(8) DATES

SAS has numerous informats for reading dates and formats for displaying dates. Dates can be read with either numeric, character, or date informats. If you read a date as a character variable, you will not be able to use it in any calculations, e.g. to calculate the number of days that separate two given dates. If you read a date with a numeric format, SAS will store the date as a number, but not as a numeric value that can be used to calculate time differences. However, if you use one of the SAS date informats, the date is stored as a numeric variable that can be used in calculations.

No matter how a date might appear in a raw data file, there probably is a SAS informat that will allow it to be read and stored as a numeric variable. The following shows a series of SAS informats that can be used to read a specified date (March 15, 1994) stored in a number of different forms...

```
RAW DATA          INFORMAT
--------          --------
031594            mmddyy6.
940315            yymmdd6.
03/15/94          mmddyy8.
94/03/15          yymmdd8.
03 15 94          mmddyy8.
94 03 15          yymmdd8.
03151994          mmddyy8.
15mar94           date7.
03/15/1994        mmddyy10.
15mar1994         date9.
```

Once a date is read with a date informat, SAS stores the date as a number, i.e. the number of days between the date and January 1, 1960 (3/15/1994 is stored as 12492). This permits the use of dates in calculations. For example, imagine you were given a data file that contained a date admission to a hospital and a date of discharge. You might be interested in computing the length stay.

...Example 8.1...

```
data patients;
informat admit disch mmddyy8.;                                          1
format admit disch date9.;                                              2
input admit disch;                                                      3
los = disch - admit;                                                    4
label
los = 'LENGTH OF STAY'
admit = 'ADMIT DATE'
disch = 'DISCHARGE DATE'
;
datalines;
03/21/90 04/01/90
05/15/90 05/16/90
;
run;

           ADMIT     DISCHARGE      LENGTH
Obs         DATE       DATE       OF STAY
 1     21MAR1990    01APR1990        11                                 5
 2     15MAY1990    16MAY1990         1
```

1 An INFORMAT statement tells SAS to read the variables ADMIT and DISCH with a DATE informat.
2 A FORMAT statement tells SAS to display the two variables as dates, not simply as numeric data.
3 LIST input is used to read the two dates.
4 A new variable is created, LOS, that is the difference in days between DISCH and ADMIT.
5 The output from PROC PRINT shows that the variables DISCH and ADMIT were treated as dates and that a difference in days was calculated correctly.

In addition to using an INFORMAT to tell SAS to treat the variables ADMIT and DISCH as dates, a FORMAT is used to control how the dates will be displayed. Without a format, the dates would be displayed as numbers...

```
proc print data=patients label;
* temporarily remove formats from variables;
format _all_;
run;
```

```
       ADMIT    DISCHARGE     LENGTH
Obs     DATE       DATE      OF STAY
 1     11037      11048         11
 2     11092      11093          1
```

The numbers shown under ADMIT and DISCHARGE are the number of days each date is from 1/1/1960. The calculation of LOS is still correct.

Just as SAS has a many INFORMATS for reading dates, it also has a wide selection of FORMATS for displaying dates, e.g. March 15, 1994 can be displayed as follows...

```
FORMAT                      DISPLAY
------                      -------
mmddyy.                   03/15/94
mmddyy6.                  031594
mmddyy8.                  03/15/94
yymmdd.                   94-03-15
yymmdd6.                  940315
yymmdd8.                  94-03-15
date7.                    15MAR94
date9.                    15MAR1994
weekdate.                       TUESDAY, MARCH 15, 1994
weekdatx.                       TUESDAY,  15 MARCH 1994
worddate.                     MARCH 15, 1994
```

The last three formats will display the leading blanks that are shown, e.g. using WORDDATE. results in five leading blanks. Example 8.1 used the DATE9. format to control the display of the variables DISCH and ADMIT.

In addition to the informats and formats that SAS supplies for reading and writing dates, there are also a number of SAS functions that can be used to work with dates.

