

6. SAS PROGRAMMING

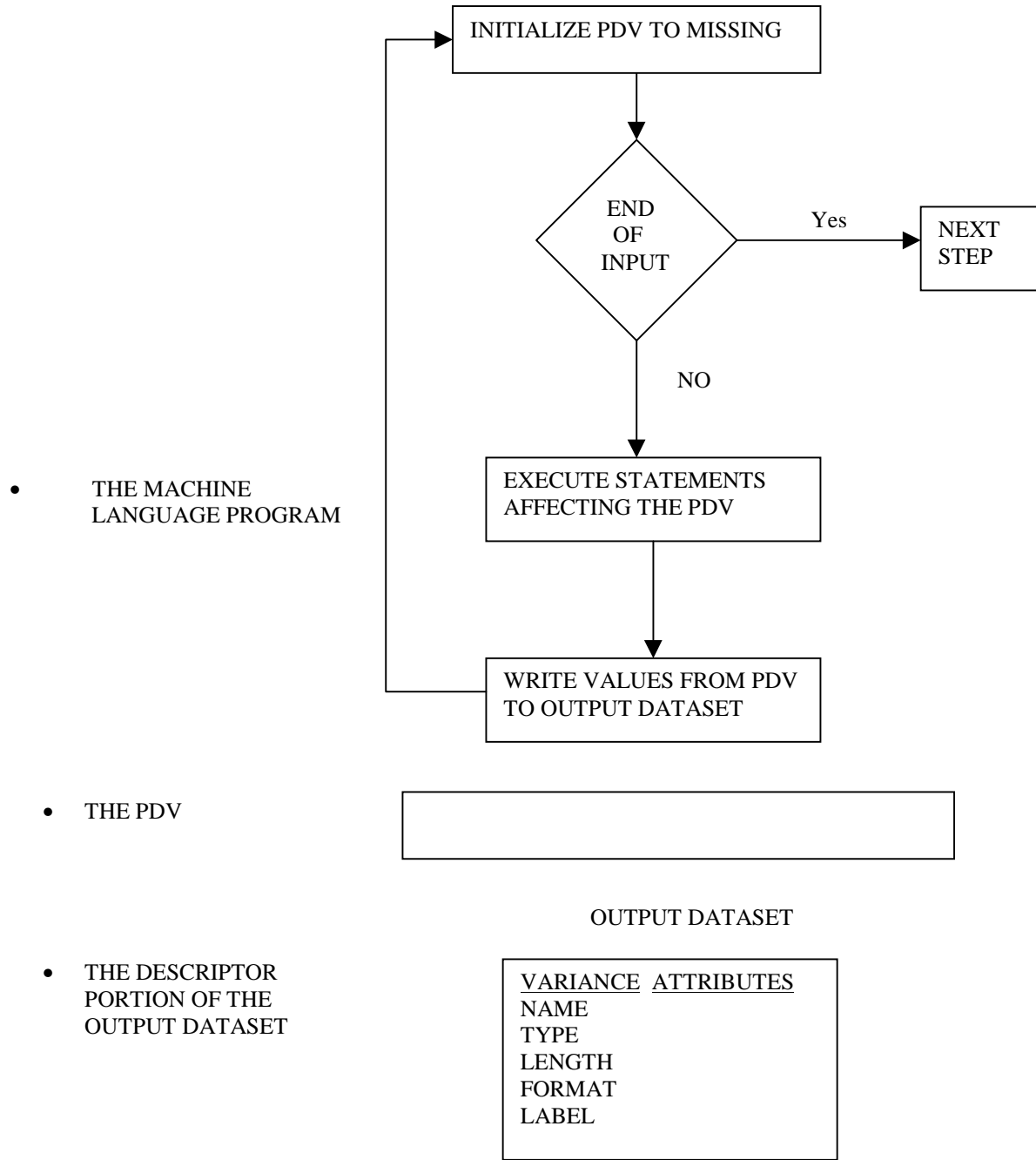
**Reading Assignment: SELECTED SAS DOCUMENTATION FOR BIOS111
Part 4: SAS Programming**

REVISED FALL 2000

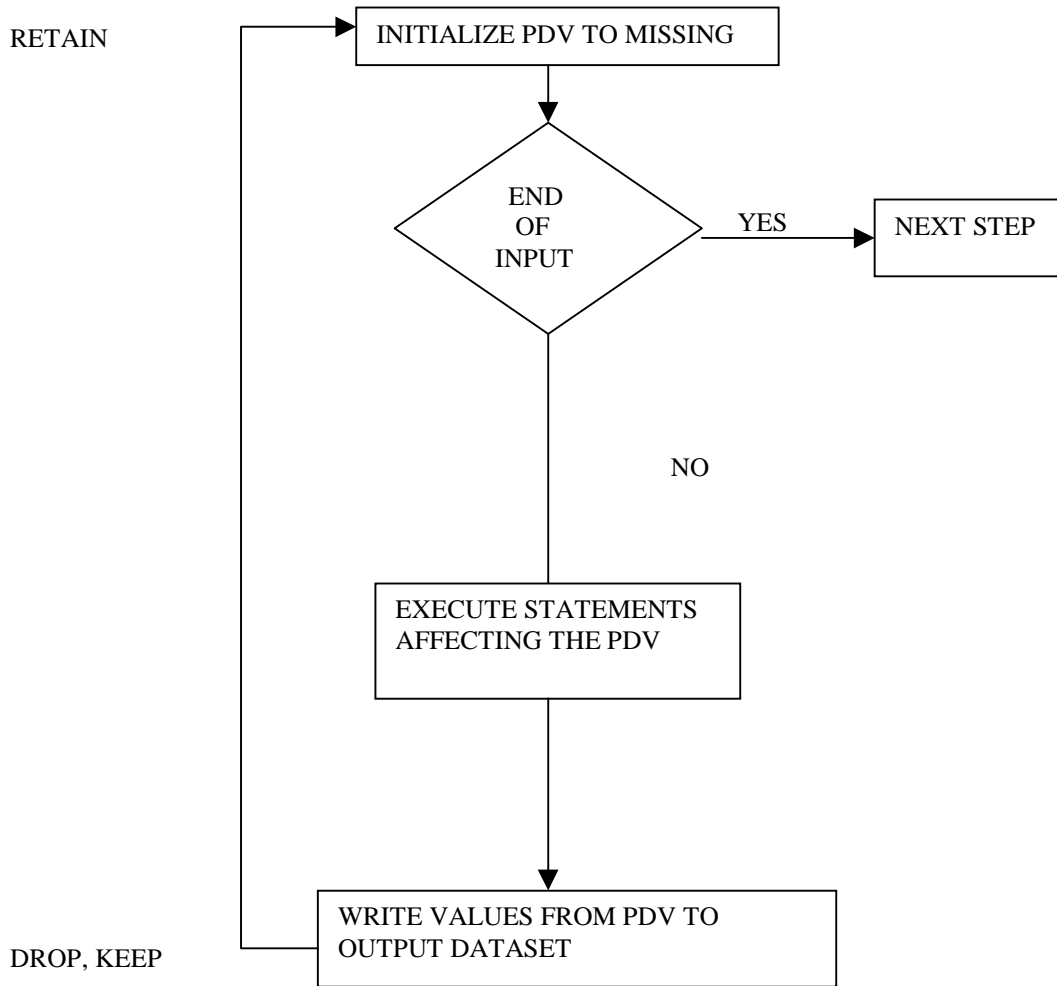
Declarative Statements

- ❖ These statements supply information to SAS during the compilation phase of DATA step processing. They define and modify the actions to be taken during the execution phase and affect the composition and contents of the PDV and the new data set being created.
- ❖ For example, the DROP and KEEP statements determine which of the variables in the PDV get output to the new data set.
- ❖ These statements are "non-executable" and their placement in the DATA step usually is unimportant, although there are cases where their order does affect the outcome. Remember, the PDV is created during the compile phase by compiling the statements in the order in which the compiler comes to them.

