THE MCHP SAS PC TUTORIAL

September 14, 2005

TABLE OF CONTENTS

THE MCHP SAS PC TUTORIAL

The SAS system provides a way of creating and/or accessing a variety of data sets, with techniques for manipulating the data to obtain output ranging from simple frequency tables to complex three-dimensional graphs. SAS software is available from the University of Manitoba for employees and students; more detailed information regarding SAS is available from the SAS website.

The goal of the MCHP online SAS tutorial is to provide the new user with enough knowledge of SAS to translate basic research questions into SAS code, enabling completion of the research project required by the Epidemiology of Health Care course at the University of Manitoba. Additional "intermediate" training material has been developed for new users of the MCHP data bases; this documentation covers arrays, do loops, first/last by-group processing, retain statements, and how to work with dates. (Complete SAS Institute manual documentation is available online under Academic Software References, Vendor Reference Materials, on the Software Team Resources Page, recognized with University of Manitoba IP addresses.)

The MCHP online SAS tutorial is best viewed using at least Version 3 of Netscape or Internet Explorer, with a minimum setting on an SVGA monitor of 256 colours, 800x600 pixels. It is intended for use with SAS Release 9.1 (Windows 95/98 or Windows NT operating system). In addition to the usual system Help menu, this version of SAS provides additional information to licensees under the Help headings of "SAS OnlineDoc" and "Online Tutor".

As a self-guided tutorial, it is suggested that the user review the general guidelines and SAS program syntax first. The remaining material might then be followed sequentially: how to prepare a data set, how to view the data, how to explore and manipulate the data, how to add observations and variables to a data set, and how to process the data. (If the user runs SAS and the browser simultaneously, example code can be copied from the browser to the Program Editor window in SAS.)

Several sample data sets are referenced:

1. **Height/weight -** is used throughout for illustrative purposes.
2. **Simulated clinical -** can be created by the user to complete the questions found at the end of the sections on viewing, exploring, manipulating, adding variables and observations, and processing the data (with links to how the program, log, and output should look).
3. **Simulated Manitoba Health -** the data set used by students of the Epidemiology of Health Care course to complete the required assignments, obtainable from MCHP. Additional questions are included here, for use with this data set, and complete with links to the program, log, and output.

SAS has improved its interface to the point where a lot of analysis can be simply and quickly carried out using menus, examples of which are provided in the alternatives to programming

section. It is often desirable, however, for users to have a basic understanding of SAS programming. Knowledge of SAS programming facilitates not only spotting some of the pitfalls inherent in processing data, but also maintaining more complete documentation of all the steps required to produce any given output. It is all too frequently necessary, in the course of carrying out research, to reproduce results which may have required a complex series of data processing steps.

It is important to recognize that there is often no one "right" way of obtaining accurate results. For simplification and continuity, this document reflects one style of writing SAS code. There are other more or less efficient ways of constructing SAS code, however, all of which may produce identical results. Where alternatives exist for generating results, a legitimate, and often preferred, choice is code that the user understands.

Resources, References and Acknowledgments

GENERAL GUIDELINES

a) Windows in SAS

This section provides a brief overview of the SAS windows environment, of how to navigate within and among windows. Navigation ***commands*** can be carried out in several ways:

- ***Command line*** - the command can be typed in the command box at the upper left of the screen, or window-specific commands can be typed on a command line (invoked using *Tools/Options/Preferences/View* in the SAS menu). This method tends to be used less frequently because of the ease of use of the other, newer, ways of navigating SAS windows.

- ***Pull-down menus*** - invoked by left-clicking on a menu heading. Different menu options are available, depending on which window is activated.

- ***Pop-up menus*** - invoked by right-clicking on a window. Most windows will have this option. They provide a useful way of accessing Help menus.

- ***Keys window*** - assigns commands to keys/key combinations (see description of Keys window below). Right clicking on this window will display a pop-up menu which includes a Help option; *Help/Using This Window* can be accessed to obtain a list of examples of key commands.

- ***Toolbars*** - icons that provide shortcuts for a number of commands; toolbars can be customized by going to the *Tools/Customize* menu. The Programming Windows icon on the toolbar, for example, can be used to restore the windows display to their default setting.

When SAS is invoked five windows are automatically opened, displaying labeled bars below each window. Left-clicking on any of the five bars will activate the window. This is important for carrying out any of the commands associated with a window.

The five windows are:

1. **Program editor** - is used for creating and editing SAS programs (and other kinds of ASCII files). Such tasks can also be accomplished with the **Notepad** window (opened through the Explorer window *File/New/Source Program*). SAS does not do anything with material entered here until the program is submitted for processing (*Submit/Run* on the Program Editor menu, or the runner on the toolbar).

2. **Log** - displays information, once a SAS program is submitted for processing, about how the program ran, or processed the information. It is extremely important to examine the SAS log and to understand its messages in order to ensure that output has been generated as instructed; they facilitate any debugging that may be required.

3. **Output** - displays output, once a SAS program is submitted, such as tables and graphs (if this is what has been specified in the program).

