SAS Workbook

for Writing SAS Programs
to Process Data on UNIX

SOCIAL SCIENCE COMPUTING COOPERATIVE

November 1999

Table of Contents

Introduction to the Workbook

This workbook provides an introduction to using version 6.12 of SAS on UNIX. It is designed to serve as a self-instructional tutorial as well as notes to accompany training sessions offered by SSCC staff. This workbook concentrates on teaching you how to write SAS programs to read in and process data. Very few statistical or graphical procedures are covered. Once you have mastered the techniques introduced in this workbook, you can consult the documentation provided by SAS Institute to learn about other features of SAS. These documents are circulated by the CDE Print/Virtual Library in 4457 Social Science. There are also numerous handouts written by SSCC staff that you may find useful. These are listed at the end of this workbook.

Prerequisites

You do not need to have any previous experience with SAS or any other statistical software to use this workbook. You also do not need any prior statistical knowledge. What you do need to have is a basic understanding of the UNIX operating system including a text editor such as EMACS or TPU.

How to Use this Workbook

This workbook is designed to be self-instructional. You should read through it in the order it is presented. Exercises are included along the way for review and practice. You should always complete the exercises before proceeding to the next section because the only way to learn SAS is to actually write and execute SAS programs.

Introduction to SAS

SAS is an integrated system of software products that enable you to access, manage, analyze, and present all of your data. The functionality of SAS is

- built around these four primary data-driven tasks common to all applications: access, management, analysis, and presentation.

- portable across computing environments, i.e. SAS applications function the same, look the same, and produce the same results no matter what operating system you are running from.

- surrounded by flexible user interfaces giving the user choices in how he/she interacts with the software.

SAS is composed of numerous integrated software products. Base SAS is the cornerstone that supports these products. It contains a programming language, a data management facility, and data analysis and reporting utilities. The programming language, with its statistics and functions, is integral to SAS because it forms the building blocks from which all SAS applications are created. Combined with the general-purpose base product utilities (or procedures), the SAS language gives base SAS software all the functionality required to access, manage, analyze, and present your data.

Other SAS products include statistical analysis (SAS/STAT), forecasting and modeling (SAS/ETS), Structured Query Language (SAS/SQL), and graphics (SAS/GRAPH). Several other products are available as well.

The material presented in this workbook pertains primarily to Base SAS.

Methods of running SAS

There are four methods of running SAS programs and displaying output. The methods differ in the speed with which they run, the amount of computer resources they require, and the interaction you have with the program (that is, the kinds of changes you can make while the program is running). The results are the same, regardless of the way the programs are run. The following list briefly describes each method:

Batch Mode: To run a program in batch mode, you prepare a file containing a SAS program using a text editor such as EMACS or TPU. Then you execute the program in the background. Your terminal session is free for you to work on something else while the program runs. The results of your SAS program go to a pre-specified destination; you can look at them when the program has finished running.

Non-interactive Mode: In non-interactive mode, you prepare a file of SAS statements and submit the SAS program to the computer. The program runs immediately and occupies your current terminal session. You don't see the results of your SAS program until it has finished running.

Interactive Line Mode: In interactive line mode, you enter one line of a SAS program at a time. SAS recognizes steps in the program and executes them automatically. You can see the results immediately on your screen. A typical SAS session in interactive line mode might look like the following:

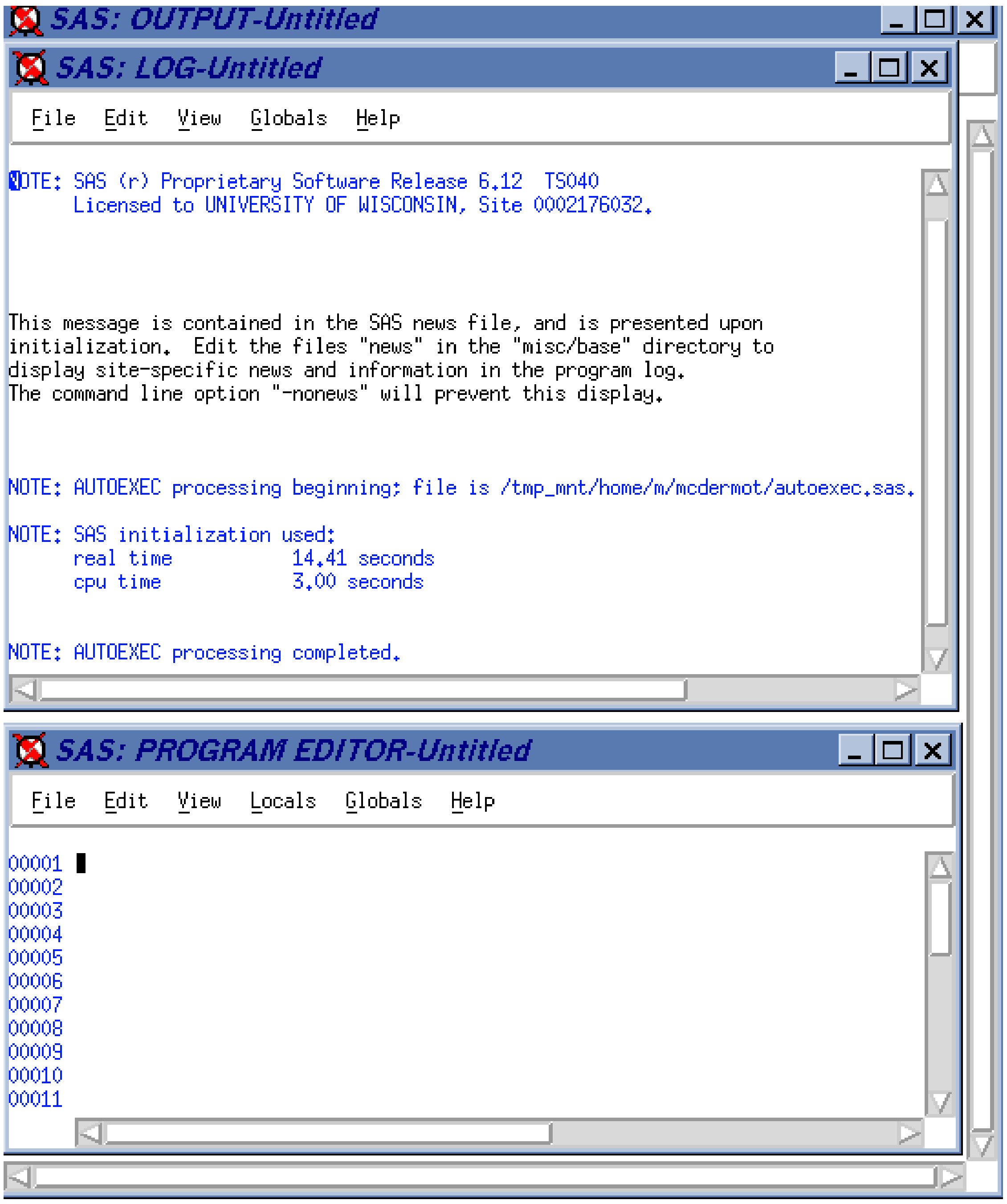
```
norman.ssc.wisc.edu> sas -nonews -nodms
NOTE: Copyright (c) 1989-1993 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 6.10  TS018
      Licensed to UNIVERSITY OF WISCONSIN, Site 0002176032.

NOTE: AUTOEXEC processing completed.

  1? data class; infile "class.dat";
  2? input name $ 1-7 height 9-12 weight 14-18 age 20-21; run;
NOTE: The infile "class.dat" is:
      File Name=/tmp_mnt/home/m/mcdermot/soc365/class.dat,
      Owner Name=mcdermot,Group Name=dpadmn,
      Access Permission=rw-r--r--,
      File Size (bytes)=1107
NOTE: 20 records were read from the infile "class.dat".
      The minimum record length was 21.
      The maximum record length was 21.
NOTE: The data set WORK.CLASS has 20 observations and 4 variables.
  3? proc means; var weight height; run;


      Variable   N          Mean       Std Dev       Minimum       Maximum
      --------------------------------------------------------------------
      WEIGHT    19   100.0263158    22.7739335    50.5000000   150.0000000
      HEIGHT    20    61.7300000     5.6805393    50.2000000    72.0000000
      --------------------------------------------------------------------
```

Display Manager Mode: In display manager mode, you interact directly with SAS via a series of windows. Display manager mode is a quick and convenient way to write, submit, and view the results of your SAS programs. A typical SAS session in display manager mode looks like the following:



```
NOTE: SAS (r) Proprietary Software Release 6.12  TS040
      Licensed to UNIVERSITY OF WISCONSIN, Site 0002176032.



This message is contained in the SAS news file, and is presented upon
initialization.  Edit the files "news" in the "misc/base" directory to
display site-specific news and information in the program log.
The command line option "-nonews" will prevent this display.



NOTE: AUTOEXEC processing beginning; file is /tmp_mnt/home/m/mcdermot/autoexec.sas.

NOTE: SAS initialization used:
      real time           14.41 seconds
      cpu time            3.00 seconds


NOTE: AUTOEXEC processing completed.
```

SAS: PROGRAM EDITOR-Untitled
File Edit View Locals Globals Help

UNIX users may run SAS programs using any of the four methods just described. However, in this workbook, only running SAS programs in noninteractive and batch mode will be described. Refer to SSCC Pub #7-4, "*Using SAS on UNIX*", for instructions on how to run SAS programs under UNIX in the other two modes.

Programming with SAS

This chapter introduces you to the basic components of SAS. It defines and describes SAS data sets, SAS programs, and SAS statements. These basic components are discussed in more detail in later chapters.

SAS Data Sets

SAS reads data in various forms and organizes them into a rectangular form called a SAS data set. Following is an example of a typical SAS data set. The data represent persons in a class and consist of their name, sex, weight, height, and age.

NAME	SEX	AGE	HEIGHT	WEIGHT
Audrey	M	41	74	170
Ron	M	42	68	166
Carl	M	32	70	155
Antonio	M	39	72	167
Deborah	F	30	66	124
Jacqueline	F	33	66	115
Helen	F	26	64	121
David	M	30	71	158
James	M	53	72	175
Michael	M	32	69	143
Ruth	F	47	69	139
Joel	M	34	72	163
Donna	F	23	62	98
Roger	M	36	75	160
Yao	M		70	145
Elizabeth	F	31	67	135
Tim	M	29	71	176
Susan	F	28	65	131

A data value is a single unit of information, such as one person's height. A variable is a set of data values that describe a specific characteristic, for example the weight of all the individuals in the class. The weight values make up the WEIGHT variable, the name values make up the NAME variable, and so on. In the figure above, each column represents a variable.

SAS variables can be classified as character or numeric. Character variables contain data values consisting of a combination of letters of the alphabet, numbers, and special characters or symbols. Numeric variables contain values consisting only of numbers and related symbols, such as decimal points, plus signs, and minus signs.

An observation is a set of data values for the same item, for example all measurements for one person. In the figure above, each row represents an observation. So, a SAS data set has a rectangular organization with variables representing the columns and observations representing the rows.

Missing values represent missing or unavailable data values to SAS and are represented as either periods or blanks, depending on the method of data entry and the type of data value. Find the missing value in the figure above.

SAS Programs

SAS programs consist of one or more steps made up of instructions called statements. The steps are always one of two types: DATA steps or PROC steps. A DATA step consists of a group of statements in the SAS language that read raw data or existing SAS data sets and perform calculations and manipulations. A PROC step consists of a group of statements that allow you analyze the data and write reports.

For processing, you simply arrange the steps in the order you want tasks to be performed. SAS processes the first step, then the second, and so on, independently of other steps. The following is an example of a SAS program that creates a SAS data set named CLASS, then creates scatter plots involving some of the variables, and finally carries out a regression analysis:

```
data class;
   infile "~/rawdata/weight.dat";
   input name $ 1-7 height 9-12 weight 14-18 age 20-21;
   ht_cm = height * 2.54;
 run;

proc plot data=class;
   plot height*age weight*age;
run;
```

```
   proc reg data=class;
      model weight=age height;
      plot student.*p.;
   run;
```

The DATA step portion of the program begins with the keyword DATA and ends with the RUN statement following the last data record. The data step is composed of the following elements:

- The DATA statement tells SAS to begin building a SAS data set named CLASS.
- The INFILE statement tells SAS where to find the data file to be processed.
- The INPUT statement identifies the fields to be read from the input data and names the SAS variables to be read from them.
- The fourth statement is an assignment statement; it converts the height measurements from inches to centimeters and assigns the results to a new variable, HT_CM.
- The RUN statement tells SAS that the preceding statements are ready to be executed. This statement ends the DATA step.

Following the DATA step are two PROC steps, PLOT and REG. The PROC PLOT step requests plots of the data and PROC REG step requests a regression analysis.