THE COMPILATION PHASE OF A DATA STEP CREATES:



THE FOLLOWING DECLARATIVE STATEMENTS CAN AFFECT THE COMPILATION PHASE:



VAR NAME
TYPE
LENGTH

LENGTH

PROGRAM DATA VECTOR

OUTPUT DATASET

VARIABLE ATTRIBUTES
NAME
TYPE
LENGTH
FORMAT
LABEL

THE RETAIN STATEMENT

- ❖ The RETAIN statement lists those variables in the PDV that should not be initialized to missing at the beginning of each execution of the DATA step.

- ❖ SYNTAX

RETAIN var-list [initial value] ...,

where

Var-list is a list of variable names to be exempt from being reset to missing.

Initial value is the value placed in the PDV at compile time.

- ❖ NOTES:

- ❖ Multiple RETAIN statements may be entered in the same data set
- ❖ A single RETAIN statement may specify both numeric and character variables
- ❖ If the first reference to a variable is in the RETAIN statement, SAS assumes it is numeric; to indicate a character variable, provide an initial value of the proper length; a better strategy is to define the variable in a preceding LENGTH statement
- ❖ Only variables created by assignment and INPUT statements may be retained; retaining a variable brought into the DATA step with a SET, MERGE, or UPDATE statement is not an error, just an action with no effect.
- ❖ Values of retained variables are held over from the last observation, but they can be changed with SET or assignment statements.
- ❖ If no initial value is given, character variables are initially blank and numeric variables are initially missing (.).

EXAMPLE: Create Cumulative Totals for Height, Weight, and Age.

```
32 DATA ONE;
33 SET CLASSLIB.CLASS ;
34
35 CUMHT = CUMHT + HT ;
36 CUMWT = CUMWT + WT ;
37 CUMAGE= CUMAGE + AGE ;
38 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).

6 at 35:11

6 at 36:11

6 at 37:11

NOTE: The data set WORK.ONE has 6 observations and 8 variables.

NOTE: The DATA statement used 2.00 seconds.

```
39 PROC PRINT DATA=ONE ;
40 TITLE 'DATA SET ONE - NO RETAIN STATEMENT' ;
41 RUN;
```

NOTE: The PROCEDURE PRINT used 1.00 seconds.

DATA SET ONE - NO RETAIN STATEMENT

OBS	NAME	SEX	AGE	HT	WT	CUMHT	CUMWT	CUMAGE
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	.	.	.

EXAMPLE: Create Cumulative Totals for Height, Weight, and Age.

```
43 DATA ONE;
44 SET CLASSLIB.CLASS ;
45
46 RETAIN CUMHT CUMWT 0 CUMAGE ;
47
48 CUMHT = CUMHT + HT ;
49 CUMWT = CUMWT + WT ;
50 CUMAGE= CUMAGE + AGE ;
51 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).

6 at 50:11

3 at 49:11

NOTE: The data set WORK.ONE has 6 observations and 8 variables.

NOTE: The DATA statement used 2.00 seconds.

```
52 PROC PRINT DATA=ONE ;
53 TITLE 'DATA SET ONE - USING A RETAIN STATEMENT' ;
54 RUN;
```

NOTE: The PROCEDURE PRINT used 0.00 seconds.

DATA SET ONE - USING A RETAIN STATEMENT

OBS	NAME	SEX	AGE	HT	WT	CUMHT	CUMWT	CUMAGE
1	CHRISTIANSEN	M	37	71	195	71	195	.
2	HOSKING J	M	31	70	160	141	355	.
3	HELMS R	M	41	74	195	215	550	.
4	PIGGY M	F	.	48	.	263	.	.
5	FROG K	M	3	12	1	275	.	.
6	GONZO		14	25	45	300	.	.

RETAIN EXAMPLE: CREATING CONSTANTS

```
1 DATA ONE ;  
2   SET CLASSLIB.CLASS ;  
3  
4   RETAIN N1 10 C1 'BIOS';  
5  
6   KEEP NAME N1 C1 ;  
7 RUN;
```

NOTE: The data set WORK.ONE has 6 observations and 3 variables.

NOTE: The DATA statement used 5.00 seconds.

```
8 PROC PRINT DATA=ONE ;  
9   TITLE 'THE RETAIN STATEMENT';  
10 RUN;
```

NOTE: The PROCEDURE PRINT used 2.00 seconds.

THE RETAIN STATEMENT

OBS	NAME	N1	C1
1	CHRISTIANSEN	10	BIOS
2	HOSKING J	10	BIOS
3	HELMS R	10	BIOS
4	PIGGY M	10	BIOS
5	FROG K	10	BIOS
6	GONZO	10	BIOS

RETAIN EXAMPLE: COUNTING

```
11 DATA ONE ;
12   SET CLASSLIB.CLASS ;
13
14   RETAIN COUNT 0 NMALES 0 ;
15
16   COUNT=COUNT + 1 ;
17   NMALES=NMALES + (SEX='M') ;
18
19   KEEP NAME SEX COUNT NMALES ;
20 RUN;
```

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

NOTE: The DATA statement used 4.00 seconds.

```
21
22 PROC PRINT DATA=ONE ;
23   TITLE 'THE RETAIN STATEMENT';
24 RUN;
```

NOTE: The PROCEDURE PRINT used 2.00 seconds.

THE RETAIN STATEMENT

OBS	NAME	SEX	COUNT	NMALES
1	CHRISTIANSEN	M	1	1
2	HOSKING J	M	2	2
3	HELMS R	M	3	3
4	PIGGY M	F	4	3
5	FROG K	M	5	4
6	GONZO		6	4

THE RENAME STATEMENT

❖ The RENAME statement changes the name of a variable between the PDV and the output data set.

❖ SYNTAX

```
rename OLDNAME=NEW NAME...;
```

❖ EXAMPLE:

DATASET A

NAME	SEX	HT
------	-----	----

```
DATA B;  
  SET A;  
  RENAME  
    NAME = LASTNAME  
    HT = HEIGHT;  
  OUTPUT;  
  RETURN;  
RUN;
```

PDV

NAME	SEX	HT
------	-----	----

DATASET B

LASTNAME	SEX	HEIGHT
----------	-----	--------

THE LABEL STATEMENT

- ❖ The LABEL statement is used during the DATA step to add a label of up to 40 characters to the descriptor section of the output data set being created.

- ❖ SYNTAX

LABEL varname = '40 character label'...;

- ❖ NOTES:

- ❖ The label must be enclosed in single or double quotes. When a single quote (apostrophe) is to be included, it is represented in the label statement by two consecutive single quotes (not a double quote). For example, to assign the label "subject's name": to the variable NAME, the LABEL statement would be LABEL NAME = 'SUBJECT'S NAME';
- ❖ Existing labels from the input data set are included on the output data set, unless they are modified by a LABEL statement.

