(1) INTRODUCTION

Once, the acronym SAS actually did stand for *Statistical Analysis System*. Now, when you use the term SAS, you are referring to a collection of integrated software products that you can use for data entry/management/analysis, report writing, statistical analysis, forecasting, graphics, mapping, etc. Some of these products are...

BASE	'basic product' - needed to run all other products - data analysis, report writing, some statistical analysis, some graphics, programming
STAT	advanced statistical analysis (regression, analysis of variance)
GRAPH	presentation quality graphics on just about any type of output device (laser printer, color printers)
ETS	time series
ETC.	'more than you'll ever need or use'

You are all probably familiar with word processing programs such as *Word* or *Word Perfect*. If so, you know that there are many features within such programs that you rarely if ever use. The main thing you do with a word processing package is create documents (letters, term papers). You do 95% of your work with about 5% of the features that are available, but it's nice to know that the other 95% of the software allows you to perform tasks such as generating a table of contents, automatically inserting footnotes, or creating mailing lists.

Unless working as a SAS consultant or teaching SAS becomes your career, you'll probably use a very small portion of all the available features in SAS. Just as with a word processing package, you'll most likely use 5% of the features to do a significant percentage of your data management, analysis, and reporting. However, given the breadth of tasks that can be accomplished with SAS and the size of the total package, mastering only that 5% might still seem daunting. If you encounter a bookcase full of SAS documentation, it's even hard to know where to start to find that important 5%. Hopefully, these notes will point you in the correct direction.

...SAS JOB (INSTREAM DATA/DATALINES STATEMENT)

Just to show you that, despite what you might have heard, getting work done with SAS is not that hard, here is a complete SAS job that will read some data, compute a few statistics, and print a report.

...Example 1.1...

```
* example of a SAS job that uses INSTREAM data;                                  1
data mystuff;                                                                    2
input                                                                            3
@01 lastname     $12.                                                            4
@13 chol           3.                                                            5
@16 sbp            3.
@19 dbp            3.
@22 dob       mmddyy8.                                                           6
@30 dod       mmddyy8.
;

aod = (dod - dob) / 365.25;                                                      7

label                                 /* a LABEL statement - more information about your data */     8
chol  = 'CHOLESTEROL'
sbp   = 'SYSTOLIC BLOOD PRESSURE'
dbp   = 'DIASTOLIC BLOOD PRESSURE'
dob   = 'DATE OF BIRTH'
dod   = 'DATE OF DEATH'
aod   = 'AGE AT DEATH'
;

format dob dod date9. aod 4.1;                                                   9
```

```
datalines;                                                                10
ADAMS       23016090 0411176702231848
FILLMORE    300156 880107180003081874
LINCOLN     222144 820212180904151865
JOHNSON     190    601229180807311875
TRUMAN      150140 700508188412261972
WASHINGTON  11015075 0222173212141799
CLINTON     20014008508191946
;
run;                                                                      11

options nocenter formdlim='-';                                            12

title "PROPERTIES OF SAS DATA SET MYSTUFF";                               13
proc contents data=mystuff;                                               14
run;                                                                      15

title "STATISTICS";
proc means data=mystuff;                                                  16
run;

title "ORIGINAL DATA";                                                    17
proc print data=mystuff label;
run;

title;                                                                    18
```

1 A COMMENT statement starts with an ASTERISK (*) and ends with a semicolon. Comments are used to document SAS jobs. All the text between the asterisk and semicolon is treated as documentation, not SAS code.

2 A DATA statement starts a SAS DATA STEP and the data step creates a SAS DATA SET named MYSTUFF.

3 An INPUT statement uses FORMATTED input to tell SAS the names of the variables to be associated with the values that are located in each line of data file. Formatted input uses INFORMATS to tell SAS how to read raw data. The @-sign followed by a number tells SAS what column to start in when reading the value of a variable.

4 The $12. is a CHARACTER INFORMAT and tells SAS to read twelve columns of data from the raw data file and to make LASTNAME a CHARACTER variable.

5 The 3. is a NUMERIC INFORMAT and tells SAS to read three columns of data from the raw data file and to make SBP a NUMERIC variable.

6 The MMDDYY8. is a DATE INFORMAT and tells SAS to read eight columns of data from the raw data file. The variable DOB is NUMERIC but is stored in a special way, i.e the number of days since January 1, 1960.

7 A calculation creates a new variable named AOD from two already existing variables.

8 A LABEL statement associates more information to variables. SAS will display the variable label in addition to or in place of the variable name in the output produced by SAS procedures.
There is also a second type of COMMENT statement. Notice that it is placed within another statement. This type of comment starts with /* and ends with */. It can occur anywhere within a SAS job

9 A FORMAT statement associates a display FORMAT with a variable. When this association is made within a data step, the association is permanent meaning that SAS will use this display rule each time it shows you the value of the given variable. Without this format, SAS would display values of the variable DOB as the number if days since 1/1/1960.

10 A DATALINES (you can also use the word CARDS) statement is used to include the data as part of the SAS data step (this is known as INSTREAM data). SAS treats all the lines after a DATALINES statement as data to be read by an INPUT statement within the data step. SAS recognizes either a semicolon (;) or a RUN statement (run;) as marking the end of the data file.

11 The RUN statement tells SAS where the data step ends. It is not always necessary to have a RUN statement at the end of a DATA STEP (or a SAS PROCEDURE), but for the rest of these notes, think of it as NECESSARY. Without the RUN statement, SAS knows to execute the DATA STEP as soon as it encounters another unit of work. In this example that unit is the PROC CONTENTS.

12 An OPTIONS statement changes some of the SAS system options. The options remain changed for the duration of your SAS session (until you exit SAS). The NOCENTER option tells SAS to LEFT JUSTIFY all text written to the OUTPUT window. The FORMDLIM option tells SAS to put a line of HYPHENS across the page instead of going to a new page in the OUTPUT window (conserves paper when you print the OUTPUT window contents). To RESET either or both of these options during a SAS session, use either or both of the following... `options center formdlim='';`
Notice that the OPTIONS statement does not occur within a data step or procedure. It is known as a GLOBAL statement.

13 A TITLE statement will write a title on the output. Once again, notice where it is, "floating" between the data step that was just completed and the procedure that follows. Very few statements are allowed to "float". Most statements must be part of a data step or a procedure

14 A SAS procedure, PROC CONTENTS, tells you the properties of the SAS data set MYSTUFF.

15 A RUN statement tells SAS to run PROC CONTENTS. Without the RUN statement, SAS would run PROC CONTENTS as soon as it encounters the statement PROC MEANS.

16 Another SAS procedure, PROC MEANS, computes several descriptive statistics. There are a number of options that can be used within PROC MEANS, but none are used in this example and you see the default output of the procedure.

17 PROC PRINT, a SAS procedure will display the values of all the variables in the data set MYSTUFF. Without a RUN statement after this procedure, SAS would not run the procedure since there are no more data steps or procedures in the SAS job.,

18 A TITLE statement with no title text clears all previous titles.

...SAS PROCEDURE OUTPUT

The results (output) from the three SAS procedures are as follows...

```
-----------------------------------------------------------------------------------------
PROPERTIES OF SAS DATA SET MYSTUFF

Data Set Name        WORK.MYSTUFF                      Observations          7
Member Type          DATA                              Variables             7
Engine               V9                                Indexes               0
Created              9:08 Thursday, January 22, 2004   Observation Length    64
Last Modified        9:08 Thursday, January 22, 2004   Deleted Observations  0
Protection                                             Compressed            NO
Data Set Type                                          Sorted                NO
Label
Data Representation  WINDOWS
Encoding             wlatin1  Western (Windows)


              Engine/Host Dependent Information

Data Set Page Size          8192
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            127
Obs in First Data Page      7
Number of Data Set Repairs  0
File Name                   d:\sas\work\_TD1980\mystuff.sas7bdat
Release Created             9.0000M0
Host Created                XP_PRO
```

```
           Alphabetic List of Variables and Attributes

#    Variable    Type    Len    Format    Label

7    aod         Num       8    4.1       AGE AT DEATH
2    chol        Num       8              CHOLESTEROL
4    dbp         Num       8              DIASTOLIC BLOOD PRESSURE
5    dob         Num       8    DATE9.    DATE OF BIRTH
6    dod         Num       8    DATE9.    DATE OF DEATH
1    lastname    Char     12
3    sbp         Num       8              SYSTOLIC BLOOD PRESSURE

-----------------------------------------------------------------------------------------

STATISTICS

Variable    Label                       N           Mean         Std Dev         Minimum         Maximum
-----------------------------------------------------------------------------------------------------------
chol        CHOLESTEROL                 7     200.2857143      60.6677132     110.0000000     300.0000000
sbp         SYSTOLIC BLOOD PRESSURE     6     148.3333333       8.4301048     140.0000000     160.0000000
dbp         DIASTOLIC BLOOD PRESSURE    7      78.5714286      10.8298529      60.0000000      90.0000000
dob         DATE OF BIRTH               7       -50688.86        26381.70        -83223.00        -4883.00
dod         DATE OF DEATH               6       -31889.67        20672.70        -58456.00         4743.00
aod         AGE AT DEATH                6      72.3709788      11.4597063      56.1697467      88.6324435
-----------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------

ORIGINAL DATA

                                      SYSTOLIC    DIASTOLIC
                                        BLOOD       BLOOD        DATE OF      DATE OF    AGE AT
Obs    lastname      CHOLESTEROL     PRESSURE    PRESSURE       BIRTH        DEATH     DEATH

 1     ADAMS             230           160          90       11APR1767    23FEB1848     80.9
 2     FILLMORE          300           156          88       07JAN1800    08MAR1874     74.2
 3     LINCOLN           222           144          82       12FEB1809    15APR1865     56.2
 4     JOHNSON           190             .          60       29DEC1808    31JUL1875     66.6
 5     TRUMAN            150           140          70       08MAY1884    26DEC1972     88.6
 6     WASHINGTON        110           150          75       22FEB1732    14DEC1799     67.8
 7     CLINTON           200           140          85       19AUG1946             .        .
```

The output from PROC CONTENTS tells you the properties of a SAS data set. It also tells you the attributes of variables within the data set. A few items you should notice in the output are: the ORDER of the variables in the data set as shown by the column labeled # (the variables are displayed in alphabetic order) is the order that SAS encounters the variables in the data step where the data set was created; the LENGTH of the variables (all NUMERIC variables have a length of 8, while the one character variable has a length of 12); though DOB was read with a date informat, it is a numeric variable; the format DATE9. is associated with variables DOB and DOD; the LABELS are displayed.

