib.class ;
37
38      ** use a character variable with a numeric operand ** ;
39      c1 = sex + 2 ;
40
41      ** numeric var on left , char var on right ** ;
42      length n1 8 ;
43      n1 = sex ;
44
45      ** char var on left , num var on right ** ;
46      length c2 $ 3 ;
47      c2 = AGE ;
48
49    RUN;

NOTE: Invalid numeric data, SEX=' M' , at line 39 column 9.
NOTE: Invalid numeric data, SEX=' M' , at line 43 column 9.
NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 C1=. N1=. C2=37 _ERROR_=1 _N_=1
NOTE: Invalid numeric data, SEX=' M' , at line 39 column 9.
NOTE: Invalid numeric data, SEX=' M' , at line 43 column 9.
NAME=HOSKING J SEX=M AGE=31 HT=70 WT=160 C1=. N1=. C2=31 _ERROR_=1 _N_=2
NOTE: Invalid numeric data, SEX=' M' , at line 39 column 9.
NOTE: Invalid numeric data, SEX=' M' , at line 43 column 9.
NAME=HELMS R SEX=M AGE=41 HT=74 WT=195 C1=. N1=. C2=41 _ERROR_=1 _N_=3
NOTE: Invalid numeric data, SEX=' F' , at line 39 column 9.
NOTE: Invalid numeric data, SEX=' M' , at line 43 column 9.
NAME=PIGGY M SEX=F AGE=. HT=48 WT=. C1=. N1=. C2=3 _ERROR_=1 _N_=4
NOTE: Invalid numeric data, SEX=' M' , at line 39 column 9.
NOTE: Invalid numeric data, SEX=' M' , at line 43 column 9.
NAME=FROG K SEX=M AGE=3 HT=12 WT=1 C1=. N1=. C2=3 _ERROR_=1 _N_=5

NOTE: Character values have been converted to numeric
  values at the places given by: (Number of times) at (Line): (Column).
  6 at 39:9
  6 at 43:9
NOTE: Missing values were generated as a result of performing an operation on
  missing values.
  Each place is given by: (Number of times) at (Line): (Column).
  6 at 39:9
NOTE: Numeric values have been converted to character
  values at the places given by: (Number of times) at (Line): (Column).
  6 at 47:9
NOTE: The data set WORK.ONE has 6 observations and 8 variables.
NOTE: The DATA statement used 3.00 seconds.
50    PROC CONTENTS ;
51    RUN;
NOTE: The PROCEDURE CONTENTS used 1.00 seconds.
52    PROC PRINT ;
53    RUN;
NOTE: The PROCEDURE PRINT used 1.00 seconds.
## CONTENTS PROCEDURE

Data Set Name: WORK.ONE  
Type:  
Observations: 6  
Record Len: 60  
Variables: 8  
Label:  

-----Alphabetic List of Variables and Attributes-----

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>AGE</td>
<td>Num</td>
<td>8</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>C1</td>
<td>Num</td>
<td>8</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>C2</td>
<td>Char</td>
<td>3</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HT</td>
<td>Num</td>
<td>8</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>N1</td>
<td>Num</td>
<td>8</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NAME</td>
<td>Char</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SEX</td>
<td>Char</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WT</td>
<td>Num</td>
<td>8</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBS</th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
<th>C1</th>
<th>N1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHRISTIANSEN</td>
<td>M</td>
<td>37</td>
<td>71</td>
<td>195</td>
<td>.</td>
<td>.</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>HOSKING J</td>
<td>M</td>
<td>31</td>
<td>70</td>
<td>160</td>
<td>.</td>
<td>.</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>HELMS R</td>
<td>M</td>
<td>41</td>
<td>74</td>
<td>195</td>
<td>.</td>
<td>.</td>
<td>41</td>
</tr>
<tr>
<td>4</td>
<td>PIGGY M</td>
<td>F</td>
<td>.</td>
<td>48</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>FROG K</td>
<td>M</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>GONZO</td>
<td>14</td>
<td>25</td>
<td>45</td>
<td>.</td>
<td>.</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>