...Example 8.2....

```
proc format;                                                                    1
value dayofwk
1 = "SUNDAY"      2 = "MONDAY"      3 = "TUESDAY"
4 = "WEDNESDAY"   5 = "THURSDAY"    6 = "FRIDAY"      7 = "SATURDAY";
run;

data function;
format dob future now mmddyy10. dob_dow dayofwk.;                               2

dob      = '15jul75'd;                                                          3
dob_dow  = weekday(dob);                                                        4
future   = mdy(12,31,2010);                                                     5
now      = today();                                                             6
agethen  = (future - dob) / 365.25;                                             7
agenow   = (now - dob) / 365.25;
age_then = yrdif(dob,future,'actual');                                          8
age_now  = yrdif(dob,now,'actual');

label
dob      = "BIRTHDATE"
dob_dow  = "DAY OF WEEK/BIRTH DATE"
future   = "DAY IN/THE FUTURE"
```

```
now      = "TODAY"
agethen  = "AGE IN THE FUTURE (OLD WAY)"
agenow   = "AGE NOW (OLD WAY)"
age_then = "AGE IN THE FUTURE (YRDIF)"
age_now  = "AGE NOW (YRDIF)"
;
run;

proc print data=function label;
run;
```

BIRTHDATE	DAY IN THE FUTURE	TODAY	DAY OF WEEK (BIRTH DATE)	AGE IN THE FUTURE (OLD WAY)	AGE NOW (OLD WAY)	AGE IN THE FUTURE (YRDIF)	AGE NOW (YRDIF)
07/15/1975	12/31/2010	04/02/2001	TUESDAY	35.4634	25.7166	35.4630	25.7151

1 PROC format is used to create a format that will allow the printing of a literal day of the week given a number in the range 1 to 7.
2 FORMATS are assigned to several variables that are dates.
3 A date variable can be created if the date is expressed as shown (day/month/year, with a 3-character literal month) and the date is in quotes followed by the letter d.
4 The day of the week of any given day can be determined via the WEEKDAY function - the function returns a number in the range of 1 to 7 (Sunday-to-Saturday). The number is a numeric variable.
5 A date variable can also be created via the MDY function if the month, day, and year are supplied.
6 The current date can be assigned to a variable using the TODAY() function.
7 New variables (AGETHEN, AGENOW) are created. These are ages in years
8 A SAS function, YRDIF, is used to create new variables similar to those created in step #7. The YRDIF function requires three arguments: the starting date of an interval; the ending date of an interval; the method to be used to calculate the interval. To get a true difference in years, use the word ACTUAL in quotes as shown in the example.

The output from PROC PRINT shows all the variables in the data set. The variables created with the YRDIF function are not too different from the old method of calculating the age in years.

You can use the capability of expressing a date referred to in step #3 in example 8.2 to group observations by date ranges with a user-written format. The next example demonstrates this capability, together with some SAS-supplied formats that can be used to group observations.

...Example 8.3...

```
proc format;                                                                    1
value interval
'01jan1997'd - '30jun1997'd = '1ST HALF 1997'
'01jul1997'd - '31dec1997'd = '2ND HALF 1997'
'01jan1998'd - '30jun1998'd = '1ST HALF 1998'
'01jul1998'd - '31dec1998'd = '2ND HALF 1998'
;
run;

data admits;
input admit1 :  mmddyy10. @@;                                                   2
admit2 = admit1; admit3 = admit1; admit4 = admit1;
admit5 = admit1; admit6 = admit1; admit7 = admit1;
label
admit1 = 'USER-WRITTEN FORMAT'
admit2 = 'YEAR FORMAT'
admit3 = 'MONTH FORMAT'
admit4 = 'MONYY7 FORMAT'
admit5 = 'QTR (QUARTER) FORMAT'
admit6 = 'MONNAME (MONTH NAME) FORMAT'
admit7 = 'DOWNAME (DAY OF WEEK NAME) FORMAT'
;
```

```
datalines;
01181998 02111998 02281998 02161998 02271998 03291998 04181998 05081998
05071998 05101998 06031998 08021998 08131998 07241998 08151998 10011998
01081997 01251997 02041997 02171997 03071997 02181997
03161997 03281997 03301997 03271997 04031997 04271997
05311997 06071997 06131997 05311997 06181997 06161997
06201997 07141997 08071997 07131997 08071997 09051997
;
run;

proc freq data=admits;                                                    3
table admit1-admit7;
format
admit1 interval.
admit2 year.
admit3 month.
admit4 monyy7.
admit5 qtr.
admit6 monname.
admit7 downame.
;
run;
```

USER-WRITTEN FORMAT

admit1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1ST HALF 1997	19	47.50	19	47.50
2ND HALF 1997	5	12.50	24	60.00
1ST HALF 1998	11	27.50	35	87.50
2ND HALF 1998	5	12.50	40	100.00