Though we did not specify what variables to analyze with PROC MEANS, the default behavior is to use all numeric variables in the data set, even the variables DOB and DOD. A mean, standard deviation, minimum, and maximum are computed for each variable. Look at the mean value computed for DOB. It was computed using the number of days since 1/1/1960 for each value of DOB. The column labeled N shows how many observations were used to calculate the statistics. Remember there was no data for SBP for Johnson. PROC MEANS ignores observations with missing data. Also, the variable AOD is computed using DOB and DOD. Since DOD is missing for Clinton, the value of AOD is also missing and only six observations are used to calculate statistics. Finally, there are more decimal places than you need (or are justified). PROC PRINT displays the variables with an observation number and the variables shown in the order seen in PROC CONTENTS.

...SAS LOG

The LOG file gives you information about how your SAS job executed. It always contains NOTES, and may also contain WARNINGS and ERROR messages The NOTES tell you what occurred when the job ran. In the LOG for the data step portion this example, you can see characteristics of the data set that is created, plus a message that tells you that a calculation resulted in a missing value

```
234  data mystuff;
235  input
236  @01 lastname     $12.
237  @13 chol           3.
238  @16 sbp            3.
239  @19 dbp            3.
240  @22 dob      mmddyy8.
241  @30 dod      mmddyy8.
242  ;
243
244  aod = (dod - dob) / 365.25;
245
246  label
247  chol  = 'CHOLESTEROL'
248  sbp   = 'SYSTOLIC BLOOD PRESSURE'
249  dbp   = 'DIASTOLIC BLOOD PRESSURE'
250  dob   = 'DATE OF BIRTH'
251  dod   = 'DATE OF DEATH'
252  aod   = 'AGE AT DEATH'
253  ;
254
255  format dob dod date9. aod 4.1;
256
257  datalines;

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 244:12
NOTE: The data set WORK.MYSTUFF has 7 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.07 seconds

265  ;
266  run;
267
268  options nocenter formdlim='-';
269
270  title "PROPERTIES OF SAS DATA SET MYSTUFF";
271  proc contents data=mystuff;
272  run;

NOTE: PROCEDURE CONTENTS used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

273
274  title "STATISTICS";
275  proc means data=mystuff;
276  run;

NOTE: There were 7 observations read from the data set WORK.MYSTUFF.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

277
278  title "ORIGINAL DATA";
279  proc print data=mystuff label;
280  run;

NOTE: There were 7 observations read from the data set WORK.MYSTUFF.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

281
282  title;
```

...SAS JOB (EXTERNAL DATA/INFILE STATEMENT)

What if the data to be used were located in a separate file rather than within the SAS job as INSTREAM data? In that case, you must use another SAS statement to tell SAS where the data are located. The only difference between the next example and example 1.1 is the use of a INFILE statement.

...Example 1.2...

```
data mystuff;
infile 'f:\sasclass\data\medical.dat';                                    1
input
@01 lastname     $12.
@13 chol           3.
@16 sbp            3.
@19 dbp            3.
@22 dob       mmddyy8.
@30 dod       mmddyy8.
;

aod = (dod - dob) / 365.25;

label
chol  = 'CHOLESTEROL'
sbp   = 'SYSTOLIC BLOOD PRESSURE'
dbp   = 'DIASTOLIC BLOOD PRESSURE'
dob   = 'DATE OF BIRTH'
dod   = 'DATE OF DEATH'
aod   = 'AGE AT DEATH'
;

format dob dod date9. aod 4.1;
run;
```

1 An INFILE statement tells SAS where the data to be read is located. If there is no INFILE statement in a data step and there is an INPUT statement, SAS expects to find INSTREAM data after a DATALINES statement. Once a INFILE statement is used, the INPUT statement will read raw data from an external file.

The LOG file now tells you that you have read data from an external file...

```
363  data mystuff;
364  infile 'f:\sasclass\data\medical.dat';
365  input
366  @01 lastname     $12.
367  @13 chol           3.
368  @16 sbp            3.
369  @19 dbp            3.
370  @22 dob       mmddyy8.
371  @30 dod       mmddyy8.
372  ;
373
374  aod = (dod - dob) / 365.25;
375
376  label
377  chol  = 'CHOLESTEROL'
378  sbp   = 'SYSTOLIC BLOOD PRESSURE'
379  dbp   = 'DIASTOLIC BLOOD PRESSURE'
380  dob   = 'DATE OF BIRTH'
381  dod   = 'DATE OF DEATH'
382  aod   = 'AGE AT DEATH'
383  ;
384
385  format dob dod date9. aod 4.1;
386  run;

NOTE: The infile 'f:\sasclass\data\medical.dat' is:
      File Name=f:\sasclass\data\medical.dat,
      RECFM=V,LRECL=256
NOTE: 7 records were read from the infile 'f:\sasclass\data\medical.dat'.
      The minimum record length was 37.
      The maximum record length was 37.
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 374:12
NOTE: The data set WORK.MYSTUFF has 7 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.08 seconds
      cpu time            0.07 seconds
```

(2) PROGRAMS, DATA SETS, RULES

...SAS PROGRAMS

For many of you, the only experience you have with working with data involves using some "point-and-click" interface, i.e. you are able to work with data by selecting analysis tools via drop-down menus and the click of a mouse button. SAS is different in that you write statements that instruct SAS how to read, analyze, and report data. This is not quite 100% true in that new features have been added to SAS that allow more of an interactive style of working with data. However, the statement is true enough in that it is how SAS will be explained in the remainder of these notes.

If you have some programming experience using other languages, you might have noticed in the first example that you do not have to write much code to get results (especially if your experience has been with a COBOL or a similar programming language). If you do not have any experience writing programs, just think of learning how to write SAS statements as being similar to learning how to express your ideas by writing in a foreign language, one where you have to be precise with the syntax and where the only really important punctuation mark is a semicolon.

A SAS program can be made up of just a data step, just a PROC (procedure), or a combination of a data steps and/or PROCs - plus GLOBAL statements, e.g. the OPTIONS and TITLE statements in example 1. The data step has a number of different functions. One of them is to convert raw data into a SAS data set. Data steps range from the very simple or to the very complex. The data step gives you the functionality of a programming language such as COBOL or BASIC.

SAS PROCs (short for procedures) are pre-written programs that work with data that are stored as a SAS data set. The short example program shown previously takes raw data, converts it to a SAS data set, and uses SAS PROCs to compute statistics and print the data.

...SAS DATA SETS

A SAS data set is data in a proprietary format (similar to storing data in a worksheet that is used by Lotus or Excel, or in a database that is used by dBASE, or even in a document that is used by Word) that can be "understood" by SAS. In addition to data, a SAS data set contains other information about the data such as the names of all the variables, variable types and lengths, the name of the data set, and the date on which it was created (look at the output from PROC CONTENTS in example 1.1). If you are familiar with spreadsheets, you'll know that you refer to data as being stored in ROWS and COLUMNS. Each row usually represents some person, place, or thing and each column usually represents some attribute of the data in each row. If you are familiar with databases, you'll know that you refer to data as being stored in RECORDS and FIELDS. Each record represents some person, place, or thing and each field represents some attribute of the data in each observation.

The analogous terms for SAS data sets are OBSERVATIONS (instead of rows or records) and VARIABLES (instead of columns or fields). SAS data sets are rectangular, i.e. each observation has the same number of variables. There may be observations where you do not have any data for a variable. You still must enter a value for this MISSING data, and this value is normally a period or a blank (remember the missing value in the example SAS job shown earlier).

At one time, SAS PROCs only would work with data in the form of a SAS data set. This is no longer true since SAS procedures can now use data stored in a number of different formats. You may be familiar with terms such as DDE (dynamic data exchange) or ODBC (open data base connectivity) that refer to data in a proprietary format for one piece of software being read by another. SAS also allows such behavior with SAS data base engines. For now, forget about using non-SAS data sets with SAS procedures and work under the assumption that if you want to use any SAS procedures, your data must be in the form of a SAS data set. That makes understanding the data step very important since it's the only way for you to take raw data and convert it into a SAS data set.

In example 1.1, the SAS data set name was one word, MYSTUFF. The real name for the data set was WORK.MYSTUFF, not just MYSTUFF as was specified. Once raw data are read and converted to a SAS data set, the data set has to be stored someplace. The SAS data set is stored in the WORK LIBRARY. All SAS data sets have names with two parts, i.e. <LIBRARY NAME>.<Data set NAME>. If you don't specify a library name when you create the data set, SAS assigns one and the default is WORK.

If you are using SAS on a PC, a LIBRARY is nothing more than a DIRECTORY (or if you prefer, a FOLDER). It could be on a diskette, on the PC hard disk, on a network disk, wherever. When you run a SAS job on a PC, the WORK library is in a location that was specified when SAS was installed. As soon as you exit SAS, all the files in the WORK library created during your SAS session are erased. SAS data sets written to the WORK library are TEMPORARY. There are many instances where you will want to save your data in a permanent (not forever, but for longer than on SAS session) SAS data set, To store a data set 'permanently' you must learn about another SAS statement, LIBNAME.

...Example 2.1...

```
*location of SAS data sets;
libname x 'f:\';                                                    1

data x.mystuff;                                                     2
infile 'f:\sasclass\data\medical.dat';
input                                                               3
lastname     $12.
chol           3.
sbp            3.
dbp            3.
dob       mmddyy8.
dod       mmddyy8.
;

aod = (dod - dob) / 365.25;

format dob dod date9. aod 4.1;
run;
```

1 A LIBNAME statement tells SAS the location of SAS data sets. After the word LIBNAME, SAS expects the name of a data library (it is your choice) and the name of an existing directory enclosed in quotes (either single or double). The name of the library X is referred to as a LIBREF (LIBRARY REFERENCE).

2 The LIBREF X is used as part of the data set name. Whenever a two-level data set name is used in a DATA statement,, the data set will be created in the directory associated with the given library name.

3 The INPUT statement is different than those used in example 1.1 and 1.2. The @<column> portion has been removed. This statement will still work and read the data correctly since the behavior of the INPUT statement involves the action of a 'pointer'. The INPUT statement always starts at column 1, unless there is an instruction to start in another column. After reading 12 columns of data (as instructed by the $12. informat), the 'pointer' will be in column 13 (just as it was when the INPUT statement included the @13 prior to the variable name CHOL). After reading data for variable CHOL, the 'pointer' will be in column 16, and so on.