The statements that created the data set CLASS are part of the SAS programming language. Of course, as with any programming language, there are rules.

Rules for SAS Statements

- SAS statements end with a semicolon.
- SAS statements can be entered in lowercase, uppercase, or a mixture of the two.
- Any number of SAS statements can appear on a single line.
- A SAS statement can be continued from one line to the next, as long as no word is split.
- SAS statements can begin in any column.

! Words in SAS statements are separated by blanks or by special characters.

Rules for SAS Names

SAS names are used for SAS data set names, variable names, and other items. The following rules apply to SAS names:

! A SAS name can contain from one to eight characters.

! The first name must be a letter or underscore (_).

! Subsequent characters must be letters, numbers, or underscores.

! Blanks cannot appear in SAS names.

Exercise 1

1. Correct the following SAS program and make it more readable:

```
proc     glm data=WGTloss;  class
   sex age;
model wgt=
se
x
age
;
run;
```

2. Which of the following are valid SAS data set names?

   Census90

   Census_90

   CEN_90

   CEN*90

   cen-90

Running Your SAS Program

Once you have written a SAS program, then you need to "run" (or execute) the program. Executing a program requires disk space and memory, both resources that must be shared by all users logged on to a UNIX computer. Because of this it is very important to have the skills necessary to manage your SAS jobs well. The SSCC Publication, *Research Computing on SSCC UNIX Systems*, will give you all the information you need. Be sure you read it before executing SAS jobs utilizing large SAS data sets.

As was stated earlier in "*Methods for running SAS*," you may run SAS in four different modes. Two of the modes are interactive (Interactive Line Mode and Display Manager Mode) and two are noninteractive (Noninteractive Mode and Batch Mode). Only the two noninteractive modes are described here.

Running SAS noninteractively requires the following basic steps:

1. Using an editor to create or edit a file containing your SAS statements.
2. Invoking SAS and executing your SAS program (in either the foreground or background).
3. Examining the output from your SAS program to ensure that no errors occurred. If errors did occur, it may be necessary to repeat steps 1 through 3 until the job runs satisfactorily.

Once you have created a file of SAS commands, you are ready to execute SAS. The general form of the SAS command is as follows:

```
 sas filename -option1...-optionn
```

where *`filename`* is the name of the file containing the SAS program to be executed. *`option`* specifies a SAS system option to configure your session. Some common options include:

`-linesize` *`n`*	Specifies the line size of the SAS output. The range is 64 to 256.
`-obs` *`n`*	Specifies the last observation from a data set that SAS is to read.
`-pagesize` *`n`*	Specifies the number of lines that can be placed in a page of SAS output. Values can range from 15 to 32,767.

`-noreplace` Specifies that SAS not replace data sets already created.

Noninteractive Mode

To invoke SAS in noninteractive mode, enter the SAS command followed by the name of the file containing the SAS program to be executed. For example, suppose you have stored your SAS statements in a file named `foo.sas.` To invoke SAS in noninteractive mode and execute the program `foo.sas` you would type the following:

```
 > sas foo
```

Note that you do not have to include the file extension in the filename when the file extension is `.sas`. SAS uses `.sas` by default.

You do not get another UNIX prompt until SAS finishes executing the program. When SAS finishes and you get the UNIX prompt, two new files are in your working directory which contain the SAS output. `foo.log` contains the log of the SAS session and `foo.lst` contains the output from the SAS commands in `foo.sas`.

In the example just shown, SAS created two files: one to hold the SAS output and one to hold the log. By default, the filenames are the same as the filename for the file containing the SAS program. If you want to direct your output and log to other files, use the `print` and `log` system options. For example,

```
 > sas foo -print report -log report.log
```

After your job is executed, the output goes to the file `report` and the log goes to the file `report.log.`

Batch Mode

To execute a program in batch mode, you simply run it in the background by typing an & at the end of the command. For the `foo.sas` example above, you would type the following:

```
 > sas foo &
```

The only difference between running batch and running noninteractive is that in batch mode, your job is executed in the background, meaning you do not have to wait until the SAS program finishes execution before you get the UNIX prompt. In other words, your shell is available for other work.

SAS errors

SAS error messages, should errors occur, will be found in the log file once a program finishes executing. SAS prints messages that enable you to verify that the

- ! appropriate raw data file was read
- ! correct number of records were read
- ! resulting SAS data set contains the correct number of variables and observations as illustrated in the following SAS output:

```
NOTE: The infile "date.dat" is:
      File Name=/tmp_mnt/home/m/mcdermot/soc365/date.dat,
      Owner Name=mcdermot,Group Name=dpadmn,
      Access Permission=rw-r--r--,
      File Size (bytes)=1107

NOTE: 27 records were read from the infile "date.dat".
      The minimum record length was 40.
      The maximum record length was 40.
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).
      10 at 13:23   10 at 14:9    10 at 14:24   10 at 14:35
NOTE: The data set WORK.DATE has 27 observations and 7 variables.
```

When SAS detects an error, it usually underlines the error or underlines the point at which it detects the error, identifying it by a number. Each number is uniquely associated with a message. Then it enters syntax check mode. In syntax check mode, SAS no longer executes statements, but continues reading statements, checking their syntax, and underlining additional errors if necessary.

In a batch program, an error in a DATA step statement causes SAS to remain in syntax check mode for the rest of the program. It doesn't execute any more DATA or PROC steps. A syntax error in a PROC step usually affects only that step. At the end of the step, SAS writes a message for each error detected.

Syntax errors are the most common type of errors encountered. These include misspelled keywords and missing or invalid punctuation. Following are examples of common syntax errors and SAS's accompanying error messages:

Omitted semicolon:

```
 21? proc print var height weight; run;
```

```
21   proc print var height weight; run;
               --- ------ ------
               202 202    202
ERROR 202-322: The option or parameter is not recognized.

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

Misspelled Words:

```
22? proc prent; run;
```

```
ERROR: Procedure PRENT not found.
NOTE: The SAS System stopped processing this step because of errors.
```

```
23? proc print; var hight weight; run;
```

```
ERROR: Variable HIGHT not found.
NOTE: The SAS System stopped processing this step because of errors.
```

Unbalanced quotation marks:

```
  1? data class; infile "class.dat;
  2?  input name $ 1-7 height 9-12 weight 14-18 age 20-21;
  3? run;
  4? proc means; var height weight age;
  5? output out=temp mean=mheight mweight mage;
  6? run;
  7? proc print; var mheight mweight mage; run;
  8? proc contents data=class; run;
```

```
WARNING: The current word or quoted string has become more than 200 characters
         long.  You may have unbalanced quotation marks.
```

Data errors:

SAS detects errors in the data and prints error messages when:

! invalid data are found in a field

! illegal arguments are used in functions

! impossible mathematical operations are requested.

When a data error is encountered, SAS

! assigns a missing value to the appropriate variable

! prints a note that describes the error

! displays the input record being read

! displays the values in the SAS observation being created

! continues reading.

The following SAS code results in a data error:

```
 24?  data class; infile "class.dat";
 25?  input name $ 1-7 height 5-12 weight 14-18 age 20-21;
run;
```

The output follows:

```
NOTE: The infile "class.dat" is:
      File=/tmp_mnt/home/m/mcdermot/soc365/class.dat
NOTE: Invalid data for HEIGHT in line 1 5-12.
RULE:
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
1         Alfred  69.0 112.5 14 21
NAME=Alfred HEIGHT=. WEIGHT=112.5 AGE=14 _ERROR_=1 _N_=1
NOTE: Invalid data for HEIGHT in line 2 5-12.
2         Alice   56.5  84.0 13 21
NAME=Alice HEIGHT=. WEIGHT=84 AGE=13 _ERROR_=1 _N_=2
NOTE: Invalid data for HEIGHT in line 3 5-12.
3         Barbara 65.3  98.0 13 21
NAME=Barbara HEIGHT=. WEIGHT=98 AGE=13 _ERROR_=1 _N_=3
NOTE: Invalid data for HEIGHT in line 4 5-12.
4         Carol   62.8 102.5 14 21
NAME=Carol HEIGHT=. WEIGHT=102.5 AGE=14 _ERROR_=1 _N_=4
NOTE: Invalid data for HEIGHT in line 5 5-12.
5         Henry   63.5 102.5 14 21
```

```
NAME=Henry HEIGHT=. WEIGHT=102.5 AGE=14 _ERROR_=1 _N_=5
NOTE: Invalid data for HEIGHT in line 20 5-12.
20        William 66.5 112.0 15 21
NAME=William HEIGHT=. WEIGHT=112 AGE=15 _ERROR_=1 _N_=20
NOTE: 20 records were read from the infile "class.dat".
      The minimum record length was 21.
      The maximum record length was 21.
NOTE: The data set WORK.CLASS has 20 observations and 4 variables.
```

You can avoid many errors by simply taking the time to read over your SAS program before you submit it. In particular, check the following:

! All SAS statements end in a semicolon; be sure you haven't omitted any semicolons or accidentally typed the wrong character.

! Any starting and ending quotes must match; you can use either single or double quotes.

! Most SAS statements begin with a SAS keyword. (Exceptions are the assignment statements and sum statement. Be sure you haven't misspelled or omitted any of the keywords.

! Every DO statement must be followed by an END statement.

You can ask SAS to check the syntax of your program without actually reading the data. This is particularly useful if you have a program that reads a lot of data and thus requires a lot of CPU time. To have SAS check the syntax of your program, invoke SAS with the following system options:

```
> sas foo -noreplace -obs 0
```

The `-obs` option identifies the last record to read in the data. In this case it is set to zero so no data are read. `-noreplace` instructs SAS not to replace any SAS data sets.

If your log file contains errors, you will first need to correct them and then resubmit the job. To correct errors, you simply edit the original command file (Not the log file!).

Exercise 2

The purpose of this exercise is to get you familiar with the steps involved in writing and running a SAS program.

? 1. Invoke EMACS and create a new file called sas_ex2.sas by typing the following at the UNIX prompt:

   ```
   emacs sas_ex2.sas
   ```

? 2. Type in the following SAS command file:

   ```
   data class;
   infile "/users/d1/mcdermot/soc365/class.dat";
   input name $ height weight age;
   run;
   proc print; run;
   ```

? 3. Save your file and exit EMACS by typing <CNTRL>X <CNTRL>C.

4. Run your SAS job by typing the following at the UNIX prompt:

   ```
   sas sas_ex2
   ```

5. When your program finishes, display the log file on your screen using the following command:

   ```
   more sas_ex2.log
   ```

6. If you find any errors in your log file, go back into EMACS, correct any mistakes, and then rerun your program.

7. Once your log file is error free, display the output file on your screen by typing the following command:

   ```
   more sas_ex2.lst
   ```

Creating SAS Data Sets

The first and often most difficult task in data analysis is getting your data into a form that the software package can recognize and handle. Once your data are in that form, it is relatively easy to analyze them and generate reports. To create a SAS data set, you must write a program of SAS statements, which instruct SAS to perform specific tasks. The following SAS statements are used to create a SAS data set:

- the DATA statement
- the INPUT statement
- the CARDS or INFILE statement

To understand how to use SAS statements, you first need to know how SAS creates a SAS data set:

- It reads the DATA statement, creates the structure of a SAS data set, and marks the statement as the point to begin processing for each data line. You can think of the DATA statement as the beginning of a DO loop.

- It uses the description in the INPUT statement to read the data line and produce an observation.

- It uses the observation to execute any other SAS statements that are present like assignment statements or IF-THEN/ELSE statements.

- It adds the observation to the data set being created.

The statements are executed once for each data line.

DATA Statement

A SAS DATA statement instructs SAS to create and name a SAS data set. DATA statements begin with the keyword DATA and specify the name you select for the data set. For example, the following DATA statement instructs SAS to create a SAS data set named CPS90:

```
   DATA CPS90;
```

The rules for naming SAS data sets are listed above in the section, "*Rules for SAS Names*".

INPUT Statement

The INPUT statement provides the information SAS requires to organize data into a SAS data set, such as each variable's name, type, and, if necessary, column location. INPUT statements follow the DATA statement as in the following example:

```
      input name $ 1-7 height 9-12 weight 14-18 age 20-21;
```

The "$" after a variable name is used to denote a character variable.

SAS provides you with three basic input styles: list, column, and formatted. These styles may be used individually or in combination with each other. Formatted input is used in special situations when the data contain nonstandard numeric or character values and are not discussed here.

List Input

List input uses the simplest form of the INPUT statement and may be used when each data value is separated from the next by at least one blank space. The general form of the INPUT statement with the list input method is:

```
   INPUT variable1 variable2 ... variablen;
```

where `variable` is the valid SAS name that identifies the variable. Notice that you do not give column locations with list input. For example:

```
      input name $ height  weight age;
      cards;
   Alfred    69.0 112.5   14
   Alice   56.5  . 13
   Barbara 65.3    98.0 13
   Carol   62.8  102.5 14
      run;
```

The "$" is used to indicate a character variable.

List input is very convenient and simple to use. Keep in mind though, the following restrictions:

! Fields must be separated by at least one blank.

- **!** Fields must be specified in order.
- **!** Missing values must be represented by a place holder such as a period (.). (A blank field causes the matching of variable names and values to get out of sync.)
- **!** Character values cannot contain embedded blanks.
- **!** Variable values must contain eight or fewer characters. A longer value is truncated.

Column Input

Column input is the most common method of entering data and is appropriate when the data values occupy the same fields within each record. The general form of the INPUT statement with the column input method is:

```
   INPUT variable1 startcol-endcol ... variablen
startcol-endcol;
```

where `variable` is the valid SAS name that identifies the variable, `startcol` identifies the beginning of the input field, and `endcol` identifies the end of the input field.

When using column input you are not required to indicate missing values with a placeholder such as a period as with list input:

```
   data class;
      input name $ 1-7 height 9-12 weight 14-18 age 20-21;
      cards;
   Alfred  69.0 112.5 14
   Alice   56.5  84.0 13
   Carol   62.8 102.5 14
   Henry   63.5 102.5 14
   James   57.3  83.0
   Radhika 50.2       13
   run;
```

One advantage of using column input over simple list input is that it allows character variables to contain embedded blanks as in the example below:

```
   data class2;
      input name $ 1-15 sex $ 16 height 17-20 weight 22-26 age
           28-29;
      cards;
   Alfred Loy     M69.0 112.5 14
   Alice Kufeld   F56.5  84.0 13
   Barbara Weeks  F65.3  98.0 13
   Carol   Wolffe F62.8 102.5 14
   ;
```

The above example also illustrates two other advantages of column input over list input:

! variable values can be over eight characters.

! data values need not be separated from the next value by blanks.

Column input also allows fields to be skipped altogether or to be read in any order. Using column input to read the same CLASS2 data above, you can cause the value for the variable WEIGHT to be omitted altogether with the following INPUT statement:

```
   input name $ 1-15 sex $ 16 height 17-20 age 28-29;
```

Exercise 3

Write an INPUT statement for the following data using list input and then column input:

```
         1         2         3         4
----5----0----5----0----5----0----5----0

morphine n  .04  .20  .10  .08
morphine n  .02  .06  .02  .02
morphine n  .07 1.40  .48  .24
morphine n  .17  .57  .35  .24
morphine y  .10  .09  .13  .14
morphine y  .12  .11  .10   .
morphine y  .07  .07  .06  .07
morphine y  .05  .07  .06  .07
trimeth  n  .03  .62  .31  .22
trimeth  n  .03 1.05  .73  .60
trimeth  n  .07  .83 1.07  .80
trimeth  n  .09 3.13 2.06 1.23
trimeth  y  .10  .09  .09  .08
trimeth  y  .08  .09  .09  .10
trimeth  y  .13  .10  .12  .12
trimeth  y  .06  .05  .05  .05
```

Mixed Input

Once you begin an INPUT statement in a particular style, you are not restricted to using that style alone. You can mix input styles in a single INPUT statement as long as you mix them in a way that appropriately describes the records of raw data. For example:

```
data class2;
   input name $ 1-15 sex $ 16 height 17-20 weight age;
   cards;
Alfred Loy     M69.0 112.5 14
Alice Kufeld   F56.5  84.0 13
Barbara Weeks  F65.3  98.0 13
Carol   Wolffe F62.8 102.5 14
;
```

The variables NAME, SEX, and HEIGHT are read with column input and the variables WEIGHT and AGE are read with list input.

Exercise 4

Write an INPUT statement for the following data using mixed input:

```
         1         2         3         4
----5----0----5----0----5----0----5----0

morphine  n  .04  .20  .10  .08
morphine  n  .02  .06  .02  .02
morphine  n  .07 1.40  .48  .24
morphine  n  .17  .57  .35  .24
morphine  y  .10  .09  .13  .14
morphine  y  .12  .11  .10
morphine  y  .07  .07  .06
morphine  y  .05  .07  .06  .07
trimethinen  .03  .62  .31  .22
trimethinen  .03 1.05  .73  .60
trimethinen  .07  .83 1.07  .80
trimethinen  .09 3.13 2.06 1.23
trimethiney  .10  .09  .09  .08
trimethiney  .08  .09  .09  .10
trimethiney  .13  .10  .12  .12
trimethiney  .06  .05  .05  .05
```

Reading from the Same Record Twice

Sometimes you may need to tell SAS to hold a record and read from it again. This is useful when you need to test a condition before creating an observation from a data record. For example, to create a SAS data set that is a subset of a larger group of records, you may need to test for a condition before deciding if a particular record should be used to create an observation in the data set you want to create. Placing the trailing at-sign (@) before the semicolon at the end of an INPUT statement instructs SAS to hold the current data line in the input buffer so it is available for a subsequent INPUT statement. You can set up this process by following these steps:

1. Use an INPUT statement to read a record.

2. Use a trailing @ at the end of the INPUT statement to hold the record in the input buffer for the execution of the next INPUT statement.

3. Use an IF statement or SELECT group (discussed later) to test for a condition.

4. If the condition is met, use INPUT again to read the record to create an observation.

To read from a record twice, you must prevent a new record from automatically being read into the input buffer when the second INPUT statement is executed. Use of a trailing @ in the first INPUT statement serves this purpose.

For example, the CLASS2 data contain information about both males and females. The following data step creates a SAS data set that contains only females:

```
data class2;
   input sex $ 16 @;
   if sex="F";
   input name $ 1-15 height 17-20 weight 22-26 age 28-29;
```

In this DATA step the following actions occur:

! The first INPUT statement reads a record into the input buffer and assigns the variable SEX.

! The IF statement allows the current iteration of the DATA step to continue only when the value of SEX is F. When it is not, the current iteration stops and SAS returns to the top of the data set and releases the held record from the input buffer.

! The second INPUT statement executes only when the value of SEX is F. It does not cause a new record to be read into the input buffer.

Following is the SAS data set created by the above example:

```
  1? data class2;
  2?    input sex $ 16 @;
  3?    if sex="F";
  4?    input name $ 1-15 height 17-20 weight 22-26 age 28-29;
  5?    cards;
6> Alfred Lay     M69.0 112.5 14
7> Alice Kufeld   F56.5  84.0 13
8> Barbara Weeks  F65.3  98.0 13
9> Carol   Wolffe F62.8 102.5 14
10>   ;

NOTE: The data set WORK.CLASS2 has 3 observations and 5 variables.
NOTE: DATA statement used:
      real time           0.979 seconds
      cpu time            0.270 seconds

 11? proc print; run;


           OBS    SEX         NAME         HEIGHT    WEIGHT    AGE
            1      F     Alice Kufeld       56.5       84.0     13
            2      F     Barbara Weeks      65.3       98.0     13
            3      F     Carol   Wolffe     62.8      102.5     14

NOTE: PROCEDURE PRINT used:
      real time           0.402 seconds
      cpu time            0.080 seconds
```

Exercise 5

Modify the following data step to read in only observations with DOSE < 40 using the methods described above:

```
DATA PROBIT;
   INPUT prep $ 1-5 dose 11-12 symptoms $ 19-24 n 30-31;
 CARDS;
stand     10      None       33
stand     10      Mild        7
stand     10      Severe     10
stand     20      None       17
stand     20      Mild       13
stand     20      Severe     17
stand     30      None       14
stand     30      Mild        3
stand     30      Severe     28
stand     40      None        9
stand     40      Mild        8
stand     40      Severe     32
test      10      None       44
test      10      Mild        6
test      10      Severe      0
test      20      None       32
test      20      Mild       10
test      20      Severe     12
test      30      None       23
test      30      Mild        7
test      30      Severe     21
test      40      None       16
test      40      Mild        6
test      40      Severe     19
;
```

Reading Multiple Records to Create a Single Observation

Consider the situation where you have several records containing all the information to construct a single observation. Consider again the CLASS data but assume the data are entered in such a way that information about a single person is spread across several records, as in:

```
Alfred
69.0 112.5
14
Barbara
65.3  98.0
13
Carol
62.8 102.5
14
```

instead of in single records as shown earlier:

```
Alfred  69.0 112.5 14
Barbara 65.3  98.0 13
Carol   62.8 102.5 14
```

There are several ways to read these data into SAS but perhaps the best way is with the #n line-pointer control. The #n line-pointer control character forces the pointer to go to the nth line in the input buffer. The following example uses the #n line-pointer control to read in the data above:

```
data class;
   input name $ 1-7
         #2 height 1-4 weight 6-10
         #3 age 1-2;
   cards;
Alfred
69.0 112.5
14
Barbara
65.3  98.0
13
Carol
62.8 102.5
14
;
```

CARDS and INFILE Statements

When you include the raw data in your SAS program you use the CARDS statement to inform SAS that data lines immediately follow. The CARDS statement follows the INPUT statement. When the raw data are stored on disk, you must tell SAS where to find the data by using an INFILE statement. INFILE statements begin with the keyword INFILE, which is followed by the name of the file containing the data. As the following example illustrates, INFILE statements precede INPUT statements:

```
data class;
   infile "~/soc365/class.dat";
   input name $ 1-7
         #2 height 9-12 weight 14-18
         #3 age 20-21;
run;
```

The name of the file must be enclosed in quotes. The file specification must be a valid path name to the external file you want to access; therefore the level of specification required depends on your location in the directory structure. Following are examples of valid file specifications:

```
infile 'class.dat';
infile '~/sasclass/class.dat';
```

Note: SAS can also read compressed data. Refer to SSCC Pub. #7-4, *Using SAS on UNIX*, for detailed instructions.

Exercise 6

Use an editor to create the following data file called `multiple.dat`:

```
1 10 22
2 500
2 3 10
1 1000
2 6 7
4
```

Each observation is contained in two records. The first record contains values for three variables: HHTYPE, HHSIZE, and REGION. The second record contains KIDSLT6 and WEEKSLY. Write a SAS program, `multiple.sas`, that creates a SAS data set named HOUSE.

Modifying Data using SAS Statements

Although many SAS data steps consist simply of DATA, INPUT, CARDS, or INFILE statements, SAS provides numerous optional statements that enable you to modify data. For example, some statements create new variables, delete observations, or perform specific tasks based on certain conditions. These modifications can be made at the time the SAS data set is first created or later using the SET statement. In this chapter you will learn about many ways to modify SAS data sets using SAS statements. Then, in the next chapter, you will use these techniques to read hierarchical files.

Using Assignment Statements

SAS enables you to create variables and modify existing variable values with assignment statements. Assignment statements appear after the INPUT statement as shown below:

```
data class;
   infile "class.dat";
   input name $ 1-7 height 9-12 weight 14-18 age 20-21;
   ht_cm = height * 2.54;
run;
```

Note that the assignment statement begins with the name of the new variable rather than a SAS keyword. To create a new variable with an assignment statement, do the following:

1. Select a name for the new variable.

2. Determine the formula needed to calculate the values of the new variable.

3. Write the formula as a SAS statement, putting the new variable name on the LEFT side of the equals sign.

SAS processes arithmetic operators in the same order of priority as standard mathematical expressions. Exponentiation calculations are processed first, then multiplication or division, and then addition or subtraction.

SAS Functions

SAS contains many built-in expressions called functions that you can use in your assignment statements. For example, to transform WEIGHT to the natural logarithm of WEIGHT you use the following assignment statement:

```
  lgweight = log(weight);
```

As another example, say you want to sum up the values of HEIGHT, WEIGHT, and AGE for each observation. You would use the SUM function as the following example illustrates:

```
 sumvars = sum(height, weight, age)
```

The SUM function calculates the sum of its arguments, ignoring missing values. You can also combine functions as the following example illustrates:

```
  logsum = log(sum(height, weight, age));
```

In general, a SAS function performs the calculation indicated using the data values within parentheses (called arguments). Separate the arguments with commas. See the "*SAS Language Guide*" for a complete list of all the SAS functions.

Storing Numeric Variables Efficiently

Data sets used by social scientists are typically very big to begin with and by the time you have created all your new variables using the methods described in the previous sections, the data sets are gigantic! At some point, you may need to think about the storage space your data set occupies. There are ways to save space when you store numeric variables in SAS data sets. This section describes one such method which uses the LENGTH statement. Another method for saving space involving data compression is discussed later in this document.

By default, SAS uses 8 bytes of storage in a data set for each numeric variable. So, if you create 500 variables (which is not at all out of the ordinary), that is 4000 bytes. When numeric variables contain only integers, you can often shorten them in the data set being created. For example, a length of four bytes stores accurately all integers up to at least 2,000,000 on most operating systems. To change the number of bytes used for each variable, use a LENGTH statement.

A LENGTH statement contains the names of the variables followed by the number of bytes to be used for their storage. For example, say your data set contains three integer fields. To create three variables INT1, INT2, and INT3 with length 4 bytes you would include the following statement in your data step:

```
  length int1 int2 int3 4;
```

If your data contain small integers like dummy variables for example, you could reduce their length even further:

```
  length int1-int3 4 dum1 dum2 2;
```

Warning: You can safely shorten variables containing integers, but do not shorten variables containing fractions as this can alter the fraction significantly.

Exercise 7

Modify the following data step to store the numeric variables more efficiently using a LENGTH statement. Create a new variable, LDOSE, that is the natural logarithm of DOSE. Create a new variable that is equivalent to DOSE divided by 10.