YEAR FORMAT

admit2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1997	24	60.00	24	60.00
1998	16	40.00	40	100.00

MONTH FORMAT

admit3	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3	7.50	3	7.50
2	7	17.50	10	25.00
3	6	15.00	16	40.00
4	3	7.50	19	47.50
5	5	12.50	24	60.00
6	6	15.00	30	75.00
7	3	7.50	33	82.50
8	5	12.50	38	95.00
9	1	2.50	39	97.50
10	1	2.50	40	100.00

MONYY7 FORMAT

admit4	Frequency	Percent	Cumulative Frequency	Cumulative Percent
JAN1997	2	5.00	2	5.00
FEB1997	3	7.50	5	12.50
MAR1997	5	12.50	10	25.00
APR1997	2	5.00	12	30.00
MAY1997	2	5.00	14	35.00
JUN1997	5	12.50	19	47.50
JUL1997	2	5.00	21	52.50
AUG1997	2	5.00	23	57.50
SEP1997	1	2.50	24	60.00
JAN1998	1	2.50	25	62.50
FEB1998	4	10.00	29	72.50
MAR1998	1	2.50	30	75.00
APR1998	1	2.50	31	77.50
MAY1998	3	7.50	34	85.00
JUN1998	1	2.50	35	87.50
JUL1998	1	2.50	36	90.00
AUG1998	3	7.50	39	97.50
OCT1998	1	2.50	40	100.00

QTR (QUARTER) FORMAT

admit5	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	16	40.00	16	40.00
2	14	35.00	30	75.00
3	9	22.50	39	97.50
4	1	2.50	40	100.00

MONNAME (MONTH NAME) FORMAT

admit6	Frequency	Percent	Cumulative Frequency	Cumulative Percent
January	3	7.50	3	7.50
February	7	17.50	10	25.00
March	6	15.00	16	40.00
April	3	7.50	19	47.50
May	5	12.50	24	60.00
June	6	15.00	30	75.00
July	3	7.50	33	82.50
August	5	12.50	38	95.00
September	1	2.50	39	97.50
October	1	2.50	40	100.00

DOWNAME (DAY OF WEEK NAME) FORMAT

admit7	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Wednesday	4	10.00	4	10.00
Saturday	7	17.50	11	27.50
Tuesday	2	5.00	13	32.50
Monday	4	10.00	17	42.50
Friday	8	20.00	25	62.50
Sunday	8	20.00	33	82.50
Thursday	7	17.50	40	100.00

1 A format is created that groups dates into four intervals
2 The data set ADMITS is created with seven variables, admit1-admit7, all having the same value.
3 Various formats are used to group observations based on the value of a variable that contains a date.

Two functions can be used to work with date intervals, INTCK and INTNX. The INTCK function computes the number of intervals between any two given dates, with the interval being either day, week, month, qtr, or year. The INTNX allows you to specify the time interval (same choices as with INTCK), a starting date, and the number of intervals you would like to cross. SAS then returns a date. These functions are not discussed here in any detail.

...YEARCUTOFF

When a year is expressed with only two digits, how does SAS know what century to use when it creates a numeric date value? There is a SAS system option called YEARCUTOFF and its value determines how two digit dates are evaluated. The default value of this option in version 8 is 1920. Any two digit date is assumed to occur AFTER 1920. If that value is changed via an OPTIONS statement, SAS will use the new value.

...Example 8.4...

```
options yearcutoff=1900;                                                    1
data test;
format dt weekdate.;
dt = mdy(01,12,10);                                                         2
label dt = 'YEARCUTOFF 1900';
run;

proc print data=test label noobs;
run;

options yearcutoff=1920;                                                    3
data test;
format dt weekdate.;
dt = mdy(01,12,10);
label dt = 'YEARCUTOFF 1920';
run;

proc print data=test label noobs;
run;

          YEARCUTOFF 1900
    Wednesday, January 12, 1910

         YEARCUTOFF 1920
    Tuesday, January 12, 2010
```

1 The YEARCUTOFF option for two-digit dates is set to 1900. Any two-digit date is considered as occurring on or after 1900.
2 The MDY function is used to create a date variable.
3 The YEARCUTOFF option is changed to 1920. Any two-digit date is considered as occurring on or after 1920.