EXAMPLE:

Contents of SAS DATA SET WORK.OLDCLASS

ALPHABETIC LIST OF VARIABLES

#	VARIABLES	TYPE	LENGTH	POSITION	LABEL
3	AGE	NUM	3	17	AGE IN YEARS
4	HT	NUM	8	20	HEIGHT OF SUBJECT
1	NAME	CHAR	12	4	
2	SEX	CHAR	1	16	SEX
5	WT	NUM	8	28	WEIGHT IN POUNDS

```
DATA NEWCLASS;  
SET OLDCLASS;
```

```
    LABEL  
        AGE = '      '  
        HT = 'HEIGHT IN INCHES'  
    ;  
OUTPUT;  
RETURN;
```

```
PROC CONTENTS DATA = NEWCLASS NOSOURCE;
```

CONTENTS OF SAS DATA SET WORK.NEWCLASS

#	VARIABLES	TYPE	LENGTH	POSITION	LABEL
3	AGE	NUM	3	17	
4	HT	NUM	8	20	HEIGHT IN INCHES
1	NAME	CHAR	12	4	
2	SEX	CHAR	1	16	SEX
5	WT	NUM	8	28	WEIGHT IN POUNDS

THE FORMAT STATEMENT

- ❖ The FORMAT statement specifies how the value of a variable is written, printed, or used by a procedure. The FORMAT statement can be used in a DATA step to permanently assign an output format to a variable by storing the format name in the description portion of the data set. Formats can be used in PROC steps to temporarily assign a format to a variable for the execution of the PROC. Formats do not affect how a variable is stored internally.

❖ SYNTAX

FORMAT variable list format...,

where

Variable list is a list of SAS variables of the same type and format is the name of the format to be used in writing the variable.

❖ NOTES

- ❖ formats may be selected from the SAS format library , or may be created using PROC FORMAT.
- ❖ the two major format types correspond to the two SAS data types, numeric and character; names of formats for character variables all begin with a dollar sign(\$)
- ❖ numeric and character refer to the way the variable is stored, not the way it appears on input or output
- ❖ output formats are specified with FORMAT statements; these can either be part of a data step or a procedure
- ❖ if a format is assigned during a data step, it travels with the data set; anytime the data set is used in the current session or in later ones, in PROCs or in other data steps, SAS will know how to display the variable; this is referred to as assigning permanent formats
- ❖ most formats have widths and alignments
- ❖ width refers to the number of columns used to display the variable
- ❖ alignment refers to how SAS behaves when the variable's formatted length is less than the specified width; usually numeric formats right-align and character formats left-align ; this means that if there are leftover columns, numbers are shifted flush right in the columns and characters are shifted flush left in the columns
- ❖ SAS issues an error message if you try to use a character format for a numeric variable or a numeric for a character
- ❖ any number of format statements can be used in a data step
- ❖ placement of the FORMAT statement in the step is usually not important

FREQUENTLY USED SAS SYSTEM FORMATS

FORMAT NAME -----	DESCRIPTION -----	ALIGNMENT -----
w.	numeric, no decimals	r
w.d	numeric, with decimals	r
commaw.d	embedded commas	r
ew.d	scientific notation	r
hexw.d	numeric hexadecimal	l
dollarw.d	dollars and commas added	r
zw.d	insert leading zeros	r
\$w.	character, leading blanks trimmed	l
\$charw.	character, leading blanks preserved	l
bestw.	SAS selects appropriate decimal places	r
date7.	numeric, used for dates (01APR93)	r
mmddyw.	numeric, used for dates (04/01/93)	r

EXAMPLE: FORMATS IN A DATA STEP

```
DATA CLASS2;
  SET CLASSLIB.CLASS;
  FORMAT SEX $HEX2.AGE ROMAN6. HT WORDS15. WT E8.;
  OUTPUT,
  RETURN,

PROC PRINT DATA=CLASS2;
  TITLE1      'Proc Print of Data Set Class2';
  TITLE2      'With Formats in the descriptor section of the Data Set';

PROC CONTENTS DATA=CLASS2;
```

Proc Print of Data Set Class2
With Formats in the descriptor section of the Data Set

OBS	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	4D	XXXVII	seventy-one	2.0E+02
2	HOSKING J	4D	XXXI	seventy	1.6E+02
3	HELMS R	4D	XLI	seventy-four	2.0E+02
4	PIGGY M	46	.	forty-eight	.
5	FROG K	4D	III	twelve	1.0E+00
6	GONZO	20	XIV	twenty-five	4.5E+01

CONTENTS PROCEDURE

Data Set Name:	WORK.CLASS2	Observations:	6
Member Type:	DATA	Variables:	5
Engine:	V612	Indexes:	0
Created:	11:03 Tuesday, June 1, 1999	Observation Length:	37
Last Modified:	11:03 Tuesday, June 1, 1999	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

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

#	Variable	Type	Len	Pos	Format
3	AGE	Num	8	13	ROMAN6.
4	HT	Num	8	21	WORDS15.
1	NAME	Char	12	0	
2	SEX	Char	1	12	\$HEX2.
5	WT	Num	8	29	E8.

EXAMPLE: FORMATS IN A PROC STEP

```
libname sc 'c:\bios111\sasdata';
proc print data=sc.sales;
  format cost dollar8.2 price fract10. day z3.1
  id dept ;
title1 'PROC PRINT OF DATA SET SALES';
title2 'WITH A FORMAT STATEMENT IN THE PRINT STEP';
run;
```

PROC PRINT OF DATA SET SALES WITH A FORMAT STATEMENT IN THE PRINT STEP

OBS	DEPT	CLERK	PRICE	COST	WEEKDAY	DAY
1	SHOES	CLEVER	99+19/20	\$41.21	TUE	003
2	SHOES	AGILE	95	\$40.49	WED	004
3	SHOES	CLEVER	65	\$33.44	WED	004
4	SHOES	CLEVER	65	\$33.44	WED	004
5	FURS	BURLEY	599+19/20	\$180.01	THR	005
6	SHOES	AGILE	49+19/20	\$28.07	THR	005
7	SHOES	AGILE	69+19/20	\$34.93	THR	005
8	SHOES	BURLEY	69+19/20	\$34.93	THR	005
9	SHOES	CLEVER	84+19/20	\$38.65	SAT	007
10	SHOES	CLEVER	54+19/20	\$30.00	SAT	007
11	SHOES	CLEVER	95	\$40.49	SAT	007
12	FURS	BURLEY	800	\$240.00	MON	009
13	SHOES	CLEVER	139+19/20	\$42.96	MON	009
14	SHOES	CLEVER	59+19/20	\$31.78	MON	009
15	SHOES	AGILE	54+19/20	\$30.00	TUE	010
16	SHOES	CLEVER	94+19/20	\$40.48	TUE	010
17	SHOES	CLEVER	65	\$33.44	TUE	010
18	SHOES	CLEVER	89+19/20	\$39.63	TUE	010
19	SHOES	BURLEY	75	\$36.31	WED	011
20	SHOES	CLEVER	54+19/20	\$30.00	WED	011

FORMAT EXAMPLE

```
37 DATA ONE ;
38   SET CLASSLIB.CLASS ;
39
40   DATE1 = '30SEP93'D ;
41   DATE2 = DATE1 ;
42
43   FORMAT DATE1 DATE2 DATE7. ;
44   KEEP DATE1 DATE2 HT ;
45 RUN;
```

NOTE: The data set WORK.ONE has 6 observations and 3 variables.

NOTE: The DATA statement used 1.00 seconds.

```
46
47 PROC PRINT DATA=ONE(OBS=1) SPLIT='*' DOUBLE ;
48   VAR DATE1 DATE2 HT ;
49   FORMAT HT 1.0 DATE2 ;
50   LABEL DATE1="TODAY'S*DATE*FORMATTED"
51         DATE2="TODAY'S*DATE*UNFORMATTED"
52   ;
53   TITLE 'FORMAT EXAMPLE' ;
54 RUN;
```

NOTE: The PROCEDURE PRINT used 1.00 seconds.

FORMAT EXAMPLE				
	TODAY'S	TODAY'S		
	DATE	DATE		
OBS	FORMATTED	UNFORMATTED	HT	
1	30SEP93	12326	*	

FORMAT EXAMPLE

```
55 DATA ONE ;
56   SET CLASSLIB.CLASS ;
57
58   DATE1 = '30SEP93'D ;
59   DATE2 = DATE1 ;
60
61   FORMAT DATE1 DATE2 DATE7. ;
62   KEEP DATE1 DATE2 HT ;
63 RUN;
```

NOTE: The data set WORK.ONE has 6 observations and 3 variables.