The LOG produced by running this job will contain information about the library reference that was created by the LIBNAME statement. The NOTES contain information about the external file that was accessed, plus the data set name is no longer WORK.MYSTUFF. It is now X.MYSTUFF.

```
388  *location of SAS data sets;
389  libname x 'f:\';
NOTE: Libref X was successfully assigned as follows:
      Engine:        V9
      Physical Name: f:\
390
391  data x.mystuff;
392  infile 'f:\sasclass\data\medical.dat';
393  input
394  lastname     $12.
395  chol           3.
396  sbp            3.
397  dbp            3.
398  dob      mmddyy8.
399  dod      mmddyy8.
400  ;
401
402  aod = (dod - dob) / 365.25;
403
404  format dob dod date9. aod 4.1;
405  run;

NOTE: The infile 'f:\sasclass\data\medical.dat' is:
      File Name=f:\sasclass\data\medical.dat,
      RECFM=V,LRECL=256

NOTE: 7 records were read from the infile 'f:\sasclass\data\medical.dat'.
      The minimum record length was 37.
      The maximum record length was 37.
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 402:12
NOTE: The data set X.MYSTUFF has 7 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           7.11 seconds
      cpu time            0.06 seconds
```

...RULES

There are a number of rules that you'll have to remember if you want to run SAS programs. There are also a lot of rules that you don't have to remember (that's why we have SAS manuals). The following is a set of rules (borrowed from a SAS consultant) that you should try to remember since most are common to all the SAS code you will ever write. Yes, there are manuals (and on-line help), but you don't want to look up everything. Some of these rules have changed with the release of SAS Version 7.

- SAS statements begin with a KEYWORD and end with a semicolon. In the example SAS jobs seen thus far, some of the keywords are DATA, INPUT, DATALINES, LABEL, INFILE, LIBNAME.

- Both data set and variable names are limited to 32 characters (prior to V7, the limit was 8 characters).

- Both data set and variable names must start with a letter or an underscore. Subsequent characters may be letters, numbers, or underscores.

- There are only two types of SAS variables, CHARACTER and NUMERIC. Unless you explicitly state otherwise, variables are assumed to be NUMERIC.

- The default length of the content of all SAS variables is eight.

- The maximum length of the content of a SAS character variable is 32,000 (prior to V7, the limit was 200).

- A SAS System unit of work is terminated by a RUN statement or by another unit of work.

- Units of work are either data steps or PROCs, and a single SAS program usually contains more than one unit of work.

- SAS operates on units of work in the exact order that they appear in the program.
- A data step operates on one observation at a time.
- A PROC operates on all the observations in a data set.
- You can't use data step statements in a PROC and vice-versa.
- There are at least three ways to do any one thing with SAS - one wrong, and at least two right ways.
- ***VERY IMPORTANT RULE...*** For each of the above rules there exists at least one exception, knowledge of which is not required to learn SAS System fundamentals.

In addition to the rules, there are also some suggestions.

- SAS statements can be written in 'free format' since SAS recognizes the beginning of a statement via a keyword and keeps reading a statement until it encounters a semicolon. You can break SAS statements across lines, start in any column, use upper and/or lower case, embed blanks, and include blank lines. More than one SAS statement can be put on one line. This gives you all the flexibility to become a real 'sloppy' programmer. You should use the flexibility to develop a style, not to be sloppy.
- Avoid using SAS keywords (DATA, INPUT, etc.) as variable names.
- Just because you can use up to 32 characters to name variables does not mean that you should get into the habit of using long variable names. Use short variable names that make sense to you, that they remind you of the content of the variable. Then, use labels to show more about what the variable represents. Remember, if you use SBP to name a variable that represents systolic blood pressure, you can use SBP each time to refer to that variable. If you use SYSTOLIC_BLOOD_PRESSURE as the variable name, you will be doing a lot more typing each time you want to use that variable in a data step or procedure.

Try to keep the rules in mind as you look at SAS code since this will help you to commit them to memory.

(3) READING DATA

If you are a SAS user, it's easy to think of data as being in either of two formats: a SAS data set; not a SAS data set. As has been discussed in chapter two, a SAS data set is data in a proprietary format for use with SAS software. Most data you encounter will not be found in SAS data sets. Converting non-SAS data (referred to from now on as *raw data*) into a SAS data set is a fundamental task that's usually accomplished with a data step. SAS can handle raw data in just about any format. SAS is instructed to read raw data via an INPUT statement, and there are three types of raw data input: LIST, COLUMN, and FORMATTED.

...LIST INPUT

LIST input is the only way to read data in which the values of variables do not always occur in the same location across observations. LIST input depends on there being a delimiter separating the values of all the variables within each observation. There are many features associated with LIST input that allow you to read "unorganized" data. Having to learn the features of LIST input is the price you pay for having, for the want of a better term, sloppy data.

Delimiters (DSD & DLM Options)

The default delimiter for LIST input is a space. We can rewrite example 1 to require LIST input.

...Example 3.1...

```
*LIST input;
data mystuff;
informat lastname  $12. dob dod mmddyy8.;                                    1
input lastname chol sbp dbp dob dod;                                         2
aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;                                                                   3
ADAMS 230 160 90 04111767 02231848
FILLMORE 300 156 88 01071800 03081874
LINCOLN 222 144 82 02121809 04151865
JOHNSON 190 . 60 12291808 07311875
TRUMAN 150 140 70 05081884 12261972
WASHINGTON 110 150 75 02221732 12141799
CLINTON  200 140 085 08191946 .
;
run;
```

1 An INFORMAT statement is used to associate informats with variables. The informat $12. tells SAS to make LASTNAME a character variable with a length of 12 (not the default length of eight). The informat MMDDYY8. tells SAS to treat values of both DOB and DOD as dates.
2 An INPUT statement has no instructions other than the names of the variables. This is the most basic form of LIST input.
3 The data are not aligned in columns as in example 1. A space (the delimiter) separates the values of the variables.

One or more spaces can be used as delimiters. If you use PROC PRINT to display the values of the variables, what will the order of the columns (variables) be?

Another common delimiter is a comma...

...Example 3.2...

```
*LIST input with a comma delimiter;
data mystuff;
infile datalines dsd;                                                    1
input                                                                    2
lastname : $12.
chol
sbp
dbp
dob      : mmddyy8.
dod      : mmddyy8.
;
aod = (dod - dob) / 365.25;
format dob dod weekdate. aod 4.1;                                        3
datalines;
ADAMS,230,160,90,04111767,02231848
FILLMORE,300,156,88,01071800,03081874
LINCOLN,222,144,82,02121809,04151865
JOHNSON,190,,60,12291808,07311875
TRUMAN,150,140,70,05081884,12261972
WASHINGTON,110,150,75,02221732,12141799
CLINTON,200,140,085,08191946,
;
run;
```

1 An INFILE statement allows you to tell SAS that the data following the DATALINES statement later in the SAS job has values of variables separated by commas rather than spaces. The option DSD informs SAS that data are separated by commas and that two consecutive commas (in the observation for Johnson) indicate missing data.

2 The INPUT statement uses an alternative method of associating informats with variables. The colon (:) tells SAS to use the informat for the preceding variable.

3 A new FORMAT is used to display the variables DOB and DOD, WEEKDATE.

If you use a spreadsheet to enter and work with data, you probably have the option of saving your data with commas separating the values of variables. For example, if you use Excel, you can save a file with the extension .CSV (comma-separated-values) and then use your data as input in a SAS job. If a delimiter other than a space or comma is used, you can use a DLM option. The following statements show two alternate delimiters: a slash (/); a TAB (the hexadecimal value 09).

```
*tell SAS that a SLASH is used as the delimiter;
infile datalines dsd dlm="/";

*tell SAS that a TAB is used as the delimiter;
infile datalines dsd dlm="09"x;
```

In addition to showing how to read delimited data, the previous two examples have also shown two different methods of assigning attributes to data, i.e. an INFORMAT statement and COLON-MODIFIED input. Both of these methods result in the same attributes assigned to variables.

Missing Data (Missover Option)

In example 3.1, a period was used to tell SAS that there was no data for variable SBP in the fourth observation (JOHNSON) and no data for DOD in observation seven. What if the period had been left out of both observationsr? Here is the LOG file....

```
127  *LIST input;
128  data mystuff;
129  informat lastname $12. dob dod mmddyy8.;
130  input lastname chol sbp dbp dob dod;
131  aod = (dod - dob) / 365.25;
132  format dob dod date9. aod 4.1;
133  datalines;

NOTE: Invalid data for dod in line 138 1-6.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0
138        TRUMAN 150 140 70 05081884 12261972
lastname=JOHNSON dob=31JUL1875 dod=. chol=190 sbp=60 dbp=12291808 aod=. _ERROR_=1 _N_=4
NOTE: LOST CARD.
141        ;
lastname=CLINTON dob=19AUG1946 dod=. chol=200 sbp=140 dbp=85 aod=. _ERROR_=1 _N_=6
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
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 131:12
NOTE: The data set WORK.MYSTUFF has 5 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time           0.06 seconds
      cpu time            0.05 seconds
```

and here are the results of PROC PRINT...

Obs	lastname	dob	dod	chol	sbp	dbp	aod
1	ADAMS	11APR1767	23FEB1848	230	160	90	80.9
2	FILLMORE	07JAN1800	08MAR1874	300	156	88	74.2
3	LINCOLN	12FEB1809	15APR1865	222	144	82	56.2
4	JOHNSON	31JUL1875	.	190	60	12291808	.
5	WASHINGTON	22FEB1732	14DEC1799	110	150	75	67.8

What happened. The INPUT statement instructs SAS to look for the values of five variables within each line of data. If values cannot be found, the default behavior of a SAS INPUT statement is to move to the next line of data to look for a value to 'satisfy' data input, i.e. SAS must assign a value to each variable in the INPUT statement. Look at the LOG. It states that ...SAS went to a new line <more>..., meaning that there was not enough data to satisfy all variables in one (or more) of the data lines. Notice what else happened. Since SAS was looking for a value for the variable DOB and the first value it saw on the next line was TRUMAN, there is another message in the LOG, stating that there is ...INVALID DATA for variable DOB <more>. DOB is NUMERIC, the value TRUMAN is character, so the value of DOB for JOHNSON is set to MISSING (look at PROC PRINT results). What else occurred? There are only five observations. Each time the INPUT statement is encountered, the DEFAULT behavior of SAS is to move to the NEXT line in the data file, in this case it is the data for WASHINGTON and that is fine. There is no data for TRUMAN. What about CLINTON? After reading the data for WASHINGTON correctly, SAS moves to the next line of data. The INPUT statement says to read the values of six variables, but there are only 5 values within the data. Since there is not enough data for the INPUT statement, the LOST CARD message appears telling you that none of the data in the last observation (CARD) in your data has been added to the data set. Any idea why the message says LOST CARD?

There is an option to circumvent the DEFAULT of moving to a new data line when there is not enough data to satisfy the INPUT statement, i.e. MISSOVER. If an INFILE statement is added to example 1...

```
infile datalines missover;
```

the DEFAULT behavior is modified and SAS is told to set to MISSING the values of all variables that are not present in a given line of data, i.e. SAS cannot find values for all variables in an INPUT statement with a given line of data. SAS will not move to a new line of data to find values for variables. However,

this option can result in problems. To see the problems that result, add the missover option to example 1, remove the periods in the data file, and use PROC PRINT to print the data set. What is the new problem?