4. **Results** - this window helps to manage SAS output.

5. **Explorer** – this window is used to view and manage files/data sets (similar to the Microsoft Windows version of Explorer). SAS libraries, files, and shortcuts can be created within this window.

Another useful window is the **Keys** window, for creating shortcuts. This window is called up from the Options menu, and shows how commands have been assigned to various keys and combinations of keys. These can be changed and saved to suit your preference; some useful key assignments are:

- ***next*** - to move through all the SAS windows that are open.

- ***recall*** - to bring back the most recently submitted set of programming code back into the Program Editor window (this does not work in any other window).

- ***clear*** - to clear everything within the activated window (Program Editor, Log, or Output). This includes material not visible on the screen.

- ***undo*** - to undo the last keystroke (e.g., undo the clear command if a window was inadvertently cleared).

For additional information on how to navigate through the SAS windows environment, new Windows users are referred to the tutorial titled *"Getting Started with SAS Software"* under the SAS Help menu, which is provided with the software.

b) File Management

"Files" can refer to data sets (both SAS and non-SAS), as well as to other types such as the ASCII files generated when SAS programs are saved from the Program Editor window or when material is saved from the Log and Output windows.

Right-clicking on a file will yield the file properties while double-clicking with the left button will open the file. SAS data files generated in SAS version 9.1 will normally have ".sd7" as an extension and SAS catalog files will have a ".sc7" extension.

The following section covers saving, organizing, and documenting files.

SAVING FILES

When saving files within the Program Editor, Log, or Output windows, the *File/Save As* option will automatically create the following file extensions: ***".sas"*** for SAS programs created in the program editor window, ***".log"*** for output from the log window, and ***".lst"*** for material

generated in the output window. The Results Window can also be used to view, save, and manage individual results which appear in the Output Window.

Material entered within a window should be saved at periodic intervals (e.g., every 10 minutes). This can be set up automatically with the *autosave* option in the Tools menu (*Options/Preferences/Edit*). It is not necessary to save logs and outputs as long as the programs and data that produce them are saved (for important runs, however, it may be desirable to save such files, in addition to generating a paper copy).

ORGANIZING FILES

A ***"My SAS Files" directory*** is automatically created upon installation of SAS Version 9.1; the location may vary with the operating system. Windows Explorer, which can be invoked using the button at the top of the SAS Window display *(/Tools/Find),* can be used to find the exact path of this directory (in the *Advanced* menu, specify *Folders* rather than *Files and Folders*, and in the *Name and Location* menu enter *SAS* as the search string). The user could then set up, for example, one directory per project, with separate subdirectories under each project directory for SAS programs, logs, output, and data sets.

Libraries refer to the physical location where SAS files are stored. By default, several libraries are already defined by SAS:

1. **WORK -** used by SAS for storage of temporary files.
2. **MAPS** - contains SAS maps for most countries in the world. These maps are used with the SAS GMAP procedure.
3. **SASUSER** - automatically generated by SAS to save SAS default settings.
4. **SASHELP** - contains the SAS help catalogs; they are views (a type of data set) that describe every active library, data base, and catalog.

Data should not be stored in any of the default libraries; however, new libraries can be defined so that they, too, are automatically created each time SAS is started up (by specifying *enable at startup* when first created).

To assist in specifying to SAS where a particular file is located, a one-word reference can be assigned for the path of the file, particularly useful when the path involves a long list of sub-directories. The LIBNAME statement can be used, for example to assign a library called *mydir* to represent *c:\My Documents\My SAS Files\projects\ami*. To direct SAS to a particular data set (e.g., *amidata.sd7* in the *ami* directory), the user could simply specify *mydir.amidata* (the SAS data set extension does not need to be specified).

DOCUMENTING FILES

Documentation of both SAS programs and SAS output such as graphs and tables is essential for

effective management of files. For programs, comment lines are useful for basic file documentation and for explaining what certain sections/lines of SAS code are intended to accomplish. For output, titles and footnotes can provide useful information such as project title, type of analysis, type of data, and what exclusions may have been made, instead of having, for example, the default "The SAS System" at the top of each page of output.

THE SAS PROGRAM

a) Syntax

SAS programs are normally developed within the Program Editor window. Nothing entered in this window is processed, or read by SAS, until the program is submitted, or executed (unlike commands). A ***SAS session*** lasts until the user exits the SAS program.

Upon submission, the program from the Program Editor Window and any messages will appear in the Log window. After SAS has finished processing the program, the Output window, with the requested output, should be displayed on top of the other windows. If there were problems in the processing, the Log window might instead be displayed. In either case, though, the first thing that should be done is to check the SAS log for messages before reviewing the output. Note that the lines in the log output and the page numbers in the output generated in the Output window are numbered sequentially and cumulatively for the entire SAS session. They are reset with each new SAS session.

If there are problems, the submitted program can be brought back into the Program Editor window with a *Recall* command, changes can be made (saving the revised program), and the program can be re-submitted. The Output and Log windows should be cleared first (*Clear* command) so that new material is not mixed in with old material (all material generated in both Log and Output windows are kept cumulatively throughout the SAS session). If desired, the contents of any of the three windows can be saved using the *File/Save As* commands.