NOTE: The DATA statement used 5.00 seconds.

```
64
65 PROC PRINT DATA=ONE(OBS=1) SPLIT='*' DOUBLE ;
66   VAR DATE1 DATE2 HT ;
67   FORMAT HT $3. DATE2 ;
```

ERROR: You are trying to use the character format \$ with the numeric variable HT.

```
68 LABEL DATE1="TODAY'S*DATE*FORMATTED"
69   DATE2="TODAY'S*DATE*UNFORMATTED"
70 ;
71 TITLE 'FORMAT EXAMPLE';
72 RUN;
```

NOTE: The SAS System stopped processing this step because of errors.

NOTE: The PROCEDURE PRINT used 3.00 seconds.

THE FORMAT PROCEDURE

- ❖ allows you to define your own output formats
- ❖ can be used to assign value labels for variable values
- ❖ can be used to recode variables in the data step
- ❖ can be used to collapse variable categories in a PROC step
- ❖ User defined formats can be used:
 - ❖ in a FORMAT statement in a DATA or PROC step
 - ❖ in a PUT statement
- ❖ There are 2 types of user written formats:
 - ❖ VALUE formats - convert one or more user-specified values into a single character string
 - ❖ PICTURE formats - specifies a template for how to print a number or range of numbers

❖ SYNTAX

PROC FORMAT ;

 VALUE fmtname

 range1 = 'label1'

 range2 = 'label2';

 where format names:

- ❖ are 8 characters or less
- ❖ begin with a letter or underscore for numeric variables
- ❖ begin with a dollar sign(\$) for character variables
- ❖ do not end with a number
- ❖ are unique

Ranges can be:

- ❖ single value or OTHER

```
VALUE AFT 1='agree'  
          2='disagree'  
          OTHER='ERROR';
```

- ❖ ranges of values including LOW and HIGH

```
VALUE AFT LOW-12='kids'  
          13-19='teens'  
          20-HIGH='adults';
```

- ❖ lists of values and ranges

```
VALUE SFT 1,3='male'  
          2,4='female'  
          0,5-9='miscoded'  
          .= 'missing';
```

- ❖ character values, ranges, or lists

```
VALUE $GR 'A+'='H'  
          'A'-'C'='P'  
          'D'='L';
```

- ❖ LABELS can be up to 40 characters in length and should be enclosed in single quotes

❖ NOTES

- ❖ values not in any ranges are displayed as is, unformatted
- ❖ missing values are not included in the LOW specification
- ❖ missing values can be included in the OTHER specification
- ❖ ranges should not overlap; a value can appear only once in all ranges
- ❖ values in the range can be explicit or implied. Examples of implied ranges include

1-10 (values 1-10)

1<-10 (greater than 1 up through 10)

1-<10 (1 up to but not including 10)

- ❖ user-written formats may be used in data step FORMAT statements. If the data set is stored permanently, you must have the format available whenever the data set is used, since the format is part of the permanent data set.

EXAMPLE: USER DEFINED FORMATS

```

PROC FORMAT ;
  VALUE HFT  LOW - <55.0 = 'SHORT'
           70.1- HIGH = 'TALL';

  VALUE WFT  LOW - 75.0 = 'SKINNY'
           75.0< - 125.0 = 'AVERAGE'
           125.0< - HIGH = 'ROBUST'
           OTHER = 'MISSING';

  VALUE $SFT 'M' = 'MALE'
           'F' = 'FEMALE'
           OTHER = 'INVALID';
RUN;

PROC PRINT DATA=SC.CLASS;
  FORMAT HT HFT. WT WFT. SEX $SFT. ;
RUN;

PROC FREQ DATA=SC.CLASS;
  TABLES HT WT ;
  FORMAT HT HFT. WT WFT. ;
RUN;

```

OBS	NAME	SEX	AGE	HT	WT
1	CHRISTIANSEN	MALE	37	TALL	ROBUST
2	HOSKING J	MALE	31	70	ROBUST
3	HELMS R	MALE	41	TALL	ROBUST
4	PIGGY M	FEMALE	.	SHORT	MISSING
5	FROG K	MALE	3	SHORT	SKINNY
6	GONZO	INVALID	14	SHORT	SKINNY

HT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SHORT	3	50.0	3	50.0
70	1	16.7	4	66.7
TALL	2	33.3	6	100.0

WT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SKINNY	2	40.0	2	40.0
ROBUST	3	60.0	5	100.0

Frequency Missing = 1

CONDITIONAL EXECUTION OF STATEMENTS

Up to this point, the statements in the DATA step have been executed sequentially for each observation processed. The ability to execute or not execute a statement based on whether or not some condition is met is one of the most powerful features of a computer. We have already discussed a very specialized conditional statement, the subsetting IF, which is used to control whether or not observations are added to the output data set. The general form of the IF statement is

**IF expression THEN statement1;
ELSE statement2;**

where *expression* is any valid SAS expression, and statement 1 and 2 are any executable SAS statements.

NOTES:

- ❖ If expression is "true" (non-zero and non-missing) then statement 1 is executed. If expression is "false" (zero or missing) then statement 2 is executed.
- ❖ The expression is usually a comparison expression (x LT 4), in which case the expression has a value of 1 for true and 0 for false.
- ❖ Arithmetic expression (Y+Z) are also valid.
- ❖ The ELSE statement is optional. If it is not used and the expression is false, control is transferred to the next statement.

CONDITIONAL EXECUTION: EXAMPLE

```
/* CREATE AGE, HT and WT Categories*/  
  
DATA CLASS;  
  Set ClassLib.Class;  
  
  If age LT 30 then AGECAT='YOUNG';  
  Else AGECAT='OLD';  
  
  If HT LT 30 then HTCAT='SHORT';  
  Else If 30 LE HT LT 70 then HTCAT='AVE';  
  Else If HT GE 70 then HTCAT='TALL';  
  
  If 2 LT WT LT 100 then WTCAT='LIGHT';  
  Else If WT GE 100 then WTCAT='HEAVY';  
  
OUTPUT;  
RETURN;  
RUN;
```

DATA SET CLASS WITH AGE, HT, WT CATEGORIES

NAME	SEX	AGE	HT	WT	AGECAT	HTCAT	WTCAT
Christiansen	M	37	71	195	OLD	TALL	HEAVY
Hosking J	M	31	70	160	OLD	TALL	HEAVY
Helms R	M	41	74	195	OLD	TALL	HEAVY
Piggy M	F	.	48	.	YOUNG	AVE	
Frog K	M	3	12	1	YOUNG	SHORT	LIGHT
Gonzo		14	25	45	YOUNG	SHORT	LIGHT

CONDITIONAL EXECUTION: EXAMPLE

```
4 data one ;
5   set classlib.class ;
6
7   retain nmales nfemales prob 0 ;
8
9   if sex='M' then nmales=nmales + 1;
10  else if sex='F' then nfemales=nfemales + 1;
11  else prob=prob + 1 ;
12 run;
```

NOTE: The data set WORK.ONE has 6 observations and 8 variables.

NOTE: The DATA statement used 4.00 seconds.

```
13
14 PROC PRINT ;
15  TITLE 'IF/THEN/ELSE RETAIN EXAMPLE';
16 RUN ;
```

NOTE: The PROCEDURE PRINT used 1.00 seconds.