...Example 3.3...

```
*LIST input;
data mystuff;
infile datalines missover;
input
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy8.
dod       : mmddyy8.
;
aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;
ADAMS 230 160 90 04111767 02231848
FILLMORE 300 156 88 01071800 03081874
LINCOLN 222 144 82 02121809 04151865
JOHNSON 190  60 12291808 07311875
TRUMAN 150 140 70 05081884 12261972
WASHINGTON 110 150 75 02221732 12141799
CLINTON  200 140 085 08191946
;
run;

title 'USE MISSOVER ON INFILE STATEMENT - LOOK AT OBSERVATION #4';
proc print data=mystuff;
run;
```

```
EX3_03 - USE MISSOVER ON INFILE STATEMENT - LOOK AT OBSERVATION #4
Obs    lastname      chol    sbp          dbp          dob          dod     aod
 1     ADAMS          230    160           90    11APR1767    23FEB1848    80.9
 2     FILLMORE       300    156           88    07JAN1800    08MAR1874    74.2
 3     LINCOLN        222    144           82    12FEB1809    15APR1865    56.2
 4     JOHNSON        190     60     12291808    31JUL1875            .       .
 5     TRUMAN         150    140           70    08MAY1884    26DEC1972    88.6
 6     WASHINGTON     110    150           75    22FEB1732    14DEC1799    67.8
 7     CLINTON        200    140           85    19AUG1946            .       .
```

The MISSOVER option took care of the problem of SAS going to a new line of data when trying to read observation number four. However, since there was no data for SBP in observation four, the value intended for DBP was read as SBP, while the date of birth was read as DBP. Since MISSOVER prevented SAS from going to a new line to read the date of birth, variable DOB is assigned a value of missing. Though one problem was solved, another one was created (and there are NO MESSAGES in the LOG).

Modifiers (Colon and Ampersand)

There are a number of MODIFIERS that can be used with LIST input. The COLON modifier has already been shown in example 3.2....

```
input lastname : $12. chol sbp dbp dob : mmddyy8. dod : mmddyy8.;
```

The $12. is a SAS-supplied INFORMAT and tells SAS that you are reading CHARACTER data and want to allow up to twelve spaces to accommodate the values of the variable lastname. The COLON tells SAS that you want to associate the given informat ($12.) with the preceding variable. The data informat MMDDYY8. is associated with variables DOB and DOD.

What is the difference between the following statements....

```
input lastname : $12.;
```

and

```
input lastname $12.;
```

The first version, with the colon, tells SAS to read an observation starting in column one up to the occurrence of the first delimiter (space, comma, tab, whatever). The $12. after the colon tells SAS that the data are character and that the variable LASTNAME will have a length of 12. The second version without the colon also tells SAS to start reading an observation in column one. However, it also tells SAS to read the first twelve characters in that observation, regardless of the occurrence of any delimiter.

Another useful modifier with list input is the AMPERSAND (&). It tells SAS that at least two spaces are needed to separate the values of variables. What if the data in the example 2 included a first and last name to be read as one variable NAME...

...Example 3.4...

```
*LIST input with an ampersand (&) modifier;
data mystuff;
input
lastname & : $20.                                                    1
chol
sbp
dbp
(dob dod) (: mmddyy8.)                                               2
;
aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;
JOHN QUINCY ADAMS   230 160 90 04111767 02231848                     3
MILLARD FILLMORE  300 156 88 01071800 03081874
ABRAHAM LINCOLN   222 144 82 02121809 04151865
ANDREW JOHNSON  190 .  60 12291808 07311875
HARRY TRUMAN  150 140 70 05081884 12261972
GEORGE WASHINGTON  110 150 75 02221732 12141799
BILL CLINTON   200 140 085 08191946 .
;
run;
```

1 The ampersand (&) tells SAS to ignore the space embedded within the variable name and look for at least two spaces before recognizing the next variable (chol) in the list. The colon followed by $20. tells SAS to allow up to 20 spaces to store the name, while the variables DOB and DOD are treated as a dates by using a colon followed by MMDDYY8. Since both dates share the same informat, the variable names are put in parentheses and they are followed by the informat in parentheses. The colon (:) is still needed within the parentheses to tell SAS that LIST input is being used.
2 Since DOB and DOD share the same informat (MMDDYY8.), an INFORMAT LIST can be used to read the values of the two variables.
3 Two spaces follow each name to tell SAS to where the value of the next variable (CHOL) begins.

Informat Statement

In example 3.1, informats were associated with variables using an INFORMAT statement rather than with the colon (:) modifier in the INPUT statement (example 3.2). Even with an informat statement, the ampersand modifier is still necessary if the variable NAME contains embedded blanks and is being read by list input.

...Example 3.5...

```
*LIST input with an INFORMAT statement and & modifier;
data mystuff;
informat lastname $20. dob dod mmddyy8.;                             1
input lastname & chol sbp dbp dob dod;                               2
format dob dod date9.;
```

```
datalines;
JOHN QUINCY ADAMS   230 160 90 04111767 02231848
MILLARD FILLMORE  300 156 88 01071800 03081874
ABRAHAM LINCOLN   222 144 82 02121809 04151865
ANDREW JOHNSON  190 .  60 12291808 07311875
HARRY TRUMAN  150 140 70 05081884 12261972
GEORGE WASHINGTON  110 150 75 02221732 12141799
BILL CLINTON   200 140 085 08191946 .
;
run;
```

1 An INFORMAT statement associates type and length with variables. The informat $20. is associated with the variable LASTNAME, while the informat MMDDYY8. is associated with both DOB and DOD.
2 The & is still needed due to the embedded blanks in the names.

Length Statement

Both the colon modifier and the informat statement told SAS that character variables were not the standard length, i.e. the content of variables was longer than eight characters. There is yet another way to associate a length with a variable, a LENGTH statement.

...Example 3.6...

```
*use of a LENGTH statement instead of an INFORMAT;
data mystuff;
length lastname $12;                                    1
input
lastname
chol
sbp
dbp
(dob dod) (: mmddyy8.)
;                                                       2
datalines;
ADAMS   230 160 90 04111767 02231848
FILLMORE  300 156 88 01071800 03081874
LINCOLN   222 144 82 02121809 04151865
JOHNSON  190 .  60 12291808 07311875
TRUMAN  150 140 70 05081884 12261972
WASHINGTON  110 150 75 02221732 12141799
CLINTON   200 140 085 08191946 .
run;
```

1 A LENGTH statement tells SAS that lastname is a character variable whose content can be up to twelve characters
2 No informat is needed to tell SAS that name is a character variable (that has been taken care of in the LENGTH statement)

...COLUMN INPUT

Data read via LIST input has little structure in that values of variables are not constrained to fall in the same columns across all observations. LIST input uses delimiters to recognize the location of the values for variables. If your data are a little more orderly, i.e. values for variables always fall in the same columns across observations, you can use COLUMN input.

...Example 3.7...

```
*COLUMN input;
data mystuff;
input
name $ 1-17
chol   18-20
sbp    21-23
dbp    24-25
;                                                       1
```

```
datalines;                                                                2
JOHN QUINCY ADAMS23016090
MILLARD FILLMORE 30015688
ABRAHAM LINCOLN  22214482
ANDREW JOHNSON   190   60                                                 3
HARRY TRUMAN     15014070
GEORGE WASHINGTON11015075
BILL CLINTON     20014085
;
run;
```

1 SAS is instructed to read variables via column locations. A starting and ending column are specified. The $ is still need for the variable NAME since it contains non-numeric data
2 No spaces (delimiters) are needed to separate the values of the variables.
3 The data for SBP in the fourth observation are BLANK to indicate missing data. Since column input is being used, we no longer have to specify missing data with a period as was done with list input (the period and the space delimiter were needed with LIST input since without it, SAS would have just read the next piece of data on the line as a value for SBP).

The variables DOB and DOD have been left out of this example. ***There is no way to read a variable as a date using column input.*** Only LIST input with an informat or FORMATTED input can be used to read dates.

Notice that there is no LENGTH statement or INFORMAT statement. All the information needed to tell SAS about variable length and type is in the INPUT statement. The LENGTH of the character variable is determined by the column numbers. All the numeric variables still have a length of eight.

...FORMATTED INPUT

An alternative to using COLUMN input for the "orderly" type of data shown in example 3.7 is FORMATTED input. The following shows a variation on the INPUT statement shown in example 3.1. The

...Example 3.8...

```
*FORMATTED input;
data mystuff;
input                                                                     1
lastname  $12.
chol        3.
sbp         3.
dbp         3.
dob   mmddyy8.
dod   mmddyy8.
;
aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;
ADAMS       23016090 0411176702231848
FILLMORE    300156 880107180003081874
LINCOLN     222144 820212180904151865
JOHNSON     190    601229180807311875
TRUMAN      150140 700508188412261972
WASHINGTON  11015075 0222173212141799
CLINTON     20014008508191946
;
run;
```

1 Informats are used as in example 1 but no column pointers (@s) are used. When SAS reads across a line of raw data, the column pointer starts in column 1 by default. After reading twelve columns of data for variable LASTNAME, the column pointer will be in column 13. After reading three columns of data for variable CHOL, the column pointer will be in the correct position to read variable SBP. The same is true for the rest of the variables.

Informats

There are many SAS-supplied informats that allow you to read data that are stored in different ways. You can also write your own informats to read data (more about that later when we look at PROC FORMAT). In the last example, we are reading two different types of data, character and numeric. The informat $12. tells SAS that there are twelve columns of data to be stored as a character variable. The 3. informat tell SAS that you are reading data that are to be stored as numeric variables and that they are located in three columns. The length of the character variable will be 12. What are the lengths of the numeric variables? The answer is eight.

...Character Data...

There are a few SAS-supplied informats that you will use repeatedly. The two informats commonly used to read characters data are $<w.> and $char<w.>, where the <w.> indicates a width that you provide (a minimum of 1 and a maximum, for now, of 200). In most instances, reading character data with either of these two formats has the same results. However, there are two instances that you should know about where they differ. If the value of a character variable has leading blanks, the $<w.> informat will get rid of them and left justify the data, while the $char<w.> will not modify the data, leaving the leading blanks. The other instance where these two formats differ is when you use a period to indicate missing character data. The $<w.> informat will read the period as indicating a missing value, while the $char<w.> informat will read the period as representing real data value.