```
DATA PROBIT;
 INPUT prep $ dose symptoms $ n;
 CARDS;
stand     10      None       33
stand     10      Mild        7
stand     10      Severe     10
stand     20      None       17
stand     20      Mild       13
stand     20      Severe     17
stand     30      None       14
stand     30      Mild        3
stand     30      Severe     28
stand     40      None        9
stand     40      Mild        8
stand     40      Severe     32
test      10      None       44
test      10      Mild        6
test      10      Severe      0
test      20      None       32
test      20      Mild       10
test      20      Severe     12
test      30      None       23
test      30      Mild        7
test      30      Severe     21
test      40      None       16
test      40      Mild        6
test      40      Severe     19
;
```

Using IF-THEN/ELSE Statements

Suppose you want to carry out an action for specific observations in a data set. You can use an IF-THEN statement to test observations to determine if certain conditions are true or false. If the condition is true, SAS carries out the action specified in the THEN clause. If the condition is false, SAS continues to the next statement. The general form of an IF-THEN statement is:

```
IF  condition THEN statement;
```

For example, to separate the people in the CLASS data set into two groups based on age, submit the following statements:

```
data class;
   infile "class.dat";
   input name $ 1-7 height 9-12 weight 14-18 age 20-21;
   ht_cm = height * 2.54;
   if age < 12 then agegroup=1;
   if age >= 12 then agegroup=0;
run;
```

An IF condition can be a simple comparison of a variable and a value, a comparison of two or more variables, or a comparison of several variables or values joined by AND and OR operators, as in the following examples:

```
if age>20 then delete;
if sex="F" and weight gt 110 then size="small";
if name="Nancy" or name="Art" then delete;
if weight le 120;
```

DELETE in the examples above cause observations meeting the condition to be deleted from the data set. The last IF statement above is an example of what SAS calls a subsetting IF statement. Subsetting IF statements contain only a condition. The implicit action in a subsetting IF statement is always the same: if the condition is true, continue processing the observation; if the condition is false, stop processing the observation and return to the top of the DATA step for a new observation.

AND and OR logical operators require you enter a variable name each time you enter a corresponding value. The last example written:

```
if name="Nancy" or "Art" then delete;
```

would NOT produce the desired effect.

ELSE statements are used after IF-THEN statements to provide alternate actions when an IF condition is false. If a condition is false, SAS omits the THEN action and continues to the next statement. ELSE statements enable you to specify what you want the system to do if a condition is false. For the example above separating people into two groups based on age, rewriting it with an ELSE statement would result in the following:

```
   data class;
      infile "class.dat";
      input name $ 1-7 height 9-12 weight 14-18 age 20-21;
      ht_cm = height * 2.54;
      if age < 12 then agegroup=1;
      else agegroup=0;
   run;
```

Using an ELSE statement after an IF-THEN statement provides ONE alternative action when the IF condition is false. However, many cases involve a series of mutually exclusive conditions, each of which requires a separate action. For example, suppose you want to classify the ages into three age groups, 1, 2, and 3. In that case, write a series of IF-THEN and ELSE statements:

```
   if age < 12 then agegroup=1;
     else if age < 15 then agegroup=2;
        else agegroup=3;
```

Exercise 8

Write an IF-THEN/ELSE statement for the following data step that creates two new variables, DUM1, that is equal to one when SYMPTOMS equals "None" and is zero otherwise, and DUM2, that is equal to one when SYMPTOMS equals "Mild" and is zero otherwise.

```
DATA PROBIT;
   INPUT prep $ dose symptoms $ n;
 CARDS;
stand     10      None       33
stand     10      Mild        7
stand     10      Severe     10
stand     20      None       17
stand     20      Mild       13
stand     20      Severe     17
stand     30      None       14
stand     30      Mild        3
stand     30      Severe     28
stand     40      None        9
stand     40      Mild        8
stand     40      Severe     32
test      10      None       44
test      10      Mild        6
test      10      Severe      0
test      20      None       32
test      20      Mild       10
test      20      Severe     12
test      30      None       23
test      30      Mild        7
test      30      Severe     21
test      40      None       16
test      40      Mild        6
test      40      Severe     19
;
```

Using Do Groups

Suppose you need to change several different variable's values for some given condition. For example, suppose you have income data over a period of years and you want to adjust the income depending on the person's sex. One way to do this is to write a series of IF-THEN statements:

```
   if sex = 'F' then salary70=salary70*1.11;
   if sex = 'M' then salary70=salary70*0.78;
   if sex = 'F' then salary80=salary80*1.23;
   if sex = 'M' then salary80=salary80*0.67;
   if sex = 'F' then salary90=salary90*1.12;
   if sex = 'M' then salary90=salary90*0.87;
```

To avoid writing the IF condition twice for each sex, use a DO group as the THEN clause, as the following example illustrates:

```
  if sex = 'F' then do;
   salary70=salary70*1.11;
   salary80=salary80*1.23;
   salary90=salary90*1.12;
  end;
  else if sex = 'M' then do;
   salary70=salary70*0.78;
   salary80=salary80*0.67;
   salary90=salary90*0.87;
  end;
```

The DO statement causes all statements following it to be treated as a unit until a matching END statement appears. A group of SAS statements beginning with DO and ending with END is called a DO group.

Using the DO group makes the program faster to write and easier to read. It also makes the program more efficient for SAS in two ways:

1. The IF condition is evaluated fewer Times. (Although there are more statements in this example than in the preceding one, the DO and END statements require very few computer resources.

2. The conditions `sex = 'F'` and `sex = 'M'` are mutually exclusive, as condensing the multiple IF-THEN statements into two statements reveals. You can make the second IF-THEN statement part of an ELSE statement; therefore, the second IF condition is not evaluated when the first condition is true.

Performing the Same Action for a Series of Variables

Suppose, like in the previous section, you need to change several different variable's values for some given condition. But this time, suppose you need to perform the same action for each of the variables. For example, suppose you want to use the same adjustment for each of the three income variables depending on the person's sex. One way to do this is to write a series of IF-THEN statements, as follows:

```
   if sex = 'F' then salary70=salary70*1.11;
   if sex = 'M' then salary70=salary70*0.78;
   if sex = 'F' then salary80=salary80*1.11;
   if sex = 'M' then salary80=salary80*0.78;
   if sex = 'F' then salary90=salary90*1.11;
   if sex = 'M' then salary90=salary90*0.78;
```

The pattern of action is the same for each sex in their three IF-THEN statements; only the variable name is different. You can make the program easier to read by telling SAS to perform the same action several Times, changing only the variable affected. The technique is called array processing, and the following sections explain it in a three-step process:

1. grouping variables into arrays

2. repeating the action

3. selecting the current variable to be acted upon.

Grouping Variables into Arrays

To define an array, use an ARRAY statement. A simple ARRAY statement has the following form:

ARRAY *array-name {number-of-variables} variable-1 ... variable-n;*

The *array-name* is a SAS name you choose to identify the group of variables. The *number-of-variables*, enclosed in braces, tells SAS how many variables you are grouping, and *variable-1 ...*

variable-n lists their names. For example, the following SAS statement creates an array called SALARY for the three salary variables:

```
array salary {3} salary70 salary80 salary90;
```

Listing a variable in an ARRAY statement assigns the variable an extra name with the form *array-name{position}*, where *position* is the position of the variable in the list (1, 2, or 3 in this case). The position can be a number or the name of a variable whose value is the number. This additional name is called an array reference, and the position is often called the subscript. The previous ARRAY statement assigns SALARY70 the array reference SALARY{1}; SALARY80, SALARY{2}; and SALARY90, SALARY{3}. From that point in the data step, you can refer to the variable by either its original name or by its array reference. For example, the names SALARY80 and SALARY{2} are equivalent.

Repeating the Action

To tell SAS to perform the same action several times, use an iterative DO loop of the following form:

DO *index-variable* = 1 TO *number-of-variables-in-array*;
  SAS statements
END;

An iterative DO loop begins with an iterative DO statement, contains other SAS statements, and ends with an END statement. The loop is processed repeatedly (iteratively) according to the directions in the iterative DO statement. The iterative DO statement contains an *index-variable* whose name you choose and whose value changes in each iteration of the loop. In array processing, you usually want the loop to execute as many times as there are variables in the array; therefore, you specify the values of *index-variable* are 1 TO *number-of-variables-in-array*. By default, SAS increases the value of *index-variable* by 1 before each new iteration of the loop. When the number becomes greater than n*umber-of-variables-in-array*, SAS stops processing the loop. By default, SAS adds the index variable to the data set being created.

An iterative DO loop that processes three times and has an index variable named COUNT looks like this:

```
do count = 1 to 3;
  SAS statements
end;
```

Selecting the Current Variable

Now that you have grouped the variables and know how many times the loop will be processed, you can tell SAS which variable in the array to use in each iteration of the loop. Recall that variables in an array can be identified by their array references and that the subscript of the reference can be a variable name as well as a number. Therefore, you can write programming statements in which the index variable of the DO loop is the subscript of the array reference, in other words, array-name{index-variable}. When the value of the index variable changes, the subscript of the array reference (and, therefore, the variable referenced) also changes.

The following statement uses COUNT as the subscript of array references:

```
 if sex = 'F' then salary{count} = salary{count}*1.11;
```

You can place this statement inside an iterative DO loop. When the value of COUNT is 1, SAS reads the array reference as SALARY{1} and processes the IF-THEN statement on SALARY{1}, that is, SALARY70. The same thing happens when COUNT has the value 2 or 3. The complete iterative DO loop with array references and array statement looks like this:

```
   array salary{3} salary70 salary80 salary90;
   do count = 1 to 3;
      if sex = "F" then salary{count}=salary{count}*1.11;
      else if sex = "M" then salary{count}=salary{count}*0.78;
   end;
```

Exercise 9

Use array processing to multiply each of the yr variables by 2 for ID's 1-5 and divide each of the yr variables by 2 for ID's 11-15.

```
data unemploy;
   input id yr85-yr92;
   cards;
1  29  26  34  29  40  35  76  64
2  84  54  41  40 136  67  74  58
3  87  66  47  37  70  56  63  55
4  80  70  53  27  68  37  49  38
5  79  48  63  53  66  48  43  29
11 113  81  88  48  84  37  91  57
12 108  92  71  49  81  72  55  52
13 123 108  59  49 121 101  98  42
14  57  43  47  28 102  56  95  55
15  82  76  80  35  84  73  50  50
;
```

Using SET Statements

Until now the INPUT statement has described raw data stored on disk or as part of the SAS program. SAS also enables you to create a new SAS data set using observations from an existing SAS data set. To retrieve data from an existing data set, use the SET statement. Simply enter the word SET instead of the INPUT statement, CARDS statement, and data lines, or in place of the INPUT and INFILE statements.

In the example below, the DATA statement prompts SAS to create a new data set named CLASS2. The SET statement tells SAS to retrieve data from the existing data set CLASS. CLASS2 is an exact copy of CLASS.