IF/THEN/ELSE RETAIN EXAMPLE

OBS	NAME	SEX	AGE	HT	WT	NMALES	NFEMALES	PROB
1	CHRISTIANSEN	M	37	71	195	1	0	0
2	HOSKING J	M	31	70	160	2	0	0
3	HELMS R	M	41	74	195	3	0	0
4	PIGGY M	F	.	48	.	3	1	0
5	FROG K	M	3	12	1	4	1	0
6	GONZO		14	25	45	4	1	1

CONDITIONAL EXECUTION: EXAMPLE

```
5 DATA ONE ;
6   SET CLASSLIB.CLASS ;

7
8   RETAIN NMALES NFEMALES PROB 0 ;
9
10  IF SEX='M' THEN NMALES=NMALES + 1;
11  ELSE IF SEX='F' THEN NFEMALES=NFEMALES + 1;
12  ELSE PROB=PROB + 1 ;
13
14  IF _N_=6 ;
15
16  RUN;
```

NOTE: The data set WORK.ONE has 1 observations and 8 variables.

NOTE: The DATA statement used 4.05 seconds.

```
17
18 PROC PRINT ;
19   TITLE 'IF/THEN/ELSE RETAIN EXAMPLE';
20 RUN ;
```

NOTE: The PROCEDURE PRINT used 0.58 seconds.

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

IF/THEN/ELSE RETAIN EXAMPLE

2

OBS	NAME	SEX	AGE	HT	WT	NMALES	NFEMALES	PROB
1	GONZO		14	25	45	4	1	1

DO/END STATEMENTS

- ❖ The DO and END statements define the beginning and end of a group of statements called a DO Group. The DO Group can be used within IF-THEN/ELSE statements to conditionally execute groups of statements.
- ❖ Execution of a DO statement specifies that all statements between the DO and its matching END statement are to be executed.
- ❖ SYNTAX:

```
DO;  
    DO GROUP STATEMENTS  
    .  
    .  
    .  
END;
```

❖ EXAMPLE

Consider a data set where some height and weight measurements were collected in English Units (inches and pounds) and some were collected in metric units (meters and kilograms). Convert all measurements to centimeters and grams.

DATA SET WITH MIXED ENGLISH AND METRIC UNITS

NAME	SEX	AGE	HT	WT	UNITS
Christiansen	M	37	1.8034	88.63636	M
Hosking J	M	31	70	160	E
Helms R	M	41	74	195	E
Piggy M	F	.	48	.	E
Frog K	M	3	0.3048	0.4545455	M
Gonzo		14	0.635	20.45455	M

```

DATA FIXED;

  SET MIXED;

  DROP UNITS;

  IF UNITS EQ 'M' THEN DO;                                /*METRIC UNITS */
    HT=HT*100;                                           /*METERS TO CM */
    WT=WT*1000;                                          /*KILOS TO GRAMS */
  END;

  ELSE DO;                                               /*ENGLISH UNITS */
    HT=HT*2.54;                                         /*INCHES TO CM */
    WT=WT*1000/2.2;                                     /*POUNDS TO GRAMS */
  END;

  OUTPUT;

  RETURN;

  RUN;

```

CLASS DATA SET WITH METRIC UNITS

NAME	SEX	AGE	HT	WT
Christiansen	M	37	180.34	88636.36
Hosking J	M	31	177.8	72727.27
Helms R	M	41	187.96	88636.36
Piggy M	F	.	121.92	.
Frog K	M	3	30.48	454.5455
Gonzo		14	63.5	20454.55

ITERATIVE EXECUTION OF DO GROUPS

DO Index-variable=start TO stop BY increment;
DO index-variable=start TO stop;

Iterative DO loops are used to repeatedly execute the statements within a DO group, changing the value of the index-variable each time. The number of iterations and the value of the index variable are determined by the "Start", "Stop" and "increment" parameters.

For example, the following code to add the odd integers from 1 to 7:

```
X=0 ;  
DO I = 1 to 7 by 2;  
    X = X + I;  
END;
```

Is equivalent to:

```
X = 0;  
I = 1; if (I GT 7) then "leave loop"  
X = X + I;  
I = I + 2; if (I GT 7) then "leave loop"  
X = X + I;  
I = I + 2; if (I GT 7) then "leave loop"  
X = X + I;  
I = I + 2; if (I GT 7) then "leave loop"  
X = X + I;  
I = I + 2; if (I GT 7) then "leave loop"
```

INTERPRETATION OF THE CONTROL EXPRESSION

- 1) The first time the DO statement is encountered, the index-variable is set to "start".
- 2) If the index variable is greater than "stop", then control passes to the statement following the END statement.
- 3) If the value of the index variable is less than or equal to "stop", the statements in the DO group are executed.
- 4) At the end of the DO group, "increment" is added to the index-variable and control branches back to the test against "stop" (step 2 above).
- 5) This process is repeated until the index-variable is greater than the "stop" value. Control then passes to the statement following the END statement.

NOTES:

- ❖ The index-variable will be included in the output data set unless it is explicitly dropped.
- ❖ "Start", "Stop", and "increment" can all be arbitrarily complex expressions whose values are only evaluated once, the first time through the loop.

EXAMPLES:

- 1) Compute the final balance resulting from depositing a given amount (CAPITAL) for a given number of years (TERM) at a given rate of interest (RATE). Assume interest is compounded yearly.

WORK.MONEY

<u>CAPITAL</u>	<u>TERM</u>	<u>RATE</u>
1000	3	.10
100	5	.15

```
DATA WORK.COMPOUND;  
  SET WORK.MONEY;  
  DO YEAR=1 TO TERM BY 1;  
    INTEREST=CAPITAL*RATE;  
    CAPITAL=CAPITAL+INTEREST,  
    END;  
  DROP YEAR INTEREST,  
RUN;
```

WORK.COMPOUND

<u>CAPITAL</u>	<u>TERM</u>	<u>RATE</u>
1331.00	3	.10
201.44	5	.15

- 2) Count the number of leap years experienced by each member of the data set WORK.KIDS, shown below;*

WORK.KIDS

<u>NAME</u>	<u>BIRTHYR</u>
JOE	1979
SUE	1981
ED	1975

```
DATA WORK.LEAPS;  
  SLI WORK.KIDS,  
  NLEAP = 0;  
  DO YEAR=BIRTHYR TO 1984 BY 1;  
    IF MOD(YEAR,4) EQ 0 THEN NLEAP=NLEAP+1;  
  END;  
DROP YEAR;  
RUN;
```

WORK.LEAPS

<u>NAME</u>	<u>BIRTHYR</u>	<u>NLEAPS</u>
JOE	1979	2
SUE	1981	1
ED	1975	3

*The solution shown does not deal appropriately with century years (e.g., 1900)

3) Compute the factorial of X, X!, where;

$$X! = 1 * 2 * 3 \dots X$$

By definition, $0! = 1$ and $X!$ is undefined for $X < 0$.

WORK.MATH

X
1
4
-3
0
12

```
DATA WORK.FACTORL;  
  SET WORK.MATH,  
  IF X GE 0,  
  FACTX=1,  
  DO I=1 TO X;  
    FACTX=FACTX*I;  
  END;  
RUN;
```

WORK-FACTORL

<u>X</u>	<u>FACTX</u>	<u>I</u>
1	1	2
4	24	5
0	1	1
12	479,001,600	13

DO-LOOPS: USAGE NOTES

DO STATEMENTS HAVE 5 COMPONENTS:

1. The INDEX is a numeric variable controlling the execution of the loop.
2. The BEGIN is a numeric variable, constant, or expression that defines the beginning value taken by the INDEX.
3. The END is a numeric variable, constant or expression that defines the last value taken by INDEX.
4. The INCREMENT is a numeric variable, constant, or expression that controls how the value of the INDEX changes. Its default value is 1. Generally, once INDEX plus INCREMENT exceeds END, the loop terminates.
5. The VALUE is a numeric or character variable, constant or expression.