You can see the difference in how SAS treats the variable DT depending on the value of YEARCUTOFF.

...LONGITUDINAL DATA

There is a feature of a sorted data set that allows you to find the first and/or last observation in a sequence using a data step. You already know that you read a SAS data set with a SET statement. You also know about BY-GROUP processing from working with grouped data. A combination of a SET statement and a BY statement within a data step gives you access to two SAS-created variables.

...Example 8.5...

```
data manyids;
input id : $2. visit : mmddyy.;                                          1
format visit mmddyy8.;
datalines;
01  01/05/89
01  05/18/90
01  11/11/90
01  02/18/91
02  12/25/91
03  01/01/90
03  02/02/91
04  05/15/91
04  08/20/91
04  03/23/92
04  07/05/92
;
run;

proc sort data=manyids;                                                  2
by id visit;
run;

data oneid;
set manyids;                                                             3
by id;
if first.id then output;                                                 4
run;

proc print data=oneid;
run;

OBS    ID       VISIT                                                    5
 1     01    01/05/89
 2     02    12/25/91
 3     03    01/01/90
 4     04    05/15/91
```

1 A data set is created containing two variables, ID and VISIT (a date variable).
2 The data set is sorted by ID and by VISIT (date) within each ID.
3 A new data set is created. The date are read with a SET/BY combination. Using BY ID; creates a new TEMPORARY variable name FIRST.ID that can be used within the data step.
4 The new data set contains only the observation with the first VISIT within each ID.

A new, SAS-created, temporary variable in example 8.5 is FIRST.ID. When you use a SET statement in combination with a BY statement, SAS will create a new variable for each by-variable. There is only one by-variable (ID), so SAS created FIRST.ID. That is not the entire story of SET/BY since SAS also creates a LAST.ID (a two-for-one deal), i.e. for every variable in the BY statement, SAS will create a FIRST.<by-variable> and a LAST.<by-variable>. The variables only exist for the duration of the data step and do not become part of any SAS dataset.

FIRST. and LAST. variables take on only two values, 1 or zero. The following are the values of FIRST.ID and LAST.ID when the sorted version of the dataset MANYIDS is used in the data step shown in example 8.5...

```
                FIRST.ID   LAST.ID
01   01/05/89       1          0
01   05/18/90       0          0
01   11/11/90       0          0
01   02/18/91       0          1
02   12/25/91       1          1
03   01/01/90       1          0
03   02/02/91       0          1
04   05/15/91       1          0
04   08/20/91       0          0
04   03/23/92       0          0
04   07/05/92       0          1
```

You can see that the first observation in a given sequence results in a FIRST. value of 1, while the last observation results in a LAST. value of 1. If an observation is both the first and last observation in a sequence (i.e. the only one as with ID 02), both the FIRST. and LAST. values are 1. If an observation is not the first or the last in a sequence, both the FIRST. and LAST. values are 0. When you use a statement such as that in example 8.5...

```
if first.id then output;
```

you are asking SAS to evaluate whether he value of the FIRST. variable is 1 or zero. If it is 1, then SAS performs the task. You could write the above statement in a number of different ways. All would result in the same data set being created....

```
if first.id eq 1 then output;

if first.id;

if not first.id then delete;
```

How could you change example 8.5 to create a data set with the last VISIT rather than the first?

...Example 8.6...

```
proc sort data=manyids;
by id descending visit;
run;

data oneid;
set manyids;
by id;
if first.id then output;
run;

proc print data=oneid;
run;

OBS    ID       VISIT
 1     01     02/18/91
 2     02     12/25/91
 3     03     02/02/91
 4     04     07/05/92
```

Since the data are sorted in descending date order within each ID, the first observation within each ID group is the last VISIT. You could also modify the data step instead of the sort.

...Example 8.7...

```
proc sort data=manyids;
by id visit;
run;

data oneid;
set manyids;
by id;
if last.id then output;
run;

proc print data=oneid;
run;
```

```
OBS    ID       VISIT
 1     01     02/18/91
 2     02     12/25/91
 3     03     02/02/91
 4     04     07/05/92
```

What if your task was to determine how long any individual in the dataset MANYIDS had been part of your study, i.e. what is the difference in days between the first and last visits? This can be done in a number different ways. One involves match-merging data sets (that's for later in the semester). The following only requires one sort, one data step, plus the use of RETAIN and DO-END statements.