...Example 3.9...

```
* difference between $ and $char informat;
data characters;
input                                                                    1
@1 a $char10.
@1 b $10.
;
datalines;                                                               2
MIKE ZDEB
.
     ZDEB
;
run;

title 'CHARACTER INFORMATS';
proc print data=characters;
run;
```

```
CHARACTER INFORMATS
Obs        a          b
 1     MIKE ZDEB    MIKE ZDEB
 2     .
 3          ZDEB    ZDEB
```

1 The same data are read twice, in two different ways (two different informats). The @1 tells SAS to start in column one of the data file in reading data for each variable (the same data are read for variables A, B, and C) and the various INFORMATS tell SAS to read ten columns.
2 The data shows values that will be treated differently by the two different character informats.

Notice in the printed results the variable has values that look exactly the same as those that occur in the DATALINES section of example 3.9. The values for variable b are different. The PERIOD in the second line of the DATALINES file has been converted to a blank, and the value of the third observation has been left justified (leading blanks have been removed). Neither of these character informats is inherently the correct one to use. That depends on how you want your raw data treated as you assign values to variables in your data set.

...Numeric Data...

When reading numeric data, the <w>. format will suffice in most instances. However, if you have numeric data containing anything other than numbers, a decimal point, or a minus sign, you can't read that data with just the <w.> informat. You must use the comma<w.> informat.

...Example 3.10...

```
*COMMA informat;
data numbers;
input                                                                    1
@1 a $8.
@1 b 8.
@1 c comma8.
;
datalines;                                                               2
1000
 1000
  1000
   1000
1,000
$1000
1000.
-1000
(1000)
10-00
10-00-00
10 00
1  00
;
run;

title 'COMMA INFORMAT';
proc print data=numbers;
run;
```

```
COMMA INFORMAT
Obs    a             b            c
  1    1000          1000        1000
  2    1000          1000        1000
  3    1000          1000        1000
  4    1000          1000        1000
  5    1,000            .        1000
  6    $1000            .        1000
  7    1000.         1000        1000
  8    -1000        -1000       -1000
  9    (1000)           .       -1000
 10    10-00            .        1000
 11    10-00-00         .      100000
 12    10 00            .        1000
 13    1  00            .         100
```

1 The same data are read three times, in three different ways (three different informats). The @1 tells SAS to start in column one of the data file in reading data for each variable (the same data are read for variables A, B, and C) and the various INFORMATS tell SAS to read eight columns.

2 The data comprise a combination of different ways of representing the number one-thousand.

This comma informat has some "smarts" in that it knows that you want the values being read to be stored as numeric data and will strip away (or convert) anything other than numbers prior to storing the values. Any characters other than numbers, decimal points, or minus signs cause the numeric informat to result in a missing value. What about observations 13 and 14? Is the conversion correct (variable c) or should there be zeroes, not blanks, in the values (should both be 10000)? Notice that SAS does not care where in the specified column range the number is located - left-justified, right-justified, centered, whatever. SAS will find the number anyplace within the range specified and store the number as it appears in your data.

One last item about numeric informats contrasts actual and implied decimal points in your data. There are occasions when numeric values in your data will contain a decimal point. However, it is not uncommon to have an implied decimal point. One example of this is values a numeric data that represent dollars and cents. A value of 100012 stored as raw data might actually represent $1000.12. If your task is to read that data and calculate a mean value, unless you read the data correctly (insert a decimal point with an informat), your calculated values will be incorrect.

...Example 3.11...

```
data decimals;
input                                                                    1
@1 a $char10.
@1 b 10.
@1 c 10.2
@1 d 10.4
;
datalines;                                                               2
100012
1000.12
1000.123
;
run;

title 'REAL AND IMPLIED DECIMAL POINTS';
proc print data=decimals;
run;

title 'REAL AND IMPLIED DECIMAL POINTS - SHOW ALL THE DECIMAL PLACES';
proc print data=decimals;
format b c d 15.5;                                                       3
run;
```

```
REAL AND IMPLIED DECIMAL POINTS
Obs       a            b           c          d
 1     100012      100012.00    1000.12      10.00
 2     1000.12       1000.12    1000.12    1000.12
 3     1000.123      1000.12    1000.12    1000.12


REAL AND IMPLIED DECIMAL POINTS
Obs       a                       b                  c                  d
 1     100012          100012.00000          1000.12000           10.00120
 2     1000.12           1000.12000          1000.12000          1000.12000
 3     1000.123          1000.12300          1000.12300          1000.12300
```

1 The same data are read four, in four different ways (three different informats). The @1 tells SAS to start in column one of the data file in reading data for each variable (the same data are read for variables A, B, C, and D) and the various INFORMATS tell SAS to read ten columns.
2 The data contain a mix of implied and real decimal places.
3 A FORMAT is used to display each number with five decimal places.

Notice that the INFORMAT 10. left the data just as it appeared in the DATALINES file. The 10.2 INFORMAT added two decimal places to the number that had no decimal places (10012), but had no effect on the numbers that already had a real decimal point. The same behavior is true for the 10.4 INFORMAT. If you print that data, you might think that SAS has dropped some of the decimal values. However, PROC PRINT makes some decisions as to how you might want the output to look. You can override the appearance of the numeric data in the output by adding a FORMAT statement.

Informat Lists

There are cases when you have data where the same format applies to a number of different variables that happen to fall one-after-another in an observation. For example, what if you have a patient record with an ID, up to five diagnoses, and a length of stay. You decide to use formatted input to read the data.

...Example 3.12...

```
data hosp0001;
input @01 pat_id   $9.                                                    1
      @10 dx1      $5.
      @15 dx2      $5.
      @20 dx3      $5.
      @25 dx4      $5.
      @30 dx5      $5.
      @35 los      4.
;
datalines;
1303476542848 2913               0010
000231234292822961030301309213150200 35
123456789V31                     0002
;
run;
```

1 Formatted input is used to read the raw data. The @-sign is use to direct SAS to a specific column in the raw data. An informat is specified for each variable, six character variables and one numeric.

There is an easier way to read these data, with an informat list.

...Example 3.13..

```
data hosp001;
input pat_id     $9.                                                      1
      (dx1-dx5) ($5.)
      los         4.
;
datalines;
1303476542848 2913               0010
000231234292822961030301309213150200 35
123456789V31                     0002
;
run;
```

1 Formatted input is used to read the raw data. Notice that the column pointer @01 is not really necessary since SAS starts reading data in column one by default. After reading the patient ID (pat_id), the column pointer is in column 6, the start of the diagnosis data. An informat list is used to read the diagnoses. The informat $5. applies to each of the five diagnoses dx1 through dx5. After reading the diagnoses, the column pointer is at column 35, the correct location for reading the length of stay (los)

You can modify the input statement in example 10 by putting all the variables and all the informats in parentheses.

```
input (pat_id dx1-dx5 los) ($9. 5*$5. 4.);
```

In an input statement, if one or more variables are listed within a set of parentheses, SAS expects another set of parentheses that contain the informats used to read the data. There should be an informat for each variable. In the input statement shown above, the $9. informat is used to read the patient id. The 5*$5. informat tells SAS to repeat using the $5. informat five times, once for each diagnosis. The numeric informat 4. is used to read length of stay.

What if you were only interested in reading the first three characters of each diagnosis (major diagnostic subgroups rather than all the detail conveyed by the full five digits)? No problem since in addition to the absolute column pointer (the @), SAS also has a relative column pointer (the +). The relative column pointer tells SAS to move a specified number of columns from its current position on the line of data currently being read. The extra +2 is needed prior to reading the variable LOS since once SAS has read variable DX5 with an informat of $3., it does not move the column pointer 2 spaces as it did after reading DX1 through DX4.

```
input @01 pat_id     $9.
      @10 (dx1-dx5) ($3. +2)
      @35 los         4.
;
```

Each diagnosis would be stored with a length of three rather than five. Informat lists are one of many instances of writing SAS code that can be considered a "shortcut", i.e. there are other ways to accomplish the same task with SAS code, but the informat list requires fewer keystrokes. One of the tasks in learning how to use SAS is to determine when the "shortcut" might be complicated enough to make the "more keystroke" solution the preferred path.

...VARIATIONS ON THE INPUT STATEMENT

In the examples up to now, all the raw data to be converted into a SAS data set have been presented in the same manner. There has been one record per observation, and all the data for variables to be read by an input statement have been located with one record. Each record in the raw data files have always contained the same amount of information, i.e. data for each of the variables mentioned in the input statement. There are methods of reading raw data files that do not have these characteristics.

Data for One Observation Stored in More than One Record (Slash and Number Sign)

If the raw data to be read by an input statement are present on more than one record, the input statement must be instructed to move to a new record to complete reading the data for all the variables mentioned in the input statement. There are two ways of instructing an input statement to move to a new record.

...Example 3.14...

```
data new;
input id      $ 1-5                                                    1
      gender  $ 6 /
      grade1-grade5;
datalines;                                                             2
12345M
100 98 78 99 100
45678F
75 64 76 80 100
99999M
100 100 99 100 100
;
run;
```

1 A slash (/) is used in the input statement to instruct SAS to move to a new line in the raw data file. Column input is used to read the first two variables, while list input is used to read variables grade1 through grade5.
2 The data intended for each observation are spread across two records in the raw data file.

The other method of moving to a new line of data is to use a number sign (#), also known as a line pointer. The input statement in example 12 would be changed to look as follows:

```
input id      $ 1-5
      gender  $ 6
#2    grade1-grade5;
```

Rather than using a slash, the #2 instructs SAS to move to a second record in the raw data file. If data were present on three records, another slash followed by more variable names could be used. A number sign followed by a new record number, #3, could also be used.

The slash can also be used to skip records in a raw data file. The next example shows an instream file containing a county number, a gender, and populations for residents in six different age groups. Only the raw data for males are to be read into data set pop_m.

...Example 3.15...

```
data pop_m;
input county $3. +1 pop1-pop6 /;                                                    1
datalines;                                                                          2
001M   1862   2015   2000   2050   2162   9943
001F   1820   1821   1954   2024   2129   9682
003M    344    342    396    392    430   2011
003F    331    322    350    364    386   1901
005M  11615  11587  11800  11475  11566  51125
005F  11234  11132  11208  11091  11458  48926
;
run;
```

1 A slash is used at the end of the input statement to skip a line of raw data after reading values for the variables county and pop1-pop6. A +1 is used to skip the gender since you know that the data are for males only.
2 The raw data file contains a line of data for males and another for females.

Different Data Across Records (At Sign)

An input statement instructs SAS to read data in a raw data file in a specific manner. If you have one input statement in a data step, you assume that the data for each observation are present in the same manner across all records in your raw data file. If the location of data changes across records, you may need more than one input statement. In the following example, the data for males and females have been entered in different orders.