NOTES:

- ❖ Loops execute until the increment exceeds the end value. This means that the end value must be reachable from begin. You can not have a begin at 100 and an end at 50 unless you used a negative increment.
- ❖ The index becomes part of the SAS dataset being created unless it is included in a drop statement.
- ❖ The begin, end, increment, and value must be nonmissing.
- ❖ You can combine the various forms of the indexed DO statements, using begin, end, and optionally, increment, with one or more "value" specification.
- ❖ Each form of these DO-loops can be nested within each other or a do group.

MORE COMPLEX DO LOOPS

The general syntax of the iterative DO statement is;

```
DO control expression 1, control expression 2, ...;
```

Where each "control expression" has the general form;

```
Start TO stop BY increment
```

If more than one control expression is included, each is executed in turn. Thus the statements within the loop;

```
DO I=1 to 3 by 1, 10 to 40 by 10;  
END;
```

would be executed 7 times, with I having, successively, the values 1, 2, 3, 10, 20, 30, 40.

The control expressions can also be abbreviated;

- * If "BY increment" is omitted, "BY 1" is assumed
- * If "TO stop BY increment" is omitted, the loop will be executed once with "index variable=start"

Example:

Check the variable Y for the missing value codes 99, 998, and 999 and recode to a SAS missing value:

```
DO MISS=99, 998, 999;  
IF Y=MISS THEN Y=.;  
END;
```

A DO loop can also "count down." In that case "increment" is negative and "stop" must be less than "start." In such cases, the loop is repeated until the value of the index variable is less than "stop".

EXAMPLE 1:

Find the length of the value of a character variable, NAME. The length will be defined as the position of the right-most non-blank character in NAME. Assume that the length of the variable NAME is 20.

```
LENGTH = 0;
DO I=20 to 1 by -1;
  If (SUBSTR(NAME,I,1) NE ' ') and (LENGTH EQ 0) then LENGTH=I;
END;
If (LENGTH=.) then LENGTH=0;
```

EXAMPLE 2: Creating a data set without input data

```
23 DATA ONE ;
24
25 LENGTH VAR1 3 ;
26 DO VAR1 = 1 TO 8 BY 2, 10, 40, 50 ;
27
28 VAR2 = SQRT(VAR1) ;
29
30 OUTPUT ;
31
32 END ;
33
34 RUN;
```

NOTE: The data set WORK.ONE has 7 observations and 2 variables.

NOTE: The DATA statement used 1.34 seconds.

DO LOOPS

OBS	VAR1	VAR2
1	1	1.00000
2	3	1.73205
3	5	2.23607
4	7	2.64575
5	10	3.16228
6	40	6.32456
7	50	7.07107

EXAMPLE 3: Creating a data set without input data

```
42 DATA ONE ;
43
44 LENGTH VAR1 3 VAR2 $ 6 ;
45 DO VAR1 = 1 TO 5 ;
46
47     DO VAR2 = 'FEMALE', 'MALE' ;
48
49     OUTPUT ;
50
51     END ;
52 END ;
53
54 RUN;
```

NOTE: The data set WORK.ONE has 10 observations and 2 variables.
NOTE: The DATA statement used 4.37 seconds.

```
55
56 PROC PRINT ;
57 TITLE 'DO LOOPS' ;
58 RUN;
```

NOTE: The PROCEDURE PRINT used 1.03 seconds.

DO LOOPS

OBS	VAR1	VAR2
1	1	FEMALE
2	1	MALE
3	2	FEMALE
4	2	MALE
5	3	FEMALE
6	3	MALE
7	4	FEMALE
8	4	MALE
9	5	FEMALE
10	5	MALE

EXAMPLE 4: Creating a data set without input data

```
91 DATA ONE ;
92
93 SET CLASSLIB.CLASS ;
94
95 N3= 0;
96
97 IF SEX='F' THEN DO;
98
99     N1 = 2 ;
100
101     DO N2 = 1 TO 3 ;
102
103         N3 = N3 + 2;
104
105     END ;
106 END ;
107
108 ELSE DO ;
109
110     N1 = 4 ;
111
112 END ;
113 RUN;
```

NOTE: The data set WORK.ONE has 6 observations and 8 variables.

NOTE: The DATA statement used 1.54 seconds.

```
115 PROC PRINT ;
116 TITLE 'DO LOOPS' ;
117 RUN;
```

NOTE: The PROCEDURE PRINT used 0.08 seconds.

DO LOOPS

OBS	NAME	SEX	AGE	HT	WT	N3	N1	N2
1	CHRISTIANSEN	M	37	71	195	0	4	.
2	HOSKING J	M	31	70	160	0	4	.
3	HELMS R	M	41	74	195	0	4	.
4	PIGGY M	F	.	48	.	6	2	4
5	FROG K	M	3	12	1	0	4	.
6	GONZO		14	25	45	0	4	.

Other Forms of Iterative DO Groups

Two additional forms of DO loop available are the DO WHILE and DO UNTIL loops. These are used in cases in which you want to execute a loop as long as (WHILE) some logical expression is true or as long as some logical expression is false (UNTIL it is TRUE).

General form:

DO WHILE (Expression);

Executable statements

END;

In a **DO WHILE** statement, the expression is evaluated at the top of the loop, before the statements in the DO group are executed. If the expression is true, the DO group is executed.

Examples:

- 1) Count the number of years needed to double an initial amount (CAPITAL) at a given rate of interest (RATE), compounding yearly.

WORK.COMPOUND

<u>CAPITAL</u>	<u>RATE</u>
1000	.10

```
DATA DOUBLE;  
  SET WORK.COMPOUND;  
  TOTAL=CAPITAL; TERM=0;  
  DO WHILE(TOTAL LT (CAPITAL*2));  
    TOTAL=TOTAL+(TOTAL*RATE);  
    TERM=TERM+1;  
  END;  
RUN;
```

- 2) Write a program to "make change"; that is to compute the smallest number of quarters, dimes, nickels, and pennies which add to an arbitrary amount between 0 and 99.

```

CHANGE
AMOUNT
  97
  32
  11

```

```

DATA COINS;

  SET CHANGE;

  LEFT=AMOUNT;
  QUARTERS=0; DIMES=0; NICKELS=0; PENNIES=0;

  DO WHILE (LEFT GE 25);
    QUARTERS=QUARTERS+1;
    LEFT=LEFT-25;
  END;

  DO WHILE (LEFT GE 10);
    DIMES=DIMES+1;
    LEFT=LEFT-10;
  END;

  IF (LEFT GT 5) THEN DO;
    NICKELS=1;
    LEFT=LEFT-5;
  END;

  PENNIES=LEFT;

  DROP LEFT;
RUN;

```

COINS				
<u>AMOUNT</u>	<u>QUARTERS</u>	<u>DIMES</u>	<u>NICKELS</u>	<u>PENNIES</u>
97	3	2	0	2
32	1	0	1	2
11	0	1	0	1

DO UNTIL (Expression);

In a DO UNTIL Statement, the expression is evaluated at the bottom of the loop, after the statements in the DO group are executed. If the expression is true, the DO group is not executed again. The DO group is always executed at least once.

General form:

DO UNTIL (Expression);

Executable statements

END;

EXAMPLE:

Find the position of the first occurrence of a character (CHAR) in NAME:

I=1;

CHAR='*';

NAME='JOE*SMITH';

DO UNTIL (SUBSTR(NAME,I,1) EQ CHAR));

I=I+1;

END;

WHAT HAPPENS IF **CHAR** IS NOT IN **NAME**?

ARRAYS

ratio1 = verbal1/math1 ;

ratio2 = verbal2/math2 ;

ratio3 = verbal3/math3;

if date1=98 or date1=99 then date1=.;

if date2=98 or date2=99 then date2=.;

if date3=98 or date3=99 then date3=.;

ARRAY – group of variables given a collective name

- ❖ Calculations in the data step can operate on arrays as they can on variables.
- ❖ The ARRAY statement can be used to execute one or more statements for each of a group of related variables.
- ❖ ARRAY statements are usually used in conjunction with DO loops.