A SAS program consists of SAS ***statements*** which are constructed using SAS ***language***, several key characteristics of which are described below, followed by an example program.

- **Case.** Lower case is typically used in writing SAS programs although SAS currently recognizes upper case in a program (upper case was used in the example program below simply to denote SAS keywords). If any values in a data set are in upper case, however, references to such values within a SAS program must also be upper case. References to *values and variables* must also distinguish, where relevant, numbers from letters. For example, the number "0" must be distinguished from the letter "O" within a SAS program for it to be read accurately (the "0" in the variable "diag01", for example, is numeric).
- **Naming conventions.** Names of both data sets and variables currently can be up to a maximum of 32 characters long (starting with a letter or underscore) and can include numbers.
- **Keywords** - are used to specify the tasks to be carried out by SAS (e.g., SET, RUN). Keywords should not be used as variable or data set names.
- **Statements** - can be longer than one line and can begin anywhere on the line (alternatively, one line can have several SAS statements). ***ALL*** SAS statements must end with a semicolon (**";"**), while indentation of statements is optional.

- **Steps** - represent broad categories of tasks within which most programming is done in SAS. Typically used are:
  - DATA steps (e.g., for creating data sets, for creating new variables, etc.) and
  - PROC steps (e.g., for creating tables, graphs, formats, etc.).

  Each DATA and PROC step should end with a "RUN;" statement.

  - **Comment lines** - are useful for program documentation or for temporarily making SAS statements non-executable. They instruct SAS to ignore material contained within:

a) **/*** and ***/** This can generally be used almost anywhere (e.g., within a SAS statement).

/* This is an example. */

b) ***** and **;** This can NOT be used *within* a SAS statement.

To comment out existing SAS statements (which already end with a ";"), simply add ***** at the beginning of a statement.

*** This is an example. ;**

EXAMPLE OF SAS PROGRAM SYNTAX

PROC FORMAT; **VALUE 'M' = 'Male'** **'F' = 'Female';** **RUN;**	*This PROC step is an example of how to create labels for a gender variable consisting of M/F values. PROC, FORMAT, VALUE, and RUN are the SAS keywords. There are 3 statements in this 4-line program (the VALUE statement, ending with ";" is 2 lines long).*
/* create a SAS data set*/ **DATA new1;** /* read a SAS data set*/ **SET original;** * keep only women ; **IF gender='F';** **RUN;**	*This DATA step is an example of creating a temporary data set called "new1" from a temporary SAS data set created earlier in the SAS session called "original"; it instructs SAS to keep only females in the "new1" data set. The two types of comments are also illustrated.*
PROC FREQ DATA=new1; **TABLES gender;** **RUN;**	*This PROC step instructs SAS to read the "new1" data set, and to create a table showing the distribution of the gender variable.*

Program Development

A document prepared by the Manitoba Centre for Health Policy on program development provides suggestions on how to structure SAS code and what might be included in the program. This section builds on that document.

SAS code can be entered consecutively within the Program Editor window to create a large program, or the code might reside in other files that SAS can be instructed to find and process.

The above 3 components in the table, for example, could be in 3 different files on a floppy disk called s*tudy.fmt, prog.sas*, and a*nalyses.sas*. Two ways in which SAS could read and process the files are:

1. Open each file into the Program Editor window until all 3 are present in the window. All code from each file will be seen in the window. Note that SAS programs can be submitted in portions (each of the above 3 components could be submitted separately) or all at once, combining a number of DATA and PROC steps.

2. Within the Program Editor window, use ***%include*** to process each file, i.e., submit the following 3-line program:

```
%include 'a:\study.fmt';
%include 'a:\prog.sas';
%include 'a:\analyses.sas';
```

An important distinction between the two approaches is that the first approach allows the exact code which generated the results to be seen in the log. ***%include*** is used more typically when code is used repeatedly or when the user is familiar with the file(s) being included.

b) Log Messages: Debugging the SAS program

Problems in programming syntax can generally be identified from the SAS log and can be corrected by recalling the program (under the run menu) into the Program Editor window to make the necessary changes. Before submitting the corrected program, it should be saved, at the same time clearing both the log and output windows.

(This section focuses only on syntax errors; however, it is also possible for SAS to calculate a new variable using syntactically correct code that results in inaccurate calculations, or in results not reflecting what the user intends. For this reason, it is always wise to check values of a new variable against values of the original variable used in the calculation (as illustrated in the section on creating new variables.)

There are 3 main types of messages that SAS will generate in the log: 1) Notes, 2) Errors, and 3) Warnings. They are highlighted here with 4 examples:

1. "NOTE"

NOTEs are always generated in the log; they provide important information about the processing of the SAS program such as:

- number of observations and number of variables in a newly created data set.
- length of time taken by the processing (both real and cpu time).
- indications of certain types of programming errors.