...Example 3.16...

```
data m_f;
input gender $1. @;                                                                 1
if gender eq 'M' then input (age ldl hdl) (3.);                                     2
else                  input (age hdl ldl) (3.);
pct_hdl = 100 * hdl / (ldl + hdl);                                                  3
datalines;
M 25120 76
F 33 80200
M 50 70 80
;
run;
```

1 An at-sign (@) is used at the end of the input statement to tell SAS to remain on the current line of raw data when the next input statement is encountered. The normal behavior of SAS is to move to a new line of raw data each time an input statement is encountered.
2 The gender read in the first input statement is used to tell SAS how to read the remaining data in the file of raw data. For males, LDL occurs prior to HDL. For females, HDL occurs prior to LDL.
3 An equation is used to calculate the percentage of total cholesterol that is HDL.

Data for More than One Observation in One Record (Double At Sign)

Raw data files sometimes have data for more than on observation present in a single record. The normal behavior of the input statement is to move to a new line of raw data. Also, the normal behavior of a data step is to release any line of raw data held by an @-sign (as was done in example 14) once the end of the data step is reached. In the next example, there are data for multiple observations within each line of raw data.

...Example 3.17...

```
data many_obs;
input id : $5. age hr @@;                                                           1
datalines;                                                                          2
12345 34 76 23456 10 55
9999 100 76 987 5 80 12121 54
80 13131 65 65
;
```

```
run;
```

1 Two @-signs are used to give SAS two instructions. The first is to leave the line pointer in the record just read when the next input statement is encountered. The second instruction is to hold the location even when the end of the data step is encountered.
2 The raw data file has data for individual observations stored in various manners, both within and across records.

In the last examples, the location of data changed across records. It is also possible to have different data in records in on file. In either case, you can read the file properly if you understand the role of slashes, number signs, line pointers, the @-sign, and the @@-sign.

...EXTERNAL DATA

Thus far, most of the examples have shown data read by SAS included as instream data, i.e. located within the data step via a DATALINES statement. Though there is no limit on the number observations you can read as instream data, you will at some time want read raw data (or SAS data sets) that are located in an external file, e.g. a file on a diskette.

Infile Statement

You must tell SAS where data in external files are located via an INFILE statement. An INFILE statement can be used with either INSTREAM or EXTERNAL data. In example 3.2, an INFILE statement was used to tell SAS that instream data contained comma delimiters. The statement used was...

```
infile datalines dsd;
```

When SAS saw the word DATALINES in the INFILE statement, it knew to look for data within the data step following a DATALINES statement. If raw data are in an external file, you can replace the word DATALINES with the name of the external file...

```
infile "a:\diabetes.dat";
```

This statement tells SAS to look for your data in a file named *diabetes.dat* on a diskette in the a-drive of your PC. There are options that can be used after the name of the external data file. One very important option is LRECL. Unless you use an LRECL option on the INFILE statement, PC SAS will only read up to 256 characters from any given observation. If the data file you are using has records longer than 256 characters, you must tell SAS to read beyond the default 256 character boundary...

...Example 3.18...

```
data mystuff;
infile 'e:\amc99.dat' lrecl=445;                                    1
input @349 los   4.;
label los = 'LENGTH OF STAY';
run;
```

1 The LRECL option is used to tell SAS to expect records that are 445 characters long.

The LOG file shows that SAS read the file correctly....

```
77    data mystuff;
78    infile 'e:\amc99.dat' lrecl=445;
79    input @349 los   4.;
80    label los = 'LENGTH OF STAY';
81    run;
NOTE: The infile 'e:\amc99.dat' is:
      File Name=e:\amc99.dat,
      RECFM=V,LRECL=445
```

```
NOTE: 941 records were read from the infile 'e:\amc99.dat'.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.MYSTUFF has 941 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.11 seconds
```

What if the LRECL option had been left off the INFILE statement? Here's is the LOG file...

```
83    data mystuff;
84    infile 'e:\amc99.dat';
85    input @349 los   4.;
86    label los = 'LENGTH OF STAY';
87    run;
NOTE: The infile 'e:\amc99.dat' is:
      File Name=e:\amc99.dat,
      RECFM=V,LRECL=256

NOTE: 941 records were read from the infile 'e:\amc99.dat'.
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.MYSTUFF has 470 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.04 seconds
```

SAS used the default maximum record length of 256 characters when reading the file. Since the INPUT statement specifies to read data at column 349, SAS went to the next record to find the data. This is a good lesson that shows how important it is to look for messages in the SAS LOG. There are no ERRORS and you have a SAS data set. However, it's not the data set you really want to use.

The LRECL option just used above statement tells SAS to expect record that contain 445 characters. If you are not sure of the record length, but know that it is longer than 256 characters, you can add a PAD option to the INFILE statement...

```
infile "e:\amc99.dat' lrecl=500 pad;
```

Here's the LOG file showing what happens if the LRECL + PAD option is used to read the file AMC99.DAT...

```
89    data mystuff;
90    infile 'e:\amc99.dat' lrecl=500 pad;
91    input @349 los   4.;
92    label los = 'LENGTH OF STAY';
93    run;

NOTE: The infile 'e:\amc99.dat' is:
      File Name=e:\amc99.dat,
      RECFM=V,LRECL=500

NOTE: 941 records were read from the infile 'e:\amc99.dat'.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.MYSTUFF has 941 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.11 seconds
```

The data are read correctly. The PAD option when used with an LRECL option tells SAS to expect records up to length specified. If any records in the input data are shorter than the specified value, the PAD option prevents SAS from going to the next record in the external file to find more data.

Filename Statement (Fileref)

An alternative to putting the name of the file directly in the INFILE statement is to use a combination of FILENAME and INFILE statements.

...Example 3.19...

```
filename mydata "a:\diabetes.dat";                                    1
data mystuff;
infile mydata;                                                        2
input id : $3. x1-x6;
run;
```

1 The FILENAME statement establishes a *fileref* (file reference), in this example it is *mydata* - from then on anytime SAS sees the fileref, it knows that it refers to the external file associated with the *fileref* in the FILENAME statement, i.e. *diabetes.dat*
2 The INFILE statement uses the fileref instead of the full name of the external file and SAS now knows to look for data in the external file *diabetes.dat*

The rules for the naming *filerefs* are the same as for naming SAS variables and data sets (remember the magic number, 8). Notice that the FILENAME statement is not located within a data or PROC step. It is a GLOBAL statement. There is no particular advantage in using a FILENAME statement in the last example, in fact it violates the rule of "...the less typing, the better..." in that there are more keystrokes needed.

There are instances where you might want to set up *filerefs* at the top of your SAS job rather than embed the full file names within the data step. One such instance might be if you are writing a SAS job that you will use with numerous different data files. It is easier to alter one or more FILENAME statements at the top of the SAS job than it is to hunt through the SAS code for all the INFILE statements and change the names of the external files.

Another advantage of the FILENAME statement is that it allows you to read multiple files within a data step. You can specify more than one external file within a FILENAME statement, but you can only specify one filename in an INFILE statement. For example, if you had three files containing hospital-based data and wanted to use an INFILE statement within a data step to tell SAS to read all three of them, you could use...

...Example 3.20...

```
filename allthree ("e:\amc97.dat" "e:\amc98.dat" "e:\amc99.dat");      1
data amc97_99;
infile allthree lrecl=500 pad;                                        2
input
@027 dischyr   $4.
@349 los        4.
;
run;
```

1 A FILENAME statement is used to create the FILEREF ALLTHREE. ALLTHREE refers to three files.
2 An INFILE statement uses the fileref and SAS reads all three files.

Here is the LOG...

```
104   filename allthree ("e:\amc97.dat" "e:\amc98.dat" "e:\amc99.dat");
105   data amc97_99;
106   infile allthree lrecl=500 pad;
107   input
108   @027 dischyr   $4.
109   @349 los        4.
110   ;
111   run;
```

```
NOTE: The infile ALLTHREE is:
      File Name=e:\amc97.dat,
      File List=('e:\amc97.dat' 'e:\amc98.dat' 'e:\amc99.dat'),
      RECFM=V,LRECL=500

NOTE: The infile ALLTHREE is:
      File Name=e:\amc98.dat,
      File List=('e:\amc97.dat' 'e:\amc98.dat' 'e:\amc99.dat'),
      RECFM=V,LRECL=500

NOTE: The infile ALLTHREE is:
      File Name=e:\amc99.dat,
      File List=('e:\amc97.dat' 'e:\amc98.dat' 'e:\amc99.dat'),
      RECFM=V,LRECL=500

NOTE: 1330 records were read from the infile ALLTHREE.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: 1320 records were read from the infile ALLTHREE.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: 941 records were read from the infile ALLTHREE.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.AMC97_99 has 3591 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.17 seconds
```

All three files were read successfully and the data set AMC97_99 contains observations from all three external files.

There are other ways to read multiple files (using INFILE options or FILEVARs), but this method is the easiest to explain (and understand).

...SAS DATA SETS

Once raw data have been converted to a SAS data set, there is no longer any need for either an INFILE or INPUT statement. INFILE statements are used to point to the location of a raw data file, not SAS data sets. INPUT statements are used to specify the manner in which raw data are to be read from a file (either instream data, i.e. a datalines file, or an external file) and converted to variables in a SAS data set. In reading raw data and creating a SAS data set, you make decisions that govern the attributes of the variables the end up in the data set: name, type, length, label, format, sequence. Once you have made all these decisions and created a data set, you do not have to go through that process again unless you want to change one or more of the attributes of one or more of the variables in your data set.

Once your data are in SAS data set, what you have to know are the data set name and where the data set is located. If you are using a temporary data set, i.e. on in the WORK library, knowing the name is enough. If you are using a permanent data set, i.e. one you have created specifying a LIBNAME other than work, you have to know where the data set was stored, Look at example 1.1 and make two changes. The first is that you create a permanent data set by using a combination of a LIBNAME statement and a two-level data set name...

```
libname perm 'i:\';
data perm.mystuff;
<rest of data step>
```

The other change is that you forget to add the statement that calculates the ratio of systolic to diastolic blood pressure. Your situation is that you have a SAS data set. You've made all those decisions about how to read the raw data and what attributes the variables should have. All you want to do is add another variable. There are two choices: rerun the data step that reads the raw data and add the statement that computes the ration; read the SAS data set and add a variable to the data set. With a

small amount of raw data, choosing to rerun the data step does not take much time. However, what if you were working with a large amount of data. Or what if you no longer had the raw data or never had the raw data and were given a SAS data set to use?