```
   data class2;      set class;   run;
```

Reading Selected Observations

If you want to retrieve only selected observations from the original data set rather than all the data, use the subsetting IF statement:

```
   data class2;
      set class;
      /* subsetting IF statement */
      if age < 12;
      htsquar= height ** 2;
```

Note: The phrase marked with slashes and asterisks is a comment. Comment statements allow you to document your program, but they have no effect on processing. An alternative way of writing the above comment is:

```
  * subsetting IF statement;
```

The asterisk marks the beginning of the comment and the semicolon ends the comment.

Reading Selected Variables

SAS allows you to create a subset of a larger data set not only by excluding observations but also by specifying which variables you want the new data set to contain. In a DATA step you can use the SET statement and the KEEP= or DROP= data set options (or DROP or KEEP statements) to create a subset from by specifying which variables you want the new data set to include.

As an example, use the KEEP= data set option to create a SAS data set that contains only the variables NAME and AGE in the CLASS data set:

```
data class3;
 set class(keep=name age);
 run;
```

The KEEP= option must be enclosed in parentheses following the name of the SAS data set.

You can also use the KEEP statement to produce the same data set, as illustrated here:

```
data class3;
  set class;
  keep name age;
  run;
```

Use the DROP= option or DROP statement to create a subset of a larger data set when you want to specify which variables are being *excluded* rather than the ones being included. You may base your decision to use DROP or KEEP on which method allows you to specify fewer variables.

Concatenating SAS Data Sets

You can also use the SET statement to concatenate two or more SAS data sets one after the other into a single SAS data set. The number of observations in the SAS data set is the sum of the number of observations in the original data sets.

To concatenate two SAS data sets, simply list them in the SET statement. The observations from the first data set you name in the SET statement appear first in the new data set; the observations from the second data set follow those from the first data set, and so on. For example:

```
data all;
   set one two;
```

The data set ALL contains all the observations for data set ONE followed by all the observations for data set TWO.

You may want to concatenate data sets when not all variables are common to the data sets named in the SET statement. In this case, the new data set includes all variables from the SAS data sets named in the SET statement. Observations corresponding to variables not found in the original data set are assigned missing values.

Exercise 10

Using the PROBIT data from the previous exercise, create a new SAS data set named PROBIT2 that uses a subsetting IF statement to read in only observations with DOSE < 40 and drop the variable PREP from the newly created SAS data set.

Using a Value in a Later Observation

SAS can retain a value from the current observation to use in future observations with the RETAIN statement. When the processing of the data step reaches the next observation, the held value represents information from the previous observation. RETAIN has the following form:

RETAIN *variable-1 ... variable-n*;

Suppose you want to identify the oldest person in the CLASS data set. In order to do this with the RETAIN statement (there are other ways to do this, by the way), you need to compare the age in one observation with the age in the next observation, create a retained variable (named HOLDAGE, in the example that follows), and assign the current value of age to it. Therefore, in the next observation HOLDAGE contains the value of AGE from the previous observation, and you can compare its value to that of AGE:

```
data oldest;
   set class;
   retain holdage;
   if age > holdage then holdage=age;
run;
```

Below is the contents of OLDEST:

```
OBS    NAME       HEIGHT    WEIGHT    AGE    HOLDAGE
  1    Alfred      69.0      112.5     14        14
  2    Alice       56.5       84.0     13        14
  3    Barbara     65.3       98.0     13        14
  4    Carol       62.8      102.5     14        14
  5    Henry       63.5      102.5     14        14
  6    James       57.3       83.0     12        14
  7    Jane        59.8       84.5     12        14
  8    Janet       62.5      112.5     15        15
  9    Jeffrey     62.5       84.0     13        15
 10    John        59.0       99.5     12        15
 11    Joyce       51.3       50.5     11        15
 12    Judy        64.3       90.0     14        15
 13    Louise      56.3       77.0     12        15
 14    Mary        66.5      112.0     15        15
 15    Philip      72.0      150.0     16        16
 16    Radhika     50.2         .      13        16
 17    Robert      64.8      128.0     12        16
 18    Ronald      67.0      133.0     15        16
 19    Thomas      57.5       85.0     11        16
```

The value of HOLDAGE in the last observation is the oldest person in the class. But how do you know which person this corresponds to? Create a variable named HOLDNAME to hold the name of the person from the observations with the oldest age. Include HOLDNAME in the RETAIN statement to retain its value until explicitly changed:

```
data oldest;
   set class;
   retain holdage holdname;
   if age > holdage then do;
      holdage=age;
      holdname=name;
   end;
run;
```

Below is the contents of OLDEST:

```
OBS    NAME       HEIGHT    WEIGHT    AGE    HOLDAGE    HOLDNAME

  1    Alfred      69.0      112.5     14       14       Alfred
  2    Alice       56.5       84.0     13       14       Alfred
  3    Barbara     65.3       98.0     13       14       Alfred
  4    Carol       62.8      102.5     14       14       Alfred
  5    Henry       63.5      102.5     14       14       Alfred
  6    James       57.3       83.0     12       14       Alfred
  7    Jane        59.8       84.5     12       14       Alfred
  8    Janet       62.5      112.5     15       15       Janet
  9    Jeffrey     62.5       84.0     13       15       Janet
 10    John        59.0       99.5     12       15       Janet
 11    Joyce       51.3       50.5     11       15       Janet
 12    Judy        64.3       90.0     14       15       Janet
 13    Louise      56.3       77.0     12       15       Janet
 14    Mary        66.5      112.0     15       15       Janet
 15    Philip      72.0      150.0     16       16       Philip
 16    Radhika     50.2         .      13       16       Philip
 17    Robert      64.8      128.0     12       16       Philip
 18    Ronald      67.0      133.0     15       16       Philip
 19    Thomas      57.5       85.0     11       16       Philip
```

Writing Observations to Multiple SAS Data Sets

SAS allows you to create multiple SAS data sets in a single data step by specifying the data set names on the DATA statement and then using the OUTPUT statement to direct the observations to the appropriate data set. If you want to write observations to a selected data set, specify that data set name directly in the OUTPUT statement. Any name appearing in the OUTPUT statement must also appear in the DATA statement. When you use an OUTPUT statement without specifying a data set name, SAS writes the current observation to all the data sets named in the DATA statement.

For example, suppose you want to split the CLASS data set into two data sets - one for the younger persons and one for the older persons. Name both data sets in the DATA statement and select the observations with IF conditions. Use an OUTPUT statement in the THEN and ELSE clauses to output the observations to the data set you specify:

```
data young old;
   set class;
   if age <= 12 then output young;
   else output old;
run;
```

Exercise 11

Using the PROBIT data from Exercise 8, create two new data sets in one data step. One data set should contain only observations corresponding to PREP equal to `stand` and the other should contain only observations corresponding to PREP equal to `test.`

Merging SAS Data Sets

Merging combines observations from two or more SAS data sets into a single observation in a new SAS data set. The new data set contains all variables from all the original data sets unless you specify otherwise.

This chapter discusses two types of merging: one-to-one merging and match merging. In one-to-one merging, observations are combined based on their positions in the input data sets. In match merging, you combine observations from the input data sets based on common groups called BY groups in SAS.

One-to-One Merging

You merge data sets using the MERGE statement in a DATA step. The form of the MERGE statement is as follows:

MERGE *SAS-data-set-list*;

where *SAS-data-set-list* is a list of SAS data sets to merge.

In a simple one-to-one merge, SAS combines the first observation in all data sets you name in the MERGE statement into the first observation in the new data set, the second observation in all data sets into the second observation in the new data set, and so on. The number of observations in the new data set is equal to the number of observations in the largest data set you name in the MERGE statement.

Suppose you have another data set in addition to the CLASS data set discussed earlier that contains height and weight information for the same people a year later. The following program creates the data set CLASS2.

```
data class2;
   input height2 weight2 @@;
   cards;
70 130  55 84  67 100  64 104  66 105
60  99  66 90  64 115  65  91  59 100
53  70  65 92  64 100  80 130  74 130
```

```
   53  99  68 99  69 140  58  98  72 140
  ;
```

The following program performs a one-to-one merge of these data sets, adding the second year's data to the original data. The merged data set is called CLASSALL.

```
data classall;
   merge class class2;
run;
```

The following output shows the CLASSALL data set:

```
OBS    NAME       HEIGHT    WEIGHT    AGE    HEIGHT2    WEIGHT2

  1    Alfred      69.0      112.5     14       70        130
  2    Alice       56.5       84.0     13       55         84
  3    Barbara     65.3       98.0     13       67        100
  4    Carol       62.8      102.5     14       64        104
  5    Henry       63.5      102.5     14       66        105
  6    James       57.3       83.0     12       60         99
  7    Jane        59.8       84.5     12       66         90
  8    Janet       62.5      112.5     15       64        115
  9    Jeffrey     62.5       84.0     13       65         91
 10    John        59.0       99.5     12       59        100
 11    Joyce       51.3       50.5     11       53         70
 12    Judy        64.3       90.0     14       65         92
 13    Louise      56.3       77.0     12       64        100
 14    Mary        66.5      112.0     15       80        130
 15    Philip      72.0      150.0     16       74        130
 16    Radhika     50.2         .      13       53         99
 17    Robert      64.8      128.0     12       68         99
 18    Ronald      67.0      133.0     15       69        140
 19    Thomas      57.5       85.0     11       58         98
 20    William     66.5      112.0     15       72        140
```

The output shows that the new data set combines the first observation from CLASS with the first observation from CLASS2, the second from CLASS with the second from CLASS2, and so on.

The previous example illustrates the simplest case of a one-to-one merge: the data sets contain the same number of observations and all variables have unique names. Suppose now that the weight and height variables in CLASS2 were called HEIGHT and WEIGHT as in the CLASS data set. To preserve both sets of values, you would need to use the RENAME= data set option to rename the variables in one of the data sets as the following example illustrates:

```
data classall;
```

```
      merge class
            class2 (rename=(weight=weight2 height=height2));
   run;
```

Now suppose that CLASS2 does not contain its last observation. The following output illustrates what the merged data set, CLASSALL, would look like in this case:

```
OBS    NAME       HEIGHT    WEIGHT    AGE    HEIGHT2    WEIGHT2

  1    Alfred      69.0      112.5     14       70        130
  2    Alice       56.5       84.0     13       55         84
  3    Barbara     65.3       98.0     13       67        100
  4    Carol       62.8      102.5     14       64        104
  5    Henry       63.5      102.5     14       66        105
  6    James       57.3       83.0     12       60         99
  7    Jane        59.8       84.5     12       66         90
  8    Janet       62.5      112.5     15       64        115
  9    Jeffrey     62.5       84.0     13       65         91
 10    John        59.0       99.5     12       59        100
 11    Joyce       51.3       50.5     11       53         70
 12    Judy        64.3       90.0     14       65         92
 13    Louise      56.3       77.0     12       64        100
 14    Mary        66.5      112.0     15       80        130
 15    Philip      72.0      150.0     16       74        130
 16    Radhika     50.2         .      13       53         99
 17    Robert      64.8      128.0     12       68         99
 18    Ronald      67.0      133.0     15       69        140
 19    Thomas      57.5       85.0     11       58         98
 20    William     66.5      112.0     15        .          .
```

The number of observations in the new data set is equal to the number of observations in the largest data set you name in the MERGE statement. When SAS runs out of values for variables in the shorter data set, it fills in missing values for its variables.

Match Merging

In match-merging, you match observations according to the values of a particular variable(s). Before you can perform a match-merge, all data sets must be sorted by the variables you want to use for the merge. SAS uses a BY statement to group observations according to values of a particular variable. So, a BY statement in combination with a MERGE statement is how you accomplish match-merging. The form of the BY statement used for this purpose is:

BY *variable-list*;

where *variable-list* is the name of a variable(s) common to all the data sets being merged.

Suppose you have a third SAS data set, CLASS3, that contains the variable NAME plus another variable, FAM_INCM, which is the family's annual income. Suppose further that Jeffrey and Philip's data are missing from this data set (two less observations than CLASSALL). The following output shows the CLASS3 data set:

```
OBS    NAME        FAM_INCM

  1    Alfred         39
  2    Alice          56
  3    Barbara        45
  4    Carol          12
  5    Henry          63
  6    James          57
  7    Jane           59
  8    Janet          62
  9    John           59
 10    Joyce          51
 11    Judy           64
 12    Louise         56
 13    Mary           66
 14    Radhika        50
 15    Robert         64
 16    Ronald         67
 17    Thomas         57
 18    William        66
```

Because of these missing observations, a simple one-to-one merge would not work correctly. Some of the salary data would be merged with the wrong observation. With match-merging though, you can merge these data sets correctly.

The variable common to both CLASSALL and CLASS3 is NAME. Therefore, NAME is the appropriate variable to use in the BY statement. You can use the following SAS program to merge CLASSALL and CLASS3:

```
data clas_inc;
  merge classall class3;
  by name;
run;
```

The following output shows the resulting CLASS_INC data set:

```
OBS    NAME       FAM_INCM    HEIGHT    WEIGHT    AGE    HEIGHT2    WEIGHT2

  1    Alfred        39         69.0      112.5     14       70        130
  2    Alice         56         56.5       84.0     13       55         84
  3    Barbara       45         65.3       98.0     13       67        100
  4    Carol         12         62.8      102.5     14       64        104
  5    Henry         63         63.5      102.5     14       66        105
  6    James         57         57.3       83.0     12       60         99
  7    Jane          59         59.8       84.5     12       66         90
  8    Janet         62         62.5      112.5     15       64        115
  9    Jeffrey        .         62.5       84.0     13       65         91
 10    John          59         59.0       99.5     12       59        100
 11    Joyce         51         51.3       50.5     11       53         70
 12    Judy          64         64.3       90.0     14       65         92
 13    Louise        56         56.3       77.0     12       64        100
 14    Mary          66         66.5      112.0     15       80        130
 15    Philip         .         72.0      150.0     16       74        130
 16    Radhika       50         50.2         .      13       53         99
 17    Robert        64         64.8      128.0     12       68         99
 18    Ronald        67         67.0      133.0     15       69        140
 19    Thomas        57         57.5       85.0     11       58         98
 20    William       66         66.5      112.0     15       72        140
```

The new data set contains one observation for each person and each observation contains all the variables from both data sets. Notice in particular the ninth and fifteenth observations. Since the CLASS3 data set doesn't have observations for these people, the values of FAM_INCM unique to CLASS3 are missing.

Note that observations in the two data sets being merged were sorted similarly. This is a requirement when using the BY statement. If the observations had not be sorted, it would have been necessary to do so prior to the merging. The procedure SORT is how you sort SAS data sets. This procedure is discussed later in the "*Using SAS Procedures*" chapter.

Only a very simple match-merge was illustrated in this section. To learn more about merging, refer to "*SAS Language and Procedures: Usage*" Guide.

Reading Hierarchical Files

Often Times in the social sciences the data file to be read is hierarchical. A hierarchical file is a special case of a file that has multiple records having different formats. (Reading multiple records to create a single observation was discussed previously in the chapter, *"Creating SAS Data Sets*". In a hierarchical file, related records of different formats occur in record groups. Take the Current Population Surveys (CPS, for short) data for example. There are three different record types. The first record is the household record, followed by a family record, and then the people within family records. Next comes another family record, if one exists, followed again by the people within that family. Then, after the last person in the last family in a household, comes the records corresponding to the next household. This grouping is critical for the construction of the SAS data set.

There is more than one way to construct SAS data sets from hierarchical files. This chapter illustrates two methods using the CPS data: 1) Creating a single SAS data set, often called a "person level" data set, that combines information from all three record types, and, 2) Creating multiple SAS data sets, one for each of the record types, often referred to as the "household level" data set, the "family level" data set, and the "person level" data set. Both methods make use of conditional processing (which was discussed in the previous chapter) to read the records accurately.

Creating a Single SAS Data Set

Often Times when you read a hierarchical file like the CPS file you want to create a single SAS data set that has an observation corresponding to each person. Each observation in the data set also contains information pertaining to the corresponding household and family record. Usually you will only be interested in some subset of the data. For example, you may only want people between the ages of 25 and 65 and only a subset of the variables. Social scientists often refer to this as an extract. Variables to be read are specified using INPUT statements and the appropriate observations (people between the ages of 25 and 65) are selected with a subsetting IF statement. The end result is one SAS data set that contains variables on households, families, and persons.

The following sections explain how to read a hierarchical file like the CPS file and create a single SAS data set in a three-step process:

1) Identify the record layout

2) Conditionally read one of the three types of records

3) Write the observation to the data set after reading all the records corresponding to a household.

Identify the Record Layout

Begin the DATA step and name the data set to be created with a DATA statement. Identify the input file with an INFILE statement:

```
data hier;
   infile "~/dissert/cps0390.dat";
```

Before you can read all the fields in a record, you need to determine which kind of record you are trying to read. To do so, read a field whose value identifies the type of record being read, and assign that value to a variable. Hold this record while you decide how to read the rest of it by using a single trailing at sign (@). The trailing at sign was discussed previously in the *"Creating SAS Data Sets*" chapter.

In the CPS data the value in the first column reveals the layout of the rest of the record. The following INPUT statement reads this value and assigns it to the variable RECTYPE:

```
input rectype 1 @;
```

The next step is to test the value of RECTYPE and use the results of that test to determine how to read the rest of the record.

Conditionally Read One of the Records

After assigning to a variable the value that reveals the format of the record, test that value and conditionally execute the INPUT statement that correctly reads the rest of the record. This can be accomplished with IF/THEN/ELSE logic. In the following example, IF/THEN/ELSE logic is used to conditionally executes the correct INPUT statement to read a record corresponding to either a household, a family, or a person, based on the value of RECTYPE (1 for household, 2 for family, and 3 for person):

```
if rectype = 1 then do;
                input hhtype    20
                      hhsize    21-22
                      tenure    35
                      region    39
```

```
                    msa       58;
              retain hhtype hhsize tenure region msa;
       end;
       else if rectype = 2 then do;
              input famno     7-8
                    famtype   9
                    kidslt6   25;
              retain famno famtype kidslt6;
       end;
       else input perno     7-8
                    marstat   17
                    sex       20
                    ws        243-248;
      end;
```

Note the two RETAIN statements in the example above. By default, values read from the previous iterations are set to missing after each iteration. An observation, therefore, will contain only values for the last detail record read. For this reason, you must override the default action of the DATA step and control explicitly when values are retained with a RETAIN statement. This makes it possible to include the household and family variables in an observation created from the person record.

Write the Observation

The observation has now been constructed with the variables you defined. Before writing out the observation you may wish to subset the observations. For example, assume you only want persons between the ages of 25 and 65. The following subsetting IF statement would accomplish this:

```
     if (25 < age < 65);
```

When you use multiple iterations of the DATA step to gather the information necessary to produce a single observation, you cannot use the automatic output at the end of each iteration of the DATA step to produce the observations you want. You must explicitly write an observation when you have read a detail record in a record group. For this example, you need to output the observation after reading each person type record. Therefore, you can use an OUTPUT statement to create an observation each time RECTYPE is equal to 3, as the following SAS statement illustrates:

```
   if rectype = 3 then output;
```

This completes the three step process. Following is the complete data step including the necessary DCL statements to run this job on VMS:

```
data hier (drop=rectype);
     infile '~/dissert/cps0390.dat';
     input rectype 1 @;
     if rectype = 1 then do;
                   input hhtype    20
                         hhsize    21-22
                         tenure    35
                         region    39
                         msa       58;
                   retain hhtype hhsize tenure region msa;
          end;
          else if rectype = 2 then do;
                   input famno     7-8
                         famtype   9
                         kidslt6   25;
                   retain famno famtype kidslt6;
          end;
          else input perno     7-8
                         lineno    9-10
                         age       15-16
                         marstat   17
                         sex       20
                         higrade   22-23
                         complt    24
                         race      25
                         pstat     26
                         msupwgt   66-73 .2
                         union     139
                         weeksly   171-172
                         hoursly   181-182
                         source    242
                         ws        243-248;
     end;
     if (25 < age < 65);  /*  select only people 25 to 65  */
     if (rectype eq 3) then output;  /*  keep HH with persons
*/
run;
```

Creating Multiple SAS Data Sets

In the previous section you read the CPS file and created a single SAS data set that had an observation corresponding to each person. In this section, you will also read the CPS file but create three SAS data sets from it, one for each of the record types - a "household level" data set, a "family level" data set, and a "person level" data set.

You still follow the same three basic steps when constructing the DATA step: 1) identify the record layout, 2) conditionally read the records, and 3) write the observation. The only major difference between this data step and the previous data step which constructed one data set is in the placement of OUTPUT statements. In the case where you are creating a separate data set for each record type, you need to include an OUTPUT statement after each record is read in directing the observation to the appropriate data set. If you want the variables from the household record included in the family and person records you need to use the RETAIN statement as you did previously. Similarly, if you want the variables from family record included in the person record, you need to include the second RETAIN statement as well. The following example illustrates the complete data step:

```
data hh family person;
     infile '~/dissert/cps0390.dat';
     input rectype 1 @;
     if rectype = 1 then do;
                  input hhtype    20
                        hhsize    21-22
                        tenure    35
                        region    39
                        msa       58;
                  retain hhtype hhsize tenure region msa;
                  output hh;
          end;
          if rectype = 2 then do;
                  input famno     7-8
                        famtype   9
                        kidslt6   25;
                  retain famno famtype kidslt6;
                  output family;
          end;
          else do;
                  input perno     7-8
                        lineno    9-10
                        age       15-16
                        marstat   17
                        sex       20
                        higrade   22-23
                        complt    24
                        race      25
                         weeksly   171-172
```

```
                    hoursly   181-182
                    ws        243-248;
              output person;
              /*  select only people 25 to 65  */
              if (25 < age < 65);
end;     run;
```

Permanent SAS Data Sets

SAS creates two types of data sets: temporary and permanent. A temporary SAS data set exists only for the duration of the current SAS program. Therefore, data stored in temporary SAS data sets cannot be retrieved for use in later SAS programs; the data set must be recreated each time you run a new SAS program. A permanent SAS data set exists after the end of the current program. You do not need to repeat the DATA step every time you access the data in a SAS program if you save the data as a permanent SAS data set. A permanent SAS data set is a self-documented file containing the data AND descriptive information such as variable names and locations, variable labels, and so forth.

There are several advantages to working with permanent SAS data sets over temporary ones:

- ! SAS can read permanent SAS data sets faster than raw files.
- ! Permanent SAS data sets are self-documenting.
- ! Permanent SAS data sets reflect all modifications made to the data.

The major disadvantage of using SAS permanent data sets is that they cannot be read directly by software other than SAS.

Warning: Because permanent SAS data sets are stored in binary, and different hosts, you can not directly read a permanent version 6.12 SAS data set from one host that was created on another. (You will be able to beginning in version 7, however.) You must first convert the permanent SAS data set on the originating host to a format that can be recognized by the receiving host.

Data set names of the form CLASS and CLASS2 as in preceding examples, are temporary data sets. When assigning permanent data sets, you specify a two-level name such as SAVE.CLASS or RESEARCH.REPEATED. The first part of the name is called the libref. A libref is the name by which you reference the directory containing the SAS data sets. Actually, temporary data sets also have two-level names. SAS automatically assigns a libref of WORK to temporary data sets. For example, when you submit DATA STUDY; SAS creates a temporary data set named WORK.STUDY. Subsequently, you need use only the one-level data set name when referring to temporary SAS data sets, because SAS already assumes the default libref WORK.

When you create a permanent SAS data set, you must specify both the libref and the data set name. You must specify a libref other than WORK because SAS reserves that libref for temporary SAS data sets. Use a LIBNAME statement to assign a libref that tells SAS where to find the directory that contains or will contain the permanent SAS data sets.

For the SAS data set, SAVE.CLASS, an example LIBNAME statement might be similar to the one below:

```
   libname save '~/sasdata';
```

The keyword LIBNAME is followed by the libref (the first-level of the two-level SAS data set name). The quoted string is the directory containing the SAS data sets you want to access.

Below is an example that creates a permanent sas data set and places it in the [.sasdata] directory:

```
   libname save '~/sasdata';
   data save.class;
      infile "class.dat";
      input name $ 1-7 height 9-12 weight 14-18 age 20-21;
      ht_cm = height * 2.54;
      if age < 12 then agegroup=1;
      if age >= 12 then agegroup=0;
   run;
```

When referring to a permanent SAS data set, use the full two-level name, that is, both the libref and the data set name. For example, assume you have submitted the SAS code in the example above to create a permanent SAS data set. In order to obtain a plot of WEIGHT by HEIGHT in a later SAS program, you could submit the following code:

```
   libname retrieve '~/sasdata';
   proc plot data=retrieve.class;
      plot weight * height;
   run;
```

Note that a DATA step is not needed here because the data were read into a permanent SAS data step in the previous SAS program. It is only necessary to specify a LIBNAME statement and then refer to the data set in the PROC step.

Storing Permanent SAS Data Sets Efficiently

At some point you may need to think about the storage space your SAS data set occupies. A permanent SAS data set is significantly larger than the data in raw form because the file not only contains the data but also all of its descriptive information. One method of reducing storage requirements which was discussed earlier in the "*Modifying Data using SAS Statements*" chapter was to use a LENGTH statement to store variables as efficiently as possible.

Another method of reducing the storage requirements is to have SAS compress your SAS data set. When you have SAS store your data set compressed the observations in the data set being created are saved as variable-length records rather than the default fixed-length. Compressing a data set reduces its size by reducing consecutive characters or numbers to 2-byte or 3-byte representations.

The COMPRESS=YES data set option species that observations in the data set being created are to be compressed, as the following example illustrates:

```
   data save.bigguy (compress=yes);
```

Once a data set is compressed, this becomes a permanent attribute of the data set. To uncompress the observations in the data set, you must use a SAS data step to copy the data set and use COMPRESS=NO for the new data set:

```
   data save.uncomp (COMPRESS=NO);
      set save.bigguy;
```

Another advantage of using the COMPRESS=YES data set option is that fewer input/output operations are required to read from or write to the data set during processing.

Data can be compressed even further using UNIX's compress facility. As an example of how much can be saved, the March 1991 CPS file can be compressed from 197,107,884 bytes to 18,203,535 bytes, over 90% reduction! UNIX compression is described in detail in SSCC Publication #7-5, *"Using compressed files"* .

Exercise 12

1. Write and execute a SAS program that creates a permanent SAS data set from the PROBIT data of Exercise 10.

2. Write a second SAS program that uses the SAS data set you created above and prints out observations in the data set.

Using SAS Procedures

Once you have created a SAS data set, SAS can analyze, process, and display data using SAS procedures, or PROC's. SAS procedures are pre-written computer programs that analyze and process data sets. These procedures read SAS data sets, process the data, and then display results. PROC steps always begin with a PROC statement, which specifies the name of the SAS procedure you want to run.

To run a SAS procedure, enter the keyword PROC, the name of the procedure, and any optional statement options, followed by a RUN statement. For example, the following statements invoke the PRINT procedure:

```
proc print;
run;
```

The above example is a proc step in its simplest form. An example of a more complex proc step follows:

```
proc means data=class;
   var height weight;
   where age >12;
   by sex;
run;
```

This example illustrates four ways you control which variables and observations get processed. First, the DATA=CLASS option on the proc statement is how you select which SAS data set you want processed. Second, to analyze and process specific variables, use a SAS statement like VAR to list the variables you want processed. The VAR statement is also used to specify the order in which you want the variables processed.

Third, the WHERE statement enables you to process selected observations from a data set based on a specified condition. Hence, the WHERE statement is similar in action to a subsetting IF statement used in the DATA step. To define a condition, enter the keyword WHERE followed by the conditions the observations should meet. Combine multiple conditions with SAS operators.

Fourth, the BY statement allows you to process your data in separate, distinct groups. In the above example, the summary statistics is computed for each sex separately as illustrated below:

```
----------------------------------- SEX=F ----------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     --------------------------------------------------------------------
     HEIGHT     7    61.1571429     5.7999589    50.2000000    66.5000000
     WEIGHT     6    99.8333333    11.5441183    84.0000000   112.5000000
     --------------------------------------------------------------------

----------------------------------- SEX=M ----------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     --------------------------------------------------------------------
     HEIGHT     6    66.7500000     3.5035696    62.5000000    72.0000000
     WEIGHT     6   115.6666667    23.1466340    84.0000000   150.0000000
     --------------------------------------------------------------------
```

Before you can use a BY statement in a PROC step, you must first sort the data. Sorting data arranges observations in order of the values of specified variables. PROC SORT is used to sort data in SAS. The following example sorts the CLASS data by SEX:

```
   proc sort data=class;
      by sex;
   run;
```

Note that if the data were already in sort order by SEX, you would not need to run PROC SORT.

Exercise 13

Obtain means for the variable N of the PROBIT data set used in previous exercises for observations with dose greater than 10. Then get the mean for the three SYMPTOMS groups separately.

Making Output Informative

Any time you generate output, you can make it more informative by adding titles and labels.

Titles

You can add up to ten titles at the top of your output using TITLE statements in your PROC calls. For example, the following statements produce output with titles on the first, third, and fourth lines:

```
proc means data=class;
   title 'Height-Weight Study';
   title3 'Students Greater than 12 Yrs. Old';
   title4 '1993 Run';
   var height weight;
   where age >12;
   by sex;
run;
```

The results are shown below:

```
                             Height-Weight Study                            4

                     Students Greater than 12 Yrs. Old
                                  1993 Run

----------------------------------- SEX=F ----------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     -------------------------------------------------------------------
     HEIGHT     7    61.1571429     5.7999589    50.2000000    66.5000000
     WEIGHT     6    99.8333333    11.5441183    84.0000000   112.5000000
     -------------------------------------------------------------------

----------------------------------- SEX=M ----------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     -------------------------------------------------------------------
     HEIGHT     6    66.7500000     3.5035696    62.5000000    72.0000000
     WEIGHT     6   115.6666667    23.1466340    84.0000000   150.0000000
     -------------------------------------------------------------------
```

Note that TITLE and TITLE1 are equivalent keywords.

To cancel a title for a specific line and all title lines beneath it, enter the keyword TITLE followed by the appropriate line number. This statement is often referred to as a null TITLE statement. The following null TITLE statement cancels titles on the third line and after:

```
proc means data=class;
   title3;
   var height weight;
   where age >12;
   by sex;
run;
```

The results are shown below:

```
                             Height-Weight Study                             5
------------------------------------ SEX=F ------------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     --------------------------------------------------------------------
     HEIGHT     7    61.1571429     5.7999589    50.2000000    66.5000000
     WEIGHT     6    99.8333333    11.5441183    84.0000000   112.5000000
     --------------------------------------------------------------------

------------------------------------ SEX=M ------------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     --------------------------------------------------------------------
     HEIGHT     6    66.7500000     3.5035696    62.5000000    72.0000000
     WEIGHT     6   115.6666667    23.1466340    84.0000000   150.0000000
     --------------------------------------------------------------------
```

Variable Labels

In procedure output, SAS automatically prints the variables with the names you specify. However, you can label some or all of your variables by specifying a LABEL statement either in the DATA step or, with some procedures, in the PROC step. Your label can be up to 40 characters long, including blanks. Suppose you want to describe the variable WEIGHT with the phrase 'weight in pounds'. Simply specify

```
label weight = 'Weight in Pounds';
```

The following example labels variables in a PROC step:

```
proc means data=class;
   title 'Height-Weight Study';
```

```
      title2 'Students Greater than 12 Yrs. Old';
      title3 '1993 Run';
      var height weight;
      label height= 'Height in inches';
      label weight = 'Weight in Pounds';
      where age >12;
   run;
```

The results are shown below:

```
                           Height-Weight Study                             6
                     Students Greater than 12 Yrs. Old
                                  1993 Run

   Variable  Label              N          Mean       Std Dev       Minimum
   ------------------------------------------------------------------------
   HEIGHT    Height in inches  13    63.7384615     5.5096186    50.2000000
   WEIGHT    Weight in Pounds  12   107.7500000    19.2996703    84.0000000
   ------------------------------------------------------------------------

                   Variable  Label                  Maximum
                   ----------------------------------------
                   HEIGHT    Height in inches    72.0000000
                   WEIGHT    Weight in Pounds   150.0000000
                   ----------------------------------------
```

If you specify the LABEL statement in the DATA step, the label is permanently stored in the data set. If you specify the LABEL statement in the PROC step, the label is associated with the variable for the duration of the PROC step only. In either case, when a label is assigned, it is printed with almost all SAS procedures. The exception is the PRINT procedure. To use variable labels with the PRINT procedure you must specify the LABEL option as follows:

```
   proc print label;
   run;
```

Note that specifying a label in the PROC step overrides any labels stored in the data set.

Value Labels

Labeling values is not as straightforward as labeling variables. To label values, use PROC FORMAT with the VALUE statement as in the example below:

```
   proc format;
      value answer 1='never' 2='sometimes' 3='always';
      value $sexfmt 'F'='Female' 'M'='Male';
```

The PROC FORMAT statement invokes the FORMAT procedure, which creates a format. The VALUE statement defines the format so the values of the variable can be associated with the newly formatted values. The newly created values can be up to 40 characters and must be enclosed in quotes.

In this example, the first VALUE statement defines a format named ANSWER. that converts numeric values to character values. The second VALUE statement defines a format named $SEXFMT. that substitutes one character string for another. Note that the $SEXFMT. starts with a "$" because it is for character variables, and that both ranges and labels are enclosed in apostrophes.

After you have created a format, you can use a FORMAT statement to associate the values of the format with the appropriate variable. The general form of the FORMAT statement is as follows:

```
   FORMAT variables format.;
```

where variables is the variable(s) you want value labels for and format is the name of the format you assigned in the VALUE statement of PROC FORMAT. Note that a period (.) must follow the format name. Below the format $SEXFMT. is applied to variable SEX:

```
   proc means data=class;
      title3;
      var height weight;
      where age >12;
      format sex $SEXFMT.;
      by sex;
   run;
```

The above statements produce the following output:

```
                             Height-Weight Study                            7
---------------------------------- SEX=Female -------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
     -------------------------------------------------------------------
     HEIGHT     7    61.1571429     5.7999589    50.2000000    66.5000000
     WEIGHT     6    99.8333333    11.5441183    84.0000000   112.5000000
     -------------------------------------------------------------------

----------------------------------- SEX=Male --------------------------------

     Variable   N          Mean       Std Dev       Minimum       Maximum
```

```
    -----------------------------------------------------------------
    HEIGHT     6    66.7500000     3.5035696    62.5000000    72.0000000
    WEIGHT     6   115.6666667    23.1466340    84.0000000   150.0000000
    -----------------------------------------------------------------
```

Note that formats assigned in PROC steps as above only remain in effect for that PROC call. Formats assigned in a DATA step, however, remain in effect for the entire SAS program.

You can also use PROC FORMAT to recode numeric values into categories:

```
   proc format;
      value agef 10-12='youngest' 13-14='middle'
15-16='oldest';
   run;
   proc means;
      format age agef.;
      class age;
      var height weight;
  run;
```

The above code produces the following output:

```
       AGE  N Obs  Variable   N          Mean       Std Dev       Minimum
   --------------------------------------------------------------------
   youngest      7  HEIGHT     7    58.0000000     4.0620192    51.3000000
                    WEIGHT     7    86.7857143    23.4358293    50.5000000

   middle        8  HEIGHT     8    61.7625000     5.8230913    50.2000000
                    WEIGHT     7    96.2142857    10.6804227    84.0000000

   oldest        5  HEIGHT     5    66.9000000     3.3800888    62.5000000
                    WEIGHT     5   123.9000000    17.1551741   112.0000000
   --------------------------------------------------------------------
```

Exercise 14

Use PROC FORMAT to create value labels for each of the DOSE levels. Then use PROC PRINT to display the data set. Enhance your output with titles and variable labels.

SAS Procedures for Verifying your SAS Data Set

In the chapter, "*Running your SAS Program*", you examined methods for verifying SAS data sets utilizing error messages SAS prints out when it detects errors in the data. Unfortunately, though there are many other kinds of data errors that you can only uncover by examining your data. In this chapter, you will learn how to examine your data with the PRINT, PLOT, CONTENTS, and FREQ procedures. Other SAS procedures useful for examining data but which are not discussed in this section include MEANS, SUMMARY, UNIVARIATE, COMPARE, and INSIGHT.

The PRINT Procedure

PROC PRINT allows you to examine the actual data values in a data set in order to verify that the data were entered correctly. For example, the following statements cause SAS to display all the observations in the CLASS data set:

```
   proc print data=class; run;
```

PROC PRINT, in combination with the VAR, WHERE, TITLE, and BY statements described above, allow you to enhance your printed output. Often times though your data set is too large to display in its entirety. What is often done in this case is to request SAS to display the first few observations in the data set by specifying the data set option OBS= on the PROC PRINT statement as illustrated below:

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

The above statement produces the following output:

```
                 Height-Weight Study                              8
  OBS    NAME        HEIGHT    WEIGHT    AGE    SEX
    1    Alice        56.5       84.0     13     F
    2    Barbara      65.3       98.0     13     F
    3    Carol        62.8      102.5     14     F
    4    Jane         59.8       84.5     12     F
    5    Janet        62.5      112.5     15     F
```

The CONTENTS Procedure

PROC CONTENTS gives you another way to look at a SAS data set. Use it to display information that describes the structure of a SAS data set rather than the data values:

```
   PROC CONTENTS DATA=CLASS; RUN;
```

The above statement produces the following output:

```
                              CONTENTS PROCEDURE

Data Set Name: WORK.ONE                              Observations:         20
Member Type:   DATA                                  Variables:            4
Engine:        V607                                  Indexes:              0
Created:       9:30 Wednesday, March 10, 1993        Observation Length:   32
Last Modified: 9:30 Wednesday, March 10, 1993        Deleted Observations: 0
Protection:                                          Compressed:           NO
Data Set Type:                                       Sorted:               NO
                 -----Engine/Host Dependent Information-----

  Max Obs per Page:         254
  Obs in First Data Page:   20
  Filename:                 $1$DIA13:[SCRATCH.SAS$WORK23000E84]ONE.SASEB$DATA
Disk Blocks Allocated:    20

            -----Alphabetic List of Variables and Attributes-----
                      #    Variable    Type    Len    Pos
                      -----------------------------------
                      4    AGE         Num       8     24
                      2    HEIGHT      Num       8      8
                      1    NAME        Char      8      0
                      3    WEIGHT      Num       8     16
```

PROC CONTENTS reports:

- the number of variables and observations
- the name, type, and length of each variable
- the position of the variable in the observation
- the format and label for each variable, if they exist
- certain operating system-specific details.

Plotting Data using the PLOT Procedure

Plotting data enables you to graphically illustrate the relationship between variables. To produce a simple plot of one set of variables, enter a PROC PLOT statement followed by a PLOT statement. The PROC PLOT statements instructs SAS which data set to use and the PLOT statement specifies the variables you want plotted. For example:

```
PROC PLOT DATA=CLASS;
   PLOT WEIGHT*HEIGHT;
RUN;
```

Note that the two variables you want plotted are joined by an asterisk (*) with the variable you want plotted on the vertical axis appearing to the left of the asterisk and the variable you want plotted on the horizontal axis appearing to the right of the asterisk.
The above SAS statements produce the following output:

```
                          Height-Weight Study                         11
       Plot of WEIGHT*HEIGHT.  Legend: A = 1 obs, B = 2 obs, etc.