5 ***NOTEs*** are illustrated in the above log excerpt:

- The first ***NOTE*** indicates that numeric values have been converted to character values for the variable *op01*, indicating that SAS sees this variable as numeric, not character. But this variable was read in as character, suggesting another kind of problem with the variable - see the 3rd note.

- The second ***NOTE*** indicates the converse of the first - that character values have been converted to numeric at two places in line 1210. This indicates that even though the

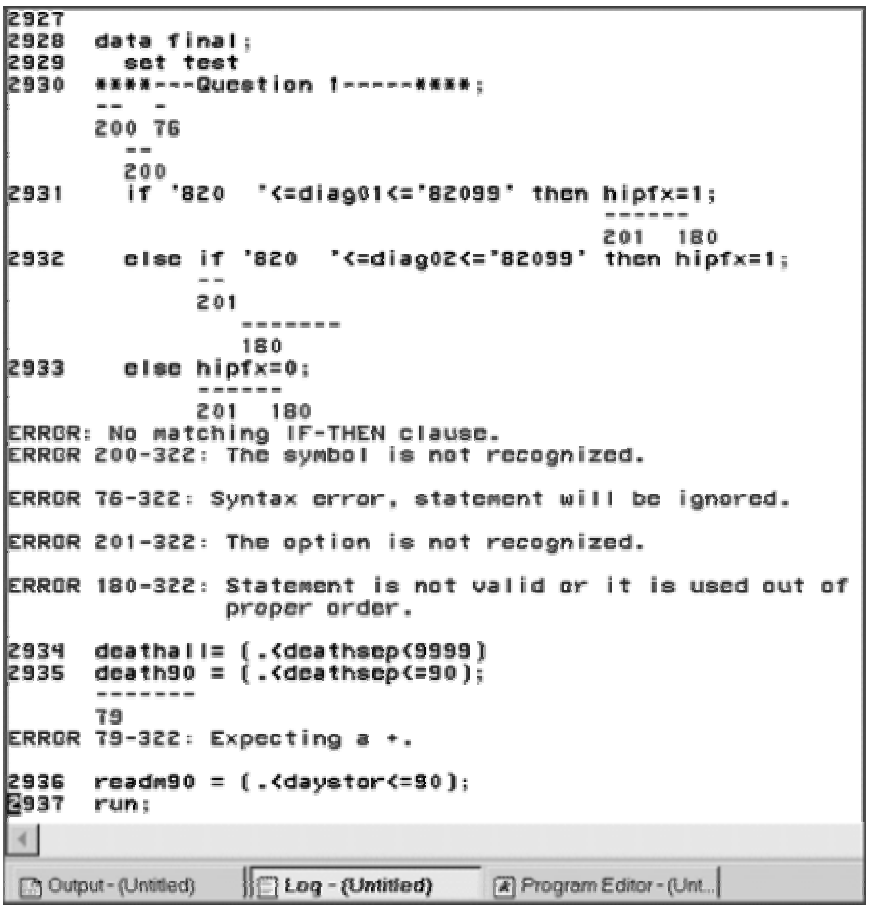
values are specified as character (with quotes), SAS will convert them to numeric, because the variable was originally read in as numeric. The conversion can affect accuracy of results, so it is advisable to make the necessary changes. This is easily fixed, simply by taking the quotes off the values so that the statement reads ***readm90 = (.<daystor<=90);***

- The third ***NOTE*** indicates that SAS does not recognize the *op01* variable and, in this case, precipitates the first note. The reason this variable is ***uninitialized*** is that it was spelled wrong, using *opo1* (letter "o") instead of *op01* (numeric "0").

- The fourth and fifth ***NOTE***s are always generated for DATA steps; they indicate, in this case, that the temporary SAS data set ***final*** has 5000 observations and 40 variables and that 55 seconds were needed (in real time) to process the DATA statement.

2. "ERROR"