...Example 8.8...

```
proc sort data=manyids;
by id visit;
run;

data duration;
retain firstvis;;
set manyids;
by id;
if first.id then firstvis=visit;
if last.id then do;
   diffdays = visit - firstvis;
   output;
end;
run;

proc print data=duration;
var id firstvis visit diffdays;
format firstvis visit mmddyy6.;
run;
```

```
OBS    ID    FIRSTVIS       VISIT    DIFFDAYS
 1     01    01/05/89    02/18/91       774
 2     02    12/25/91    12/25/91         0
 3     03    01/01/90    02/02/91       397
 4     04    05/15/91    07/05/92       417
```

The IF FIRST.ID THEN... statement tells SAS to store the value of the variable VISIT as variable FIRSTVIS when the first observation within a given ID is encountered. The RETAIN statement tells SAS to hold onto the value assigned to FIRSTVIS rather than set it back to missing each time SAS cycles back to the top of the data step. Remember, the default behavior of SAS within a data step is to set the value of MOST (not all) variables to missing each time SAS reaches the top of the data step. The RETAIN statement can selectively alter that behavior. The DO-END statement allows you to perform multiple actions. Since the DO-END statement is embedded in an IF-THEN statement, SAS will perform multiple actions if the IF-THEN statement is TRUE. When the observation within an ID group is found, the date of the first visit is subtracted from the date of last visit and an observation is writtem to the data set by using an OUTPUT statement.

What if in addition to the VISIT data, you also had another variable that measured some characteristic of an individual at each visit, e.g. cholesterol, that you hoped was changing over time as the result of some study intervention. How could you modify the data step in example 8.8 to determine both the difference in days and cholesterol values between first and last visits?

...Example 8.9...

```
data manyids;
input id : $2. visit : mmddyy8. chol;
format visit mmddyy8.;
datalines;
01  01/05/89   400
01  05/18/90   350
01  11/11/90   305
01  02/18/91   260
02  12/25/91   200
03  01/01/90   387
03  02/02/91   380
04  05/15/91   380
04  08/20/91   370
04  03/23/92   355
04  07/05/92   261
;
run;

proc sort data=manyids;
by id visit;
run;

data twodiffs;
retain firstvis firstcho;
format firstvis mmddyy8.;
set manyids;
by id;
if first.id then do;
   firstvis=visit;
   firstcho=chol;
end;
if last.id then do;
   diffdays = visit - firstvis;
   diffchol = chol  - firstcho;
   output;
end;
run;

proc print data=twodiffs;
var id firstvis visit diffdays firstcho chol diffchol;
format firstvis mmddyy8.;
run;
```

OBS	ID	FIRSTVIS	VISIT	DIFFDAYS	FIRSTCHO	CHOL	DIFFCHOL
1	01	01/05/89	02/18/91	774	400	260	-140
2	02	12/25/91	12/25/91	0	200	200	0
3	03	01/01/90	02/02/91	397	387	380	-7
4	04	05/15/91	07/05/92	417	380	261	-119

In example 8.8, the value of the variable VISIT was stored when the first observation within an ID group was read. In example 8.9, both the date (the value of the variable VISIT) and the initial cholesterol reading are stored. When the last observation in an ID group is read, the difference in days between the first and last visit, and the difference in cholesterol can be calculated.

If you read a data set with a combination of SET and BY and use more than one by variable, you create more than one pair of first.<var> and last.<var> variables. Each variable listed in the BY statement results in a pair of first.<var> and last.<var> variables.

...Example 8.10...

```
data test;
input x y z;
datalines;                                                              1
1 1 100
1 1 110
1 1 120
2 1 10
3 1 99
1 2 200
1 2 210
3 2 199
1 3 300
1 3 310
;
run;
proc sort data=test;                                                    2
by x y;
run;

data first_last;
set test;                                                               3
by x y;
firstx = first.x;
firsty = first.y;                                                       4
lastx  = last.x;
lasty  = last.y;
run;

proc print data=first_last;
var x y z firstx lastx firsty lasty;
run;

Obs    x    y     z     firstx    lastx    firsty    lasty              5
  1    1    1    100       1        0         1         0
  2    1    1    110       0        0         0         0
  3    1    1    120       0        0         0         1
  4    1    2    200       0        0         1         0
  5    1    2    210       0        0         0         1
  6    1    3    300       0        0         1         0
  7    1    3    310       0        1         0         1
  8    2    1     10       1        1         1         1
  9    3    1     99       1        0         1         1
 10    3    2    199       0        1         1         1
```