...Example 3.21...

```
libname perm 'i:\';                                                    1
data perm.mystuff;                                                     2
set perm.mystuff;                                                      3
ratio = sbp /dbp;                                                      4
label ratio = 'SBP/DBP';                                               5
run;
```

1 A LIBNAME statement creates a LIBREF, equating the libref PERM to a directory (or folder).
2 The modified data set will have the same name as the old data set (replacing it).
3 A SET statement reads the data set named MYSTUFF in the directory I:\.
4 The variable RATIO is created.
5 A label is added to the variable ratio.

Notice, there are **NO INFILE** and **NO INPUT STATEMENTS** used when reading data from a SAS data set. INFILE and INPUT statements are used with raw data and no raw data are being read. If you wanted to create a new data set from the old data set, use a different data set name.

..Example 3.22...

```
data newstuff;                                                         1
set perm.mystuff;
ratio = sbp /dbp;
run;
```

1 Data set NEWSTUFF is created in the WORK library. It will be identical to the data set MYSTUFF in the library PERM, except it will have one additional variable, i.e. ratio.

4) RESTRICTING OBSERVATIONS

Up to now, all the examples of creating SAS data sets from either an instream data file or from an external data file have converted all of the available raw data into a SAS data set. There may be occasions where you want to restrict the data that are placed into a data set, i.e. you do not want to use all of the observations.

...RESTRICTING OBSERVATIONS USING INFILE OPTIONS (FIRSTOBS & OBS)

The OBS and/or FIRSTOBS options can be used to restrict the observations that are processed by an INPUT statement. Within a data step, they are used as options on the INFILE statement.

```
data new;
   infile mydata firstobs=101 obs=200;
   input.....
run;
```

This data step would only read records 101 through 200 from the raw data file. You need not use both options. If you just wanted to skip the first one hundred observations, you could use FIRSTOBS.

```
infile mydata firstobs=101;
```

If you just wanted to process the first one hundred observations, you could use OBS.

```
infile mydata obs=100;
```

Note that OBS refers to an ABSOLUTE observation number, not a relative number of observations. If you write the following INFILE statement with the intent of starting at record number 101 and reading 100 records, you will not read any records.

```
infile mydata firstobs=101 obs=100;
```

Though not written as such, the OBS option is really the (LAST)OBS option, telling SAS the last observation to be processed. Since 100 is less than 101, no records would be processed. If you just use the OBS option, there is an implied FIRSTOBS=1.

If your data are already in SAS data set, you can use FIRSTOBS and OBS as data set options. Some examples are...

```
proc print data=oldstuff (obs=5);
run;

data newstuff;
set oldstuff (firstobs=11 obs=20);
run;
```

...RESTRICTING OBSERVATIONS USING IF, IF-THEN-ELSE, DELETE, OUTPUT, RETURN

Rather than restrict observations based on record number, you can also restrict observations based on the value of one or more variables. For example, you have a data file of that contains the information on births. You only want to save keep the observations where the infant was a male.

...Example 4.1...

```
*SUBSETTING IF statement;
data males;
input
gender      $1.
bwt          4.
county      $2.
;
if gender eq "M";                                                    1
label bwt = 'BIRTHWEIGHT (GRAMS)';
datalines;
M300002
M450011
F234570
m350012
;
run;
```

1 This statement is known as a SUBSETTING IF statement. If the statement is TRUE (i.e., the variable gender is equal to M), SAS continues to process subsequent statements. If the statement is FALSE, SAS immediately returns to the top of the data step. Since the bottom of the data step is never reached, no observation is added to the data set. The "eq" in the subsetting IF statement is a SAS operator meaning equals and is equivalent to using an =='s sign (explained in appendix A). Note that in this example, the observation that contains a gender 'm' would not be added to the data set since the subsetting if statement specifies 'M', not 'm'.

This is the first time that you've seen an explicit mention of when an observation is added to a SAS data set, i.e. when the end of the data step is reached. Any SAS statement that prevents the end of the data step being reached prevents an observation from being added to a data set. The data step in example 4.1 could also have been written as follows.

...Example 4.2...

```
*OUTPUT statement;
data males;
input
gender      $1.
bwt          4.
county      $2.
;
if gender eq "M" then output;                                        1
label bwt = 'BIRTHWEIGHT (GRAMS)';
datalines;
M300002
M450011
F234570
m350012
;
run;
```

1 An OUTPUT command is used to write an observation to a SAS data set

Remember that an observation is normally written to a SAS data set when the end of the data step is reached. However, you can write an observation to a data set at any time within the data step by using OUTPUT. If an OUTPUT statement is used anywhere within a data step, the AUTOMATIC OUTPUT of an observation at the bottom of the data step is "turned off". The utility of the OUTPUT command is may not be obvious from example 4.2, but there are situations where an OUTPUT command is very useful. You might want to create two data sets using one data step.

..Example 4.3...

```
*IF-THEN-ELSE statement;
data males females;                                                    1
input
gender $upcase1.                                                       2
bwt           4.
county       $2.
;
if gender eq "M" then output males;                                    3
else                  output females;
label bwt = 'BIRTHWEIGHT (GRAMS)';
datalines;
M300002
M450011
F234570
m350012
;
run;
```

1 Two data sets, males and females, are created with one data step
2 A SAS-supplied character informat $UPCASE is used to convert character data to uppercase as the data are read. The lower case 'm' in observation four will be converted to an upper case 'M'.
3 An IF-THEN-ELSE statement and OUTPUT commands are used to direct observations to the appropriate data set

If you use an OUTPUT command without specifying a data set, observations are written to all the data sets being created within the data step. In example 4.3, if the data set name FEMALES had been left off of the second OUTPUT command, data set MALES would contain all the observations regardless of gender, while the FEMALES data set would only contain females. Why would this happen? The answer requires knowledge not only of how the OUTPUT statement works, but also of the IF-THEN-ELSE statement.

All the portions of an IF-THEN-ELSE statement are executed until a portion is reached that is true. If a birth record is encountered with a gender equal to 'M', the first portion of the statement is evaluated as TRUE and no more of the statement is executed (the ELSE... portion is not executed). If the first portion is FALSE, i.e. a record with gender not equal to 'M' is encountered, only the ELSE... portion is executed. What would happen if the data set name males is left of the first portion of the if-then-else statement, but the data set name females is present on the second portion?

The IF statement in examples 4.1 can also be written as follows, and the result will be exactly the same as if the SUBSETTING If statement had been used...

```
if sex ne "M" then delete;
```

The following statement may or may not have the same result as a SUBSETTING IF.

```
if sex ne "M" then return;
```

depending on whether there is an OUTPUT statement within the data step. If there is no OUTPUT statement, the RETURN command instructs SAS to perform two actions: write an observation to a SAS data set immediately (an implied OUTPUT command); go back to the top of the data step. If there is an OUTPUT statement anywhere in the data step, a RETURN command instructs SAS to perform only one action, i.e. go back to the top of the data step. Thus, a RETURN command can mean OUTPUT then RETURN, or just RETURN.

What if you collected data from a number of clinics. You knew that the instrument used to measure blood pressure in one clinic (clinic number 2) produced values that were always 5% too low. You could use a RETURN statement to write a data step that corrected only values from that clinic.

..Example 4.4...

```
*RETURN statement;
data clinics;
input
clinic      $1.
sbp          3.
dbp          3.
;
if clinic ne '2' then return;                                                    1
sbp = 1.05 * sbp;
dbp = 1.05 * dbp;
datalines;
1080120
2075110
1110200
;
run;
```

1 If an observation is encountered that is not from clinic '2', the RETURN statement results in an observation written immediately to the data set and a return to the top of the data step to read another observation. If an observation is found from clinic '2', the blood pressure measurements are corrected prior to adding the observation to the data set. The data set clinics will contain 3 observations.

What would happen if an output statement were added to the data step as follows...

```
if clinic ne '2' then return;
sbp = 1.05 * sbp;
dbp = 1.05 * dbp;
output;
```

How many observations would there be in the data set CLINICS? The answer is 1, why?

An IF statement may contain more than one condition. The combination of all the conditions must be true for SAS to perform the desired action...

```
if sex eq "M" and bwt ge 2500;
```

-or-

```
if sex ne "M" or bwt lt 2500 then delete;
```

Remember, the more complex the logic, the harder it might be to write a single statement that selects (or deletes) the appropriate observations. Sometimes, parentheses make it easier to understand the logic. You can also break one statement into a series of statements and achieve the same result...

```
if (gender eq "M") and (bwt ge 2500) and (county ^= "70");
```

-or-

```
if gender eq "M";
if bwt ge 2500;
if county ne "70";
```

...RESTRICTING OBSERVATIONS USING A WHERE DATA SET OPTION OR WHERE STATEMENT

If your data are already in a SAS data set, you can tell SAS that you only want to use selected observations via either a WHERE data set option or a WHERE statement. Both the option or statement can be used in either a data step or a procedure. For example, if you have a data set named ALL that contained data for both males and females. You print a report using only those observations where the variable gender has a value of M. You can do this in a variety of ways.

...Example 4.5...

```
*create data set ALL;
data all;
input gender : $1. age zip;
datalines;
M 50 12205
F 60 12205
M 12 12203
M 32 12212
F 60 12205
F 80 12212
;
run;

data males;
   set all (where=(gender eq "M"));                                    1
run;

proc print data=males;
run;
```

or...

```
data males;
set all;
where gender eq "M";                                                   2
run;

proc print data=males;
run;
```

or...

```
proc print data=all (where=(gender eq "M"));                           3
run;
```

or...

```
proc print data=all;
where gender eq "M";                                                   4
run;
```

1 A WHERE data set option is used. Only males (observations with a gender of M) will be added to data set males.
2 A WHERE statement is used. It has the same result as a WHERE data set option. However, you won't get confused with a lot of parentheses.
3 No data step is used to restrict observations and no new data set is created. The WHERE data set option is used in a PROC and the only obersvations for males are printed.
4 The same as #3, except a WHERE statement is used instead of a data set option in order to avoid counting parentheses.

Notice that since a WHERE statement can be used in a PROC, there is no need to create a new data set if you only want to use a procedure to work with a specified subset of your data set. If you wanted to create a new data set comprising a subset of your original data, you could also use an IF statement within the data steps shown in example 4.5....

```
if gender eq ‘M’;
```

and create a new data set with only observations for males. However, using a WHERE statement is more efficient than a subsetting IF statement since SAS processes the observations differently when a WHERE statement (or data set option) is used. The WHERE statement executes prior to an observation being read by the SET statement while an IF statement executes once the observation is read. Since the WHERE statement reduces the number of observations processed by the SET statement, it is more efficient to use the WHERE statement. Look at the LOG file that results from creating a data set of only males first using a WHERE statement, then a subsetting IF statement.