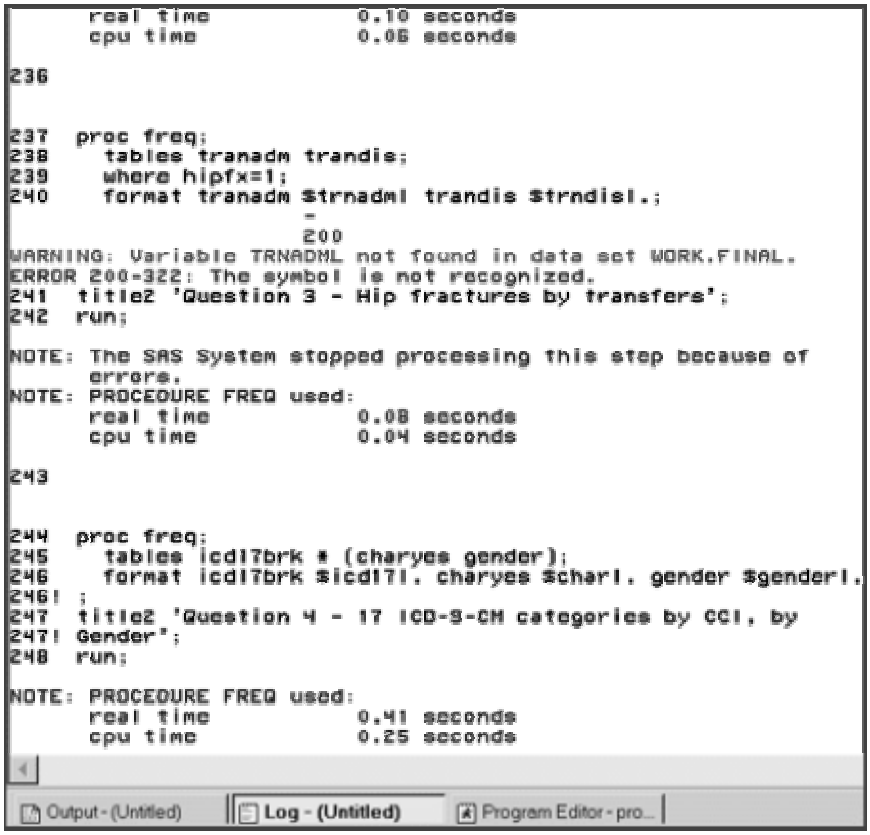
Error messages are the most obvious clue that something is wrong with the SAS program. Unlike Notes and Warnings, the program will not complete processing until the necessary changes have been made. Because one error can result in multiple error messages, fixing the first-occurring error will frequently clear up the remaining error messages.



The above log excerpt shows a number of Error messages, all resulting from only one error. The first indication of the problem (200) denotes that the underlined symbol is not recognized. Reviewing the SAS code in the line just before ******** shows that the statement ***set test*** has no semicolon. Adding ***;*** at the end of the statement will resolve ALL Error messages in this program.

3. "WARNING"

Warnings are frequently indications of problems in how SAS processed the program (although not always, and it should be noted that SAS may not always stop processing the program). Warnings should always be investigated.



In the above example, the ***WARNING*** indicates that SAS is expecting to find a variable called ***$trnadml***, and an ***ERROR*** message is generated indicating that the ***$*** symbol is not recognized. The problem, however, is that SAS does not recognize that ***$trnadml*** is a format which, because it is associated with the variable ***tranadm***, requires a period at the end (i.e., ***format tranadm $trnadml. trandis $trndisl.;*** will resolve the problem).

A very common ***WARNING*** is the one illustrated above, saying that a quoted string has become extremely long. Most frequently, the problem is a quote being inadvertently left out. In this case, adding the missing quote (i.e., ***'820 '<=diag02<='82099'***) will fix the problem and remove the Error messages showing up in the rest of the program.

This will not, however, fix an associated problem. A most important caveat when receiving this type of log message is to also check the message above the menu on the Log window. If it says, as in this example, ***"DATA STEP running"***, then steps must be taken to stop the program from running. Even though SAS will continue to process other programs, results of such programs may be inaccurate, without any indication of syntax problems showing up in the log. Several suggestions to stop the program are:

- Submit the following line: ***'; run;***
- Submit the following line: ****))%*'''))*/;***
- If all else fails, exit SAS entirely (making sure that the revised program has been saved) and restart SAS.

I. DATA PREPARATION

TYPES OF DATA SETS

This section provides both general guidelines and specific details on preparing data sets (both SAS and non-SAS). For the data sets referenced in this manual, detailed instructions are provided in other sections on how to prepare the *htwt* data set, the *clinical* data set, and the *simulated Manitoba Health* data set.

Data sets can be thought of as a table having columns and rows, and consisting of three main components:

1. Values	Numbers and/or letters of the alphabet comprising the information in each cell (column/row combination).
2. Variables	Names assigned to columns of information; currently, they can be up to 32 characters long (starting with a letter or underscore).
3. Observations (Records)	Usually one line, or row, of information per person or event; each observation (also known as a record) consists of a set of values.

Illustrated below, on the left, is the raw data set *htwt*. The viewtable on the right shows how the data become meaningful once the appropriate information on the data has been specified to SAS.

```
Aubrey      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
```

	name	sex	age	height	weight
1	Aubrey	M	41	74	170
2	Ron	M	42	68	166
3	Carl	M	32	70	155
4	Antonio	M	39	72	167
5	Deborah	F	30	66	124
6	Jacqueline	F	33	66	115
7	Helen	F	26	64	121
8	David	M	30	71	158
9	James	M	53	72	175
10	Michael	M	32	69	143
11	Ruth	F	47	69	139
12	Joel	M	34	72	163
13	Donna	F	23	62	98
14	Roger	M	36	75	160
15	Yao	M	.	70	145
16	Elizabeth	F	31	67	135
17	Tim	M	29	71	176
18	Susan	F	28	65	131

M and ***F***, for example, are the ***values*** for the ***variable*** SEX; the next column represents the values for the variable AGE. Each ***observation*** is now consecutively numbered, in this case,

from 1 to 18. The first observation thus has a value of Aubrey for the variable called NAME, a value of M for the variable SEX, a value of 41 for the variable AGE, a value of 74 for the variable HEIGHT, and a value of 170 for the variable WEIGHT.

In other words, the first observation is an individual named Aubrey who is a 41-year-old male who is 6'2" tall and weighs 170 pounds. (Normally a codebook will specify the units in which the values are being measured. In this case, height, for example, is measured in inches and weight is measured in pounds.)

The values for any given variable will have the following characteristics:

- **Length**. Numeric values are stored in SAS as floating-point, or real binary, numbers. According to the SAS Language Reference (1990:86), floating point representation is "a form of storing in scientific notation" ("in which values are represented as numbers between 0 and 1 times a power of 10") "except that on most operating systems the base is not 10, but is either 2 or 16".

  SAS assigns a default length of 8 bytes of space to numeric variables and, where space permits, this need not be reduced. Cody and Pass (1995:276) indicate that "this does not mean 8 significant figures; it means that 8 times 8, or 64 bits (8 bits per byte) are used to store the number". They add that 8 bytes "is equivalent to what used to be called 'double-precision' in other languages. This will vary not only by which computer language you are using, but on which computer and under what operating system you are running".

  4 bytes of space is generally sufficient for most numeric variables, but a SAS Institute manual should generally be consulted before changing the lengths of numeric variables because of the potential loss of precision.

- **Numeric vs. character values**. If calculations are not necessary (e.g., values of 1 and 2 for GENDER), it is recommended that such numeric values be assigned as character values (in the INPUT statement when reading a raw data set, or using a PUT statement within a DATA step when accessing a SAS data set). Conversion, where possible, to character decreases space requirements. Values for the variable GENDER, for example, if numeric, can take up to 8 bytes of space, but assigning it as a character value decreases its size to 1 byte. References to all character values in a program must be enclosed in single quotes (e.g., if gender='1';).

- **Missing values**. Numeric missing values are denoted with a period (.) while character missing values are denoted with a space in quotes (' ').

Analyses using SAS software require that the data be in the form of a SAS data set. If the data are in this form, no data preparation is needed; the data set can be easily viewed and explored using a SAS program to create either: a) a permanent or b) a temporary SAS data set. A ***permanent*** SAS data set is normally created if it is known which subsets are being used for analysis (e.g., if only a bypass procedure is of interest, only the hospital records that actually

contain this surgery would be needed). ***Temporary*** SAS data sets, on the other hand, last for the duration of the SAS session, and are useful when developing and debugging programs. **Non-SAS data sets** require additional preparation because they need to be converted to SAS data sets.

A. PREPARING SAS DATA SETS.

To create a ***temporary*** SAS data set from another temporary SAS data set requires only the DATA and SET statements and the one-word name of the data set. To read or to create a ***permanent*** SAS data set requires a third statement - LIBNAME - to tell SAS where to find the data set.

- **LIBNAME sasref 'c:\sasdir';** tells SAS that the user has chosen *sasref* as the name to represent a directory on c: drive called *sasdir*. That is, ***LIBNAME*** tells SAS where the permanent SAS data set is located (or to be located).

- **DATA two;** (to create a temporary SAS data set) **OR**
  DATA sasref.two; (to create a permanent SAS data set).

  The ***DATA*** statement is required to tell SAS to create another, new, SAS data set from the data set specified in the ***SET*** statement following.

- **SET one;** (to read/access a temporary SAS data set) **OR**
  SET sasref.one; (to read/access a permanent SAS data set)

  For permanent SAS data sets, the ***SET*** statement tells SAS: a) the name assigned to the directory, or location, of the data set (*sasref*) and b) the name of the permanent SAS data set (*one*). For temporary SAS data sets only the one-word file name need be specified. (SAS will automatically assign the temporary SAS data set to the WORK library for the duration of the SAS session, but the temporary data set can be referred to without specifying this library.) The ***SET*** statement thus tells SAS to read (or access) the data set, loading the information into memory so that the user can view or manipulate it. Any changes made to the data set specified in the ***SET*** statement will be reflected in the data set specified in the ***DATA*** statement, NOT in the original data (unless it is being saved with exactly the same name).

These keywords are illustrated in the accompanying SAS program examples.

It can be possible to create a large number of temporary SAS data sets in the course of a SAS session but generally it is desirable to conserve space. If the reason for creating the new data set is to create more variables, for example, the same data set name can be used (e.g., ***data one; set one;)***. This will simply overwrite the previous data set. Subsetting the data (e.g., keeping only age 65+) will also conserve space. Assigning a new name in this case (e.g., ***data age65p; set one;***) will permit the user to access either data set during the SAS session.

Note that two options can be useful when creating permanent SAS data sets; both are placed in

the DATA statement; for example: ***data sasref.new (compress=yes label='Simulated MB Health data');***

- ***compress*** - removes the extra space used by non-filled or partially filled variables. This option can reduce the size of a data set quite substantially, but note that in certain cases, it can actually increase its size. The log will provide this type of information so that any necessary adjustments can be made.

- ***label*** - permits adding a brief description of the data set; this information would then be seen in output generated by ***PROC CONTENTS***.

Additional information on reducing the space taken up by SAS data sets is available from the MCHP document titled "Saving Space in SAS".

B. PREPARING NON-SAS DATA SETS

The non-SAS data set, which can be converted to a temporary or permanent SAS data set, can take one of two forms: I) an ASCII file or II) a file generated by another software package.

I. A file of ASCII (raw) data

This may look something like the following:

```
12 38 8 011275
22 18 9 000088
31  0 4 100
```

These numbers cannot be meaningfully manipulated unless the user is given additional information to tell SAS the variable names, their locations, and whether they should be read in as numeric or character. The SAS program must use an INPUT statement to provide SAS with this information, as well as FILENAME and INFILE statements (the simulated Manitoba Health data set is an example of this type of data set):

- A **FILENAME** statement is necessary to specify which **file** contains the raw data. The statement **FILENAME rawref 'c:\sasref\rawdata';** provides both the location (*sasref* directory) and the file containing the raw data (*rawdata*).

- An **INFILE** statement is used to indicate that a raw data set should be **read in** by SAS, as specified in the FILENAME statement (INFILE is used to read raw data, while SET is used to read in SAS data). **INFILE rawref;** tells SAS to read the raw data set, name and location as described in the FILENAME statement.

- An **INPUT** statement is used to provide SAS with variable names, column numbers, and numeric/character information. For example, **INPUT age 14-15 gender $20;** indicates that age is two columns, or spaces, in width, starting at column 14 (this would not accommodate ages with 3 digits, i.e., ages>99), and that gender is 1 column wide,

located at column 20. The dollar sign denotes that gender is to be read in as a character variable while the absence of a dollar sign indicates that age is to be read in as a numeric variable.

Data values can also be located within a SAS program. In this case, a CARDS keyword is necessary to signal SAS that raw data values are to be read in within the program. The Height/Weight and clinical data sets illustrate this approach.

II A file generated by another software package

While ASCII files and input statements are the most commonly used method for importing data into SAS, many other programs store information into a format that SAS can read. These programs often keep information on variable type, length, and format. SAS can access information stored in formats saved by other programs in several ways.

1. Using delimiters in a 2-step process

a) Convert the external file to ASCII (text) format.

The external file from the native program is saved in ASCII format, with a special character or delimiter between each field (variable). Quote/comma and tab delimited are the most common types of delimiter. For example, delimited ASCII files can be saved from Excel 2002 by selecting 'Save as' and saving the file with a type of Text (Tab delimited)(*.txt).

b) Import the file into SAS.

- If using a SAS program, the delimiting character is defined in the INFILE statement, e.g.:

```
FILENAME RAWFILE 'C:\SASREF\RAWDATA' ;
 DATA NEW ;
     INFILE RAWFILE DELIMITER='09'x  ;
          ** '09'x stands for the tab
            character in hexidecimal format,
            the delimiter=dsd option can be used
            for quote/comma separated files;
     INPUT ONE TWO ;
```

- If using a graphical interface, select *Import Data* from the SAS *File* menu and select *User Defined file format*: EFI or select one of the pre-defined files under *Standard data source*, i.e., *Delimited*, *Comma Separated Values*, or *Tab Delimited File*.

2. Direct Access Using Engines

A format-specific engine on a LIBNAME statement will permit reading some data file formats directly. Generally files must be saved in a general or portable format prior to importing them into SAS. SPSS is used here as an example but other engines are available. Prior to importing an SPSS file it must first be saved as an SPSS Portable format (.por) file

from within SPSS (note that the SPSS engine does not work under SAS 6.12 and Solaris (UNIX)). A SAS program could then be used to read in the file, for example:

```
LIBNAME IMPORT SPSS 'c:\temp\spssfile.por' ;
** Note: file must be in SPSS portable format ;
DATA TEST ;
  SET IMPORT._FIRST_ ;
  ** _first_ is the first, and only, data set
     in the library, or directory.  ;
RUN ;
```

or, PROC CONVERT can be used with the following syntax:

```
FILENAME IMPTSP 'c:\temp\spssfile.por' ;
PROC CONVERT SPSS=IMPTSP DATA=TEST ;
RUN;
```

3. Importing data into SAS

PROC IMPORT may be used to import a data set, or a data table, from a variety of different sources. In some cases, the specific rows and columns will have to be defined; in other cases whole tables can be imported. PROC IMPORT can be used to import files created from such programs as MS Access (ACCESS), DBase (DBF), Lotus (WK1, WK3, WK4), and Excel (EXCEL, EXCEL4, EXCEL5, EXCEL97, EXCEL 2002). ASCII delimited files can also be imported as delimited files (DLM, CSV, TAB). The *Access* component of SAS software must be installed and available for each file type.

Data can be imported:

- from an MS Access database:

```
PROC IMPORT OUT= temp
DATATABLE= "Base"
DBMS=ACCESS  REPLACE;
** DBMS defines the database or file type;
DATABASE="C:\database.mdb";
RUN;
```

- from an Excel database:

```
PROC IMPORT OUT= htwt2
      DATAFILE= "x:\sasdir\htwt.xls"
      DBMS=EXCEL2002 REPLACE;
  GETNAMES=YES;
RUN;
```

- from the SAS menu, i.e., this procedure can be used interactively through the *File/Import/Standard Data Source* option.

PROGRAM EXAMPLES FOR DATA PREPARATION

Several examples are shown here of creating and reading/accessing data sets for: 1) Permanent SAS data sets, 2) Temporary SAS data sets, and 3) Raw data sets.

1) Permanent SAS Data Sets

```
*************************************************
*This program CREATES a permanent SAS data set  *
*it assumes that a temporary SAS data set has   *
*already been created during the SAS session.   *
*************************************************;
libname sasref 'c:\sasdir';
    /* where to store the new permanent SAS data set*/
data sasref.new;
       /* create a permanent SAS data set in the
          sasdir directory */
set htwt;
       /* access, or read, a temporary SAS data set*/
run;
*************************************************
*This program READS a permanent SAS data set,     *
*creating a temporary SAS data set.               *
*************************************************;
libname sasref 'c:\sasdir';
    /* location of permanent SAS data set on C: drive */
data one;
     /* create a temporary SAS data set */
set sasref.survey;
     /* read the permanent SAS data set called "survey" */
run;
*************************************************
*The above program also CREATES a temporary SAS  *
* data set from a permanent SAS data set.        *
*************************************************;
```

2) Temporary SAS data sets

```
*************************************************
*This program READS a temporary SAS data set,   *
*creating another temporary SAS data set.       *
*It assumes that "one" has already been created *
*during the SAS session.                        *
*************************************************;
data two;
     /* create a temporary SAS data set called "two" */
  set one;
     /* read the data set called "htwt" */
if gender='F';
     /* keep only females in the "two" data set */
        /* (this assumes the variable "gender" exists in "one")*/
run;
```

3) Raw data sets

```
*****************************************************
*This program CREATES a permanent SAS data set      *
*from a file containing raw data. (To create a      *
*temporary SAS data set, a libname is not needed,   *
*so the libname would come out and the data statement*
*would be data one instead of data sasref.one).     *
*****************************************************;
filename rawref 'c:\sasdir\rawdata';
  /* name and location of raw data file on C: drive */
libname sasref 'c:\sasdir';
/* location for new permanent SAS data set */
data sasref.one;
  /* create a permanent SAS data at the sasref location*/
  infile rawref;
  input name $1-10
        sex $12
        sales 20-25;
run;
```

II. VIEW THE DATA: SAS PROCEDURES

Four SAS procedures are described here. Two SAS procedures - CONTENTS and PRINT - are frequently used to take a first look at the data. Two other procedures - PROC FORMAT and PROC SORT - can be used with them to enhance the output, the former for labelling or grouping data values, and the latter to change the order in which the records are sorted. Except for PROC CONTENTS, all examples assume that a temporary SAS data set has been created from the height/weight data.

1. PROC CONTENTS

PROC CONTENTS can be used to obtain general information about a SAS data set, including an alphabetic list of variables and their attributes (e.g. type, length). Details are also provided regarding the data set itself, such as number of observations and number of variables, and whether the data set was sorted by any variable(s) or compressed.

```
*****************************************************
*This program was used on the simulated Manitoba    *
*Health data, both for Version 1 and Version 2      *
*(the latter showing the output with labels added to*
*both variables and values)                         *
*****************************************************;


proc contents data=test;
run;
```

2. PROC PRINT

PROC PRINT can be used to display the values for any of the variables and for any number of observations in the SAS data set. Five examples of PROC PRINT, using the height/weight data set, are shown here, the latter three being illustrated with the use of PROC SORT.

Example 1: PROC PRINT

```
      ****************************************************
      *This program creates a listing                    *
      *of all the values and all the variables.          *
      ****************************************************;

      proc print data=htwt;      /* Begin the PROC step */
                              /* Add 2 titles */
        title1 'PROC PRINT: Example 1';
        title2 'No keywords specified except for TITLE';
      run;             /* End the PROC step */
```

Example 2: PROC PRINT

```
      ****************************************************
      *This program produces output that illustrates     *
      *the use of a number of optional keywords and      *
      *statements that can be used with PROC PRINT.      *
      ****************************************************;
      /* Display the first 10 records (this requires the data=
         option).  The LABEL keyword is necessary for the LABEL
         statement below */

      proc print data=htwt (obs=10) label;
      /* Instead of numbering the records sequentially,
         identify them by the values of the name variable */
        id name;
      /* Only print the data values for two
         variables (age and sex) */
        var sex age;
      /* Add up the values for the weight variable */
        sum weight;
      /* Add labels for 4 variables */
        label name   = 'Name of student'
              weight = 'Weight in pounds'
              sex    = 'Gender of student'
              age    = 'Age of student';
       /* Instead of displaying sex with values of M and F
          use the format $sexl (previously created) and the
          format statement to label them as Male and Female */
        format sex $sexl.;
           /* Add 2 titles */
        title1 'PROC PRINT:  Example 2';
        title2 'Use of OBS=, LABEL, ID