1 A data set is created that will be used to show first.<var> and last.<var> values for two by variables, X and Y.
2 SET with a BY statement requires that data be sorted according to the BY variable(s).
3 A SET statement in combination with a BY statement is used to read the data set. There are two BY variables.
4 Since first.<var> and last.<var> variables are temporary (i.e. they only exist during the execution of the data step), new variables are created to hold the their values for subsequent printing.
5 The output from PROC PRINT shows the values of all the first.<var> and last.<var> variables.

If you look at the values of the variable X, you can see that the values of the first.X and last.X are what you have already seen in the previous examples. However, notice that the value of first.Y and last.Y cycle within each value of X.

(9) FUNCTIONS

There are a large number of SAS functions that allow you to work with data WITHIN a single observation. Remember that SAS PROCs work WITHIN variables/ACROSS observations. SAS functions work ACROSS variables/WITHIN observations. A number of SAS functions have already been introduced. In the section on DATES, several functions were shown that could be used to work with date (numeric) variables, e.g. MDY, WEEKDAY, TODAY, YRDFIF. One feature common to all SAS functions is that they are all followed by a set of parentheses that contain zero or more arguments. The arguments (sometimes referred to as parameters) are information needed by the function to produce a result. The TODAY function requires no argument, just the parentheses, to return the value of the current date. The WEEKDAY function requires one argument, a SAS date. The YEARDIF function requires three arguments. Just as SAS variables can be classed as either NUMERIC or CHARACTER, SAS functions can be divided into those that are used with numeric variables and those that are used with character variables. There are only a few functions that can be used with either type of data.

...NUMERIC FUNCTIONS

SAS numeric functions can be further divided into categories that describe the action of the function. The SAS language manual divides numeric functions into the following categories: arithmetic, date and time (dates are numeric variables), financial, mathematical (or 'more complicated arithmetic'), probability, quantile, random number, simple statistic, special, trigonometric and hyperbolic (no exaggeration), truncation. The following example uses grades for 29 students who took a series of exams.

...Example 9.1...

```
data midterm (drop=g1-g29);                                                    1
input type : $7. g1-g29;
sumgr  = sum(of g1-g29);
mingr  = min(of g1-g29);                                                       2
maxgr  = max(of g1-g29);
meangr = mean(of g1-g29);
medgr  = median(of g1-g29);
stdgr  = std(of g1-g29);
vargr  = var(of g1-g29);
missgr = nmiss(of g1-g29);
nmbgr  = n(of g1-g29);
meangr = round(meangr,.1);                                                     3
stdgr  = round(stdgr,.1);
vargr  = round(vargr,.1);
label
type   = 'TEST'
mingr  = 'MINIMUM'
maxgr  = 'MAXIMUM'
sumgr  = 'SUM'
medgr  = 'MEDIAN'
meangr = 'MEAN'
stdgr  = 'STANDARD DEVIATION'
vargr  = 'VARIANCE'
missgr = '# MISSING'
nmbgr  = '# NON-MISSING'
;
datalines;
QUIZ1    6  8  5 10 10  9 10  8  9  9  7 10 10 10 10  8 10  6 10 10 10 10 . 10  8 10 10 10 10
QUIZ2   10 10  2 10 10  . 10 10 10 10 10  2 10 10 10 10 10 10 10 10 10 10 . 10 10 10 10 10 10
MIDTERM  9  2  0  9  7 10 10 10  8  9  5  3  9  8  9 10  2 10  9  9  8 10 .  8  8 10  9 10  8
QUIZ3   10  2  2 18 19 12 20  . 10  8  .  2  . 10 10 18 12 10 15 10  5 15 .  5 12 20 20 20 20
QUIZ4   23  5  5 20 25 15 25 22 25 18 18  2 18 22 20 22 15 18 23 22 20 25 . 20 18 25 25 20 22
FINAL   25 20 15 25 25 22 25 25 22 18 21 15 22 20 18  . 20 15 23 24 22 20 . 15 24 22 25 20 20
;
run;

proc print data=midterm label noobs;
var type nmbgr missgr sumgr meangr stdgr vargr mingr maxgr medgr;
run;
```