ARRAYS(EXPLICIT) SYNTAX

ARRAY name{dim} [\$] [len] [elements] [(starting_values)] ;

NAME is the name of the array; cannot be a variable or an array already in the data set

DIM is the number of elements in the array. An asterick(*) may also be entered. The DIM can an also be enclosed in brackets[] or parentheses.

The \$ indicates that the elements of the array are character variables that have not yet been assigned to SAS.

The **LEN** indicates the length of any variables that have not yet been assigned to SAS.

The **ELEMENTS** are the names of the variables in the array. Any combination of variable lists and variable names are permitted. All elements in the array must be of the same data type.

STARTING_VALUES indicate initial values for array elements. These values are separated by a comma and/or one or more blanks. Starting values do not replace variables already known to SAS.

ARRAYS(EXPLICIT) EXAMPLES

THE FOLLOWING ARE VALID ARRAY STATEMENTS:

```
ARRAY TEST{3} TEST1-TEST3 ;
```

```
ARRAY TEST{*} TEST1-TEST3 ;
```

```
ARRAY TEST{3} ; /* DEFINES VARIABLES TEST1-TEST3 */
```

```
ARRAY DAY{4} $2 DAY1-DAY4 ('S','M','TU','W') ;
```

```
ARRAY X{*} _NUMERIC_ ;
```

```
ARRAY Y{*} TEST1-TEST3 SCORE4-SCORE6 ;
```

```
ARRAY Z{*} X Y ;
```

```
ARRAY {5,3} SCORE1-SCORE15 ;
```

EXAMPLES

- 1) Convert the homework scores HW1-HW3 from a 10 point scale to a 100 point scale.

DATA SET OLDHW

NAME	HW1	HW2	HW3	EXAM
Christiansen	10	10	9	97
Hosking J	9	7	4	38
Helms R	8	9	.	88
Piggy M	10	10	10	90
Frog K	5	5	3	64
Gonzo	9	5	.	100

```
DATA NEWHW;  
  
SET OLDHW;  
  
DROP 1;  
  
ARRAY HOMEWORK (3) HW1 HW2 HW3;  
  
DO I = 1 TO 3;  
    HOMEWORK (I) = HOMEWORK(I) * 10;  
  
END;  
RUN;
```

DATA SET NEWHW

NAME	HW1	HW2	HW3	EXAM
Christiansen	100	100	90	97
Hosking J	90	70	40	38
Helms R	80	90	.	88
Piggy M	100	100	100	90
Frog K	50	50	30	64
Gonzo	90	50	.	100

- 2) The DATA step below searches through the homework scores in NEWHW, recording each student's worst homework score.

```
DATA WORK.WORST;
    SET WORK.NEWHW;
    KEEP NAME SCORE;
    ARRAY HW (3) HW1 HW2 HW3;
    SCORE = 101;
    DO J = 1 TO 3;
        IF (HW(J) LT SCORE) THEN SCORE = HW(J);
    END;
RUN;
```

WORST

<u>NAME</u>	<u>SCORE</u>
Christiansen	90
Hosking J	40
Helms R	.
Piggy M	100
Frog K	30
Gonzo	.

- 3) Convert HW1 - HW3 and exam from the NEWHW data set to letter grades L1-L3 and LEXAM using the following grading scale:

Grade > 90	H
80 < Grade < 90	P
70 < Grade < 80	L
Grade < 70 or missing	F

```

DATA LETTERHW;
  SET NEWHW;
  ARRAY HW (4) HW1 - HW3 EXAM;
  ARRAY L (4) $1 L1-L3 LEXAM;
  DO I = 1 TO 4;
    IF (HW(I) LT 70) THEN L(I) = 'F';
    ELSE IF (70 LE HW(I) LT 80) THEN L(I) = 'L';
    ELSE IF (80 LE HW(I) LT 90) THEN L(I) = 'P';
    ELSE L(I) = 'H';
  END;
RUN;

```

DATA SET LETTERHW

NAME	HW1	HW2	HW3	EXAM	L1	L2	L3	EXAM
Christiansen	100	100	90	97	H	H	H	H
Hosking J	90	70	40	38	H	L	F	F
Helms R	80	90	.	88	P	H	F	P
Piggy M	100	100	100	90	H	H	H	H
Frog K	50	50	30	64	P	F	F	F
Gonzo	90	50	.	100	H	F	F	H

- 4) SAS data set ONE contains one record per subject. Each record contains three scores. Convert the data sets to 3 records per subject, where each record contains one score and The score number (1,2, or 3).

```

31 PROC PRINT DATA=ONE ;
32   TITLE 'DATA ONE';
33   RUN;
NOTE: THE PROCEDURE PRINT USED 0.00 SECONDS.
34
35
36 DATA TWO ;
37   SET ONE ;
38
39   DROP SCORE1-SCORE3 ;
40
41   ARRAY S{3} SCORE1-SCORE3 ;
42
43   DO I=1 TO 3 ;
44     SCORE = S{I} ;
45     OUTPUT ;
46   END;
47
48 RUN;
NOTE: THE DATA SET WORK.TWO HAS 6 OBSERVATIONS AND 3 VARIABLES.
NOTE: THE DATA STATEMENT USED 2.00 SECONDS.
49
50 PROC PRINT DATA=TWO ;
51   TITLE 'DATA TWO';
52   RUN;
NOTE: THE PROCEDURE PRINT USED 1.00 SECONDS.

```

DATA ONE

OBS	ID	SCORE1	SCORE2	SCORE3
1	101	20	30	40
2	102	50	60	70

DATA TWO

OBS	ID	I	SCORE
1	101	1	20
2	101	2	30
3	101	3	40
4	102	1	50
5	102	2	60
6	102	3	70

- 5) Convert data set two (from previous example) from three records per subject back to one record per subject.

```

DATA THREE ;
  SET TWO ;

  RETAIN SCORE1-SCORE3 ;

  DROP I SCORE ;

  IF I=1 THEN SCORE1=SCORE ;
  IF I=2 THEN SCORE2=SCORE ;
  IF I=3 THEN SCORE3=SCORE ;

  OR

  ARRAY S{3} SCORE1-SCORE3 ;
  DO J=1 TO 3;
    IF I=J THEN S{J}=SCORE ;
  END ;
  or S{I} = SCORE ;

  IF I=3 THEN OUTPUT ;

RUN;

```

ID	I	SCORE	SCORE1	SCORE2	SCORE3
101	1	20	20	.	.
101	2	30	20	30	.
101	3	40	20	30	40
201	1	50	50	30	40
201	2	60	50	60	40
201	3	70	50	60	70

- **Note:** The above is not a general solution. A more general solution will be discussed in the next section.

- 6) Data set one contains three systolic blood pressure measures for each subject. Find the first non-missing measure.

```

34 DATA TWO ;
35 SET ONE ;
36
37 ARRAY S{3} SBP1-SBP3 ;
38
39 SBP=. ; /* SBP will contain the first non-missing measure */
40 I=1 ;
41 DO UNTIL ((SBP> .Z) OR (I>3 )) ;
42   IF S{I} > .Z THEN SBP=S{I} ;
43   I=I+1 ;
44 END ;
45 RUN;

```

NOTE: THE DATA SET WORK.TWO HAS 4 OBSERVATIONS AND 5 VARIABLES.

NOTE: THE DATA STATEMENT USED 3.00 SECONDS.

```

46
47 PROC PRINT DATA=TWO ;
48 TITLE 'DATA TWO' ;
49 RUN;

```

NOTE: THE PROCEDURE PRINT USED 1.00 SECONDS.

DATA ONE

OBS	SBP1	SBP2	SBP3
1	.	90	104
2	101	120	130
3	.	.	95
4	.	.	.