...Example 4.6...

```
data all;
input gender : $1. age zip;
datalines;
M 50 12205
F 60 12205
M 12 12203
M 32 12212
F 60 12205
F 80 12212
;
run;

data males_w1;
set all;
where gender eq 'M';                                                    1
run;

data males_w2;
set all (where=(gender eq 'M'));                                        2
run;

data males_w3 (where=(gender eq 'M'));                                  3
set all;
run;

data males_if;
set all;
if gender eq 'M';                                                       4
run;
```

1 A WHERE statement is used to create a subset of the original data set. SAS only reads observations that are accepted by the WHERE statement.
2 A WHERE data set option is used instead of a WHERE statement. Once again SAS only reads observations accepted by the WHERE option.
3 The WHERE data set option is used on the new data set, not on the data set being read. SAS reads every observation in data set ALL and creates a subset when it wriires observations to the data set MALES, accepting only those that meet the criteria specified in the WHERE option.
4 A SUBSETTING IF statement is used. As in #3, SAS reads every observation in data set ALL and only writes observations to the data set MALES that meet the criteria specified in the IF statement.

Here is the LOG...

```
242   data males_w1;
243   set all;
244   where gender eq 'M';
245   run;
```

```
NOTE: There were 3 observations read from the data set WORK.ALL.
      WHERE gender='M';
NOTE: The data set WORK.MALES_W1 has 3 observations and 3 variables.

246
247   data males_w2;
248   set all (where=(gender eq 'M'));
249   run;

NOTE: There were 3 observations read from the data set WORK.ALL.
      WHERE gender='M';
NOTE: The data set WORK.MALES_W2 has 3 observations and 3 variables.

250
251
252   data males_w3 (where=(gender eq 'M'));
253   set all;
254   run;

NOTE: There were 6 observations read from the data set WORK.ALL.
NOTE: The data set WORK.MALES_W3 has 3 observations and 3 variables.

255
256   data males_if;
257   set all;
258   if gender eq 'M';
259   run;

NOTE: There were 6 observations read from the data set WORK.ALL.
NOTE: The data set WORK.MALES_IF has 3 observations and 3 variables.
```

Notice the differences among the four data steps in how many observations were read from the data set ALL.

There are a number of expressions that can be used within WHERE statements that are not available for use in IF statements. Several of these expressions are shown in appendix B, for example BETWEEN

..Example 4.7...

```
data all;
input gender : $1. age zip;
datalines;
M 50 12205
F 60 12205
M 12 12203
M 32 12212
F 60 12205
F 80 12212
;
run;

data age_50_69;
set all;
where age between 50 and 69;                                    1
run;
```

1 A WHERE statement is used to select observations in a given age range. A BETWEEN expression is used. Note, you can only use BETWEEN in a WHERE statement. It is not allowed in an IF statement.

When using WHERE statements, you should remember the following...

ONLY USED WITH SAS DATA SETS - WHERE statements can only be used with SAS data sets. You can't subset observations with a WHERE statement if you are reading RAW data (you must use an IF statement).

CAN BE USED IN BOTH DATA STEPS AND PROCEDURES - An advantage of WHERE over an IF statement is that is it can be used in a PROCEDURE to subset observations.

PRODUCES AN ERROR IF THERE IS A 'TYPE MISMATCH' - When you use a WHERE statement to compare the value of a variable to a constant or an other variable, the variable types must match. If they do not, an error occurs. If there is a type mismatch in an IF statement, the data step will continue a NOTE produced informing you that SAS performed either a character-to-numeric or numeric-to-character conversion.

...Example 4.8...

```
data all;
input gender : $1. age zip;
datalines;
M 50 12205
F 60 12205
M 12 12203
M 32 12212
F 60 12205
F 80 12212
;
run;

* a 'forgiving' IF statement;
data z_12205;
set all;
if zip eq '12205';                                                    1
run;

* a 'non-forgiving WHERE statement;
data z_12205;
set all;
where zip eq '12205';                                                 2
run;
```

1 A subsetting IF statement is used to create a data set that contains only observations from the zip 12205. However, ZIP is a numeric variable in data set ALL and the value 12205 in the IF statement is enclosed in quotes — as you would do when comparing a variable value to a character constant. The LOG shows...

```
72   * a 'forgiving' IF statement;
73   data z_12205;
74   set all;
75   if zip eq '12205';
76   run;

NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column).
      75:11
NOTE: There were 6 observations read from the data set WORK.ALL.
NOTE: The data set WORK.Z_12205 has 3 observations and 3 variables.
```

indicating that the data set was created, but character-to-numeric conversion was done in line 75, the line where you used a character value ('12205') instead of a numeric value (12205) to select observations. SAS gave you the benefit of the doubt and assumed you knew what you wanted to do.

2 A WHERE statement is used to create a data set that contains only observations from the zip 12205. This time the LOG shows...

```
78   * a 'non-forgiving WHERE statement;
79   data z_12205;
80   set all;
81   where zip eq '12205';
ERROR: Where clause operator requires compatible variables.
82   run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.Z_12205 may be incomplete.  When this step was stopped there were 0
observations and 3 variables.
WARNING: Data set WORK.Z_12205 was not replaced because this step was stopped.
```

indicating that no new data set was created. The same logic that was permitted by the IF statement produces an ERROR when a WHERE statement is used. You are told that a WHERE statement requires 'compatible' variables. Since ZIP is a NUMERIC variable, you can only compare it to a numeric value.

What happens in the above example if you try either...

```
IF GENDER EQ M;
```

or

```
WHERE GENDER EQ M;
```

in the data step?

(5) RESTRICTING VARIABLES

Just as there are times when you will want to restrict the observations in a SAS data set, you may also want to restrict the variables. By default, all variables mentioned in a data step are added to the SAS data set being created. The default behavior can be altered by DROP and KEEP statements, or by DROP and KEEP data set options.

...Example 5.1...

```
data mystuff;
input
lastname $12.
chol       3.
sbp        3.
dbp        3.
dob  mmddyy8.
;
ratio = sbp / dbp;                                                    1
drop sbp dbp;                                                         2
format dob date9.;
datalines;
ADAMS       23016090 02041767
FILLMORE    300156 8805071800
LINCOLN     222144 8202111809
JOHNSON     190    6010101808
TRUMAN      150140 7012121884
WASHINGTON  11015075 02201732
;
run;
```

1 The variables SBP and DBP are used to calculate the variable RATIO.
2 A DROP statement is used to eliminate variables SBP and DBP from data set MYSTUFF.

Rather than using a DROP statement, a DROP data set option could have been used...

```
data mystuff (drop=sbp sdp);
```

Both the DROP statement and DROP data set option have the same result, i.e variables SBP and DBP are not added to data set MYSTUFF. If there are many variables in a data set, it may be easier to write a KEEP statement or a KEEP data set option. Assume you are given a SAS data set name BIRTHS that contains 5 variables. Among the variables are a birth weight (BWT) and gestation (GEST). You want to read that data set, create two new variables, and keep only the new variables in a new data set.

...Example 5.2...

```
data births;                                                          1
input
id     : $2.
gender : $1.
zip    : $5.
bwt
gest
;
datalines;
01 M 12211 3500 280
02 M 12503 1500 210
03 M 12345 2400 245
04 F 15643 2450 260
;
run;
```

```
data b_risk (keep=bwt_risk gest_risk);                                    2
set births;                                                               3
*** check for low birth weight;
if bwt lt 2500 then bwt_risk = 1;                                         4
else                bwt_risk = 0;
*** check for short gestation;
if gest lt 259 then gest_risk = 1;                                        5
else                gest_risk = 0;
run;

proc print data=b_risk;
run;
```

```
                   gest_
Obs    bwt_risk     risk
 1         0         0
 2         1         1
 3         1         1
 4         1         0
```

1 Data set BIRTHS is created with four observations.
2 A new data set named B_RISK will be created and it will only contain the two variables BWT_RISK and GEST_RISK. A KEEP data set option is used instead of a KEEP statement.
3 The data set with many variables is read by using a SET statement.
4 If the birth weight (in grams) is below 2500, the variable BWT_RISK is given a value of 1, otherwise it is given a value of 0;
5 If the gestation (in days) is below 259 (37 weeks), the variable GEST_RISK is given a value of 1, otherwise it is given a value of 0.

Rather than using a KEEP data set option, a KEEP statement could have been used...

```
data b_risk;
set births;
keep bwt_risk gest_risk;
.....
run;
```

Since we are only using two variables from the data set BIRTHS, we can make the SAS job more efficient by only reading those two variables.

...Example 5.3...

```
data b_risk (keep=bwt_risk gest_risk);                                    1
set births (keep=bwt gest);                                               2
if bwt lt 2500 then bwt_risk = 1;
else                bwt_risk = 0;
if gest lt 259 then gest_risk = 1;
else                gest_risk = 0;
run;
```

1 As in example 26, the contents of the new data set are restricted by using a KEEP data set option.
2 Rather than read all the variables in data set BIRTHS with the SET statement, only the two variables BWT and GEST are read by using a KEEP data set option.

The dropping and keeping of variables is a topic that is part of learning how to write efficient SAS jobs, and learning how to reduce the size of SAS data sets. Neither efficiency nor reduction of data set size is necessary to write a SAS job that performs the tasks you want to accomplish. However, your SAS jobs will take less time to execute and you will save disk space if you learn how and when to use DROP and KEEP statements and data set options. You may have noticed that there was no mention as to where to place DROP and KEEP statements in a data step. Do you think it matters? Part of learning how to use SAS is learning about placement of statements. Maybe you should experiment.

There is one last topic to mention about examples 5.2 and 5.3. In each example, the focus was on how to create a data set with a limited number of variables. In each example, we were creating new variables that took the value of either zero or one. This is sometimes referred to as creating dummy variables. It is a common task with many types of data when one wants to indicate the presence (1) or absence (0) of a particular risk factor. In examples 5.2 and 5.3, IF-THEN-ELSE statements were used to create the dummy variables. There is a shortcut.

...Example 5.4

```
data b_risk (keep=bwt_risk gest_risk);
set births (keep=bwt gest);
*** check for low birth weight;
bwt_risk  = (bwt lt 2500);                                    1
gest_risk = (gest lt 259);                                    2
run;
```

1 An equation is used to create the new variable BWT_RISK. There is an expression within parentheses on the right of the equation. That expression is evaluated as being either TRUE or FALSE. If it is TRUE, BWT_RISK will have a value of 1, and if it is FALSE, a value of 0.
2 The same method is used to create the variable GEST_RISK.

The expression within parentheses can be more involved than just one variable. For example, if you had defined RISK PRESENT (1) as being both low birth weight and short gestation, you could have used...

```
risk  = (bwt lt 2500 and gest lt 259);
```

Only those births with both risk factors would produce a value of one. If you have a lot of variables and are creating risk profiles (0-risk absent, 1-risk present), this method of variable creation will substantially reduce the amount of SAS code used.