   150 +                                                      A
       |
       |
       |                                          A
WEIGHT |                                     A
       |
       |
       |                               A         B     A
       |
   100 +                       A        AA    A
       |
       |                                   A
       |                 A B     A     A
       |                A
       |
       |
       |
       |
    50 +    A
       --+-----------+-----------+-----------+-----------+-----------+--
        50          55          60          65          70          75
                                     HEIGHT
```

Notice that PROC PLOT automatically selects the plotting symbol A to represent one occurrence at each point. If two occurrences coincide at a point, the plotting symbol B is used; and so forth. PROC PLOT also selects ranges for both axes, places tick marks at reasonably spaced intervals, and prints a legend that names the variables and explains the plotting symbols.

Although PROC PLOT determines many plot characteristics by default, you can override these defaults and specify selections of your own. For example, to specify plotting symbols of your choice replace the above PLOT statement with the following:

```
   PLOT WEIGHT*HEIGHT='*';
```

When you specify a plotting symbol, PROC PLOT uses that symbol for all points on the plot regardless of how many points coincide. If points coincide, a message appears at the bottom of the plot telling you how many observations are hidden.

There are many other ways to customize your plots with PROC PLOT. Refer to '*The SAS Procedures Guide*'' for details.

Generating Frequency and Cross tabulation Tables using PROC FREQ

PROC FREQ produces frequency tables and Cross tabulation tables which illustrate the frequency at which individual values or combinations of values occur within a SAS data set.

Frequency tables summarize data by displaying the frequency count, or how often the value of a variable occurs in a SAS data set. These tables also display the percent, cumulative frequency, and cumulative percent of each data value. The following example produces a frequency table:

```
   PROC FREQ DATA=CLASS;
      TABLES AGE;
   RUN;
```

The above statements produce the following output:

```
                          Height-Weight Study                         12

                                         Cumulative  Cumulative
           AGE   Frequency   Percent   Frequency    Percent
           ------------------------------------------------
            11          2      10.0           2       10.0
            12          5      25.0           7       35.0
            13          4      20.0          11       55.0
            14          4      20.0          15       75.0
            15          4      20.0          19       95.0
            16          1       5.0          20      100.0
```

The TABLES statement also allows you to specify options that suppress specific categories of output. NOCUM suppresses the cumulative statistics and NOPERCENT suppresses the display of the percent column. To use either of these options enter a slash (/) after the last variable on the table statement followed by the options as in the following example:

```
   PROC FREQ DATA=CLASS;
      TABLES AGE WEIGHT / NOCUM NOPERCENT;
   RUN;
```

You can describe data further with a Cross tabulation table. A Cross tabulation table is a frequency table that displays the frequency distribution for two or more variables.

To produce a Cross tabulation table, enter a TABLES statement containing the keyword TABLES, followed by the name of the variables you want to process, separated by an asterisk (*). The values of the first variable listed form the rows of the table; the second variable form the columns. For example:

```
   PROC FREQ;
      TABLES SEX*AGE;
   RUN;
```

The above statements produce the following output:

```
                         Height-Weight Study                        13

                          TABLE OF SEX BY AGE

SEX       AGE

Frequency|
Percent  |
Row Pct  |
Col Pct  |      11|      12|      13|      14|      15|      16|  Total
---------+--------+--------+--------+--------+--------+--------+
F        |      1 |      2 |      3 |      2 |      2 |      0 |     10
         |   5.00 |  10.00 |  15.00 |  10.00 |  10.00 |   0.00 |  50.00
         |  10.00 |  20.00 |  30.00 |  20.00 |  20.00 |   0.00 |
         |  50.00 |  40.00 |  75.00 |  50.00 |  50.00 |   0.00 |
---------+--------+--------+--------+--------+--------+--------+
M        |      1 |      3 |      1 |      2 |      2 |      1 |     10
         |   5.00 |  15.00 |   5.00 |  10.00 |  10.00 |   5.00 |  50.00
         |  10.00 |  30.00 |  10.00 |  20.00 |  20.00 |  10.00 |
         |  50.00 |  60.00 |  25.00 |  50.00 |  50.00 | 100.00 |
---------+--------+--------+--------+--------+--------+--------+
Total           2        5        4        4        4        1       20
            10.00    25.00    20.00    20.00    20.00     5.00   100.00
```

The above output is a default Cross tabulation table containing frequency, percent, row percent, and column percent statistics. The table consists of blocks of data, also known as cells, where the rows and columns intersect. Each cell contains the four statistics listed above. You can also request other statistics by adding options to the TABLES statement in the manner described above. For example:

```
   PROC FREQ;
      TABLES SEX*AGE / CHISQ CELLCHI2;
   RUN;
```

CHISQ performs chi-square tests of homogeneity or independence, and computes measures of association based on chi-square. CELLCHI2 prints each cell's contribution to the total chi-square statistic.

The above statements produce the following output:

```
                    Height-Weight Study                         15

                    TABLE OF SEX BY AGE

     SEX             AGE

     Frequency      |
     Cell Chi-Square|
     Percent        |
     Row Pct        |
     Col Pct        |      14|      15|      16|  Total
     ---------------+--------+--------+--------+
     F              |      2 |      2 |      0 |     10
                    |      0 |      0 |    0.5 |
                    |  10.00 |  10.00 |   0.00 |  50.00
                    |  20.00 |  20.00 |   0.00 |
                    |  50.00 |  50.00 |   0.00 |
     ---------------+--------+--------+--------+
     M              |      2 |      2 |      1 |     10
                    |      0 |      0 |    0.5 |
                    |  10.00 |  10.00 |   5.00 |  50.00
                    |  20.00 |  20.00 |  10.00 |
                    |  50.00 |  50.00 | 100.00 |
     ---------------+--------+--------+--------+
     Total                 4        4        1       20
                       20.00    20.00     5.00   100.00
```

```
                Height-Weight Study                         16
                                16:17 Wednesday, March 3, 1993

         STATISTICS FOR TABLE OF SEX BY AGE

Statistic                     DF     Value        Prob
------------------------------------------------------
Chi-Square                     5     2.200       0.821
Likelihood Ratio Chi-Square    5     2.634       0.756
Mantel-Haenszel Chi-Square     1     0.095       0.758
Phi Coefficient                      0.332
Contingency Coefficient              0.315
Cramer's V                           0.332

Sample Size = 20
WARNING: 100% of the cells have expected counts less
           than 5. Chi-Square may not be a valid test.
```

SAS also enables you to create n-way tables by connecting all the variable names in the TABLES statement with asterisks. Values of the last variable form the columns of the table; values of the next-to-last variable form the rows. Each level (or combination of levels) of other variables form one stratum, and a separate contingency table is produced for each stratum. For example, the following statements produce two tables, one for males and one for females:

```
PROC FREQ;
   TABLES SEX*WEIGHT*AGE;
RUN;
```

Multi-way tables can generate a lot of printed output. For example, if the variables A, B, C, D, and E each have ten levels, five-way tables of A*B*C*D*E could generate 4000 or more pages of output.

Exercise 15

1. Create a 2-way frequency table using the PROBIT data. Make PREP the rows and DOSE the columns. Request chi-square statistics.

2. Create a 3-way table adding SYMPTOMS as the stratum variable.

References

SAS Language and Procedures: Introduction, Version 6

SAS Language and Procedures: Usage, Version 6

SAS Language and Procedures: Usage 2, Version 6

SAS Companion for the UNIX Environment and Derivatives, Version 6

SSCC has a complete set of SAS documentation. These documents are circulated by the CDE Print/Virtual Library in 4457 Social Science. There are also numerous handouts written by SSCC staff that you may find useful. These include:

Research Computing on SSCC UNIX Systems

Using SAS on UNIX (SSCC Pub. 7-4)

Using SAS to Perform a Table Lookup (SSCC Pub. 4-1)

Constructing Indicator Variables with SAS (SSCC Pub. 4-2)

SAS Programming Efficiencies (SSCC Pub. 4-3)

Converting a Code Book to a SAS FORMAT Library (SSCC Pub 4-4)

Using SAS to Reformat Data Records from One to Several (SSCC Pub 4-5)

How to Write/Read UNIX Compressed SAS Data Sets Directly (#1)

How to Write a Macro to Transport SAS Data Sets from VMS to UNIX (#7)

How to Transport SAS Data Sets from VMS to UNIX (#17)

How to Transfer SAS Data Sets from VMS to UNIX using SAS Version 7 (#22)

How to Transfer SAS and SPSS system files between 3480 and disk (#23)

How to Print SAS/GRAPH Output (#26)

How to Read Compressed SAS Transport files Directly in your SAS Program (#28)

How to Avoid Running out of Disk Space when using Statistical Software on UNIX (#30)

How to Convert an SPSS Save File into a SAS Data Set (#38)

How to Transport SAS Data Sets between a PC and SSCC Computers (#42)

SSCC Publications are available from the Consultant, either Public Terminal Room (Social Science 2470 and 7413), or online at SSCC's web pages: http//www.ssc.wisc.edu/.

Solutions to Exercises

Exercise 1:

1.
```
proc glm data=wgtloss;
   class sex age;
   model wgt = sex age;
run;
```

2. Census90
   CEN_90

Exercise 3:

```
input drug $ answer $ wk1-wk4;

input drug $ 1-8 answer $ 10 wk1 13-15 wk2 17-20
      wk3 22-25 wk4 27-30;
```

Exercise 4:

```
input drug $ 1-10 answer $ 11 wk1-wk4;
```

Exercise 5:

```
DATA PROBIT;
INPUT dose 11-12 @;
if dose < 40;
INPUT prep $ dose symptoms $ n;
CARDS;
```

Exercise 6:

```
DATA HOUSE;
   INFILE 'multiple.dat';
```

```
      INPUT HHTYPE HHSIZE REGION
         #2 KIDSLT6 WEEKSLY;
```

Exercise 7:

```
   DATA PROBIT;
      length prep $ 5 dose n 2 symptoms $ 6;
      INPUT prep $ dose symptoms $ n;
      ldose = LOG10(dose);
      newdose=dose/10;
   CARDS;
```

Exercise 8:

```
   DATA PROBIT;
      INPUT prep $ dose symptoms $ n;
      if symptoms = "None" then dum1=1; else dum1=0;
      else if symptoms = "Mild" then dum2=1; else dum2=0;
  CARDS;
```

Exercise 9:

```
   data unemploy;
      input id yr85-yr92;
      array years {8} yr85-yr92;
      do i=1 to 8;
         if id <6 then years{i} = years{i} * 2;
         else if id >5 then years{i} = years{i} / 2;
      end;
```

Exercise 10:

```
   data probit2 (drop=prep);
      set probit;
      if dose < 40;
   run;
```

Exercise 11:

```
   data stand test;
      set probit;
      if prep = 'stand' then output stand;
         else if prep = 'test' then output test;
   run;
```

Exercise 12:

1.

```
   libname save '~/sasstuff';
   DATA SAVE.PROBIT;
      INPUT prep $ dose symptoms $ n;
      if dose lt 40;
      ldose = LOG10(dose);
      if prep = 'test' then prepdose = ldose;
         else prepdose = 0;
   CARDS;
```

2.

```
   libname retrieve '~/sasstuff';
   proc print data=retrieve.probit; run;
```

Exercise 13:

```
  proc means data=probit;
     where dose > 10;
     var n;
  run;

  proc sort data=probit;
     by symptoms;
  run;
  proc means data=probit;
     where dose >10;
     var n;
     by symptoms;
  run;
```

Exercise 14:

```
   proc format;
      value dosefmt 10='low' 20='med' 30='high' 40='very
high';
```

```
   run;

   proc print data=probit;
      format dose dosefmt.;
      label n "total";
      title 'data for probit analysis';
   run;
```

Exercise 15:

1.

```
   proc freq data=probit;
      tables prep*dose / chisq;
   run;
```

2.

```
   proc freq data=probit;
      tables symptoms*prep*dose;
   run;
```