TEST	# NON-MISSING	# MISSING	SUM	MEAN	STANDARD DEVIATION	VARIANCE	MINIMUM	MAXIMUM	MEDIAN
QUIZ1	28	1	253	9.0	1.5	2.2	5	10	10
QUIZ2	27	2	254	9.4	2.1	4.6	2	10	10
MIDTERM	28	1	219	7.8	2.8	7.8	0	10	9
QUIZ3	25	4	305	12.2	6.1	37.3	2	20	12
QUIZ4	28	1	538	19.2	6.1	37.6	2	25	20
FINAL	27	2	568	21.0	3.3	11.2	15	25	22

1 Since only the new values computed in the data step are to be placed in the data set, the values of the individual grades are dropped.
2 Several arithmetic and statistical functions are used to compute the values of new variables.
3 The ROUND function is used to round the value of three variables to one decimal place.

Naming the grade variables G1 through G29 made it very simple to place the values of the grades as arguments in the various functions. It might not be obvious, but each of the functions that used grades 1 through 29 ignored all grades with a missing value, just as would have been done if the data had been rearranged and analyzed with PROC MEANS. Both SAS functions and SAS procedures will ignore missing values when computing arithmetic or statistical values. The N (and NMISS) function allow you to determine how many values were used to compute various values. The output from PROC PRINT shows the mean, standard deviation, and variance of the grades for each question rounded to one decimal place. These values were stored in the data set in lieu of keeping values with many decimal places. If the ROUND function had not been used, but the following format placed in the data step or PROC PRINT...

```
format meangr stdgr vargr 6.1;
```

the same values would have been printed, but the stored values of the variables would still have many decimal places. The ROUND function actually changes the stored value while a FORMAT would only affect the appearance of the variable value.

There are several other functions that will alter the stored value of a numeric variable. They differ in the values that are returned for positive versus negative numbers.

...Example 9.2...

```
data alter;
input x;
ceil_x  = ceil(x);
floor_x = floor(x);
int_x   = int(x);
round_x = round(x,1.);
y = x;
format y 6.;
label
x       = 'ORIGINAL VALUE X'
y       = 'FORMATTED VALUE OF X'
ceil_x  = 'CEILING'
floor_x = 'FLOOR'
int_x   = 'INTEGER'
round_x = 'ROUND'
;
datalines;
7.5
-7.5
8.5
-8.5
;
run;
```

```
proc print data=alter noobs label;
var x y int_x round_x ceil_x floor_x;
run;

ORIGINAL    FORMATTED
 VALUE X    VALUE OF X    INTEGER    ROUND    CEILING    FLOOR
   7.5            8          7        8         8         7
  -7.5           -8         -7       -8        -7        -8
   8.5            9          8        9         9         8
  -8.5           -9         -8       -9        -8        -9
```

The results of the ceiling and floor functions depend on the whether the value of a variable is positive or negative. The integer and round functions work the same regardless of sign. The formatted value of the variable Y is the same as the rounded value, but Y is still stored as 7.5, -7.5, etc. since it is stored with the same values as X (remember, Y=X in the data step).

There are a number of ways of expressing the arguments required by functions that require the values of a number of numeric variables. In example 9.1, the convention of naming variables <name>1 through <name>N made it easy to place a large number of variable values as a function argument.

...Example 9.3...

```
data test;
input gradeone gradetwo gradethr;
mean_one = mean (of gradeone gradetwo gradethr);
mean_two = mean (of gradeone--gradethr);
datalines;
80 80 95
;
run;

proc print data=test noobs;
run;

gradeone    gradetwo    gradethr    mean_one    mean_two
   80          80          95          85          85
```

The name of each variable can be placed in the argument list. If you know the order of the variables in the data set, the convention of specifying a list with two variable names separated by two dashes (- -) can be used. Remember that using <start var>- -<end var> depends on the order of variables within the data set. If you are not sure of the order, you can see what is by using PROC CONTENTS and looking at the first column (showing the position of the variable in the data set.

As stated earlier, a function will ignore missing values. What if you use a function and the value of each variable in the argument list is missing? The value returned by the function will also be missing. To avoid this, you can specify the following...

```
sum_one = sum (of gradeone--gradethr, 0);
```

Adding a zero to the argument list ensures that the value returned by the function will never be missing, even if the value of each variable in the argument list is missing. Whether you want the function to result in missing or zero is up to you.