DATA TWO

OBS	SBP1	SBP2	SBP3	SBP	I
1	.	90	104	90	3
2	101	120	130	101	2
3	.	.	95	95	4
4	4

ARRAYS(EXPLICIT) EXAMPLES

```
ARRAY VAR{3} VAR1-VAR3 ;
```

```
X1 = VAR{1} ;
```

```
J=1 ;
```

```
X1 = VAR{J} ;
```

YOU CAN REFER TO AN ENTIRE ARRAY:

```
X2 = SUM(OF VAR{*} ) ;
```

DIM FUNCTION

the DIM function returns the number of elements in a dimension of an array.

```
ARRAY TEST{*} TEST1-TEST3 ;
```

```
DO I=1 TO DIM(TEST) ;
```

```
TEST{I} = TEST{I} + 10 ;
```

```
END;
```

```
ARRAY VAR{*} _NUMERIC_ ;
```

```
DO I=1 TO DIM(VAR) ;
```

```
IF VAR{I}=. THEN VAR{I}=0;
```

```
END;
```

ARRAYS(IMPLICIT) SYNTAX

ARRAY name(index) [\$] [len] [elements] [(starting_values)] ;

NAME is the name of the array; cannot be a variable or an array already in the data set

INDEX gives the name of a variable whose value defines the current element of the array. The index must be enclosed in parentheses; brackets and braces are not allowed.

The \$ indicates that the elements of the array are character variables that have not yet been assigned to SAS.

The *LEN* indicates the length of any variables that have not yet been assigned to SAS.

The *ELEMENTS* are the names of the variables in the array. Any combination of variable lists and variable names are permitted. All elements in the array must be of the same data type

STARTING_VALUES indicate initial values for array elements. These values are separated by a comma and/or one or more blanks. Starting values do not replace variables already known to SAS.

ARRAYS(IMPLICIT) EXAMPLES

THE FOLLOWING ARE VALID ARRAY STATEMENTS:

```
ARRAY TEST(I) TEST1-TEST3 ;
```

```
ARRAY TEST TEST1-TEST3 ;
```

```
ARRAY DAY(J) $2 DAY1-DAY4 ('S','M','TU','W') ;
```

```
ARRAY X _NUMERIC_ ;
```

```
ARRAY Y TEST1-TEST3 SCORE4-SCORE6 ;
```

```
ARRAY Z(L) X Y ;
```

ARRAYS(IMPLICIT) DO OVER PROCESSING

- ❖ The DO OVER statement is used to process the elements of an implicit array.
- ❖ Repeats the statements inside the DO loop for all elements of the array.
- ❖ EXAMPLES:

```
ARRAY F F1-F100 ; /* BOTH ARRAYS USE _I_ AS AN INDEX */  
ARRAY C C1-C100 ;  
DO OVER F; /* REPEATS DO LOOP 100 TIMES */  
C=(F-32)*5/9 ;  
END;
```

```
ARRAY S(I) SCORE1-SCORE3 ;  
DO I=1 TO 3 ;  
IF S>98 THEN S=. ;  
END;
```

```
ARRAY S SCORE1-SCORE3 ;  
DO _I_=1 TO 3 ;  
IF S>98 THEN S=. ;  
END;
```

```
ARRAY S SCORE1-SCORE3 ;  
DO OVER S ;  
IF S>98 THEN S=. ;  
END;
```

TRANSFORMATIONS INVOLVING MISSING VALUES

- ❖ Missing values occur in most data and it is important to understand what effect these missing values have on transformations of variables.
- ❖ Missing values for numeric variables are:
 - ❖ presented in programming statements by: ._, ., .A-.z
 - ❖ checked for missing by: if X <=.Z. then DELETE;
- ❖ Missing value for character variables are
 - ❖ Represented by: Blank Field
 - ❖ Checked for missing by: If G=' ' then delete;
- ❖ Variables can be assigned missing values
 - ❖ If WT GT 500 then WT=.
 - ❖ If STATE NE 'NC' then STATE=' ';
- ❖ Missing values propagate through arithmetic expressions.
- ❖ Illegal arguments sent to functions return missing values.

DATA SET A

X	Y
4	2
.	7
-3	2

```
DATA WORK.B;  
  
  SET WORK.A;  
  
  Z = X + Y;  
  
  S = SQRT(X);  
  
RUN;
```

DATA SET B

X	Y	Z	S
4	2	6	2
.	7	.	.
-1	2	-1	.

- ❖ Missing values are treated like minus infinity in comparison expressions,
- ❖ Special missing values compare in the sort sequence.

DATASET C

X	Y
0	4
-12	.
.	9
A	B

```
DATA WORK.D1
```

```
  SET WORK.C;
```

```
  IF X LT Y then T1 = 'TRUE'
```

```
    ELSET T1 = 'FALSE';
```

```
RUN;
```

DATA SET D1

X	Y	T1
0	4	TRUE
-12	.	FALS
.	9	TRUE
A	B	TRUE

- ❖ Missing values are false in logical operations.

```
DATA WORK.D2;  
    SET WORK.C;  
    IF X THEN T2 = 'TRUE';  
        ELSE T2 = 'FALSE';  
RUN;
```

DATA SET D2

X	Y	T2
0	4	FALS
-12	.	TRUE
.	9	FALS
A	B	FALS

- ❖ Special missing values compare in the sort sequence

```
DATA WORK.D3;  
    SET WORK.C;  
    IF Y LT .B THEN T3 = 'TRUE'  
    ELSE T3 = 'FALSE';  
RUN;
```

DATA SET D3

X	Y	T3
0	4	FALS
-12	.	TRUE
.	9	FALS
A	B	FALS

- ❖ Functions that compute sample statistics use only nonmissing values of the arguments.

DATA SET E

A	B	C
3	2	7
.	4	9

```
DATA WORK.F;  
  SET WORK.E;  
  TOT = A + B + C;  
  AVE = TOT/3;  
  S = SUM(A,B,C);  
  M = MEAN(OF A--C);  
RUN;
```

DATA SET F

A	B	C	TOT	AVE	S	M
3	2	7	12	4	12	4.0
.	4	9	.	.	13	6.5

- ❖ The SUM function can be used to prevent cumulative totals of variables involving missing values from becoming missing.

```
DATA CUMLAT;  
    SET CLASSLIB.CLASS;  
    RETAIN CUMHT CUMWT CUMAGE;  
    CUMHT = SUM(CUMHT,HT);  
    CUMWT = SUM(CUMWT,WT);  
    CUMAGE = SUM(CUMAGE,AGE);  
RUN;
```

DATA SET CUMLAT

NAME	SEX	AGE	HT	WT	CUMHT	CUMWT	CUMAGE
Christiansen	M	37	71	195	71	195	37
Hosking J	M	31	70	160	141	355	68
Helms R	M	41	74	195	215	550	109
Piggy M	F	.	48	.	263	550	109
Frog K	M	3	12	1	275	551	112
Gonzo		14	25	45	300	596	126

THE SUM STATEMENT

The SUM statement can be used to sum expressions over observations. It implies a RETAIN and only sums nonmissing values.

SYNTAX

Sum_variable + Expression;

```
DATA CUMTLAT2;
    SET CLASSLIB.CLASS;
    CUMHT + HT;
    CUMWT + WT;
    CUMAGE + AGE;
RUN;
```

DATA SET CUPLAT2

NAME	SEX	AGE	HT	WT	CUMHT	CUMWT	CUMAGE
Christiansen	M	37	71	195	71	195	37
Hosking J	M	31	70	160	141	355	68
Helms R	M	41	74	195	215	550	109
Piggy M	F	.	48	.	263	550	109
Frog K	M	3	12	1	275	551	112
Gonzo		14	25	45	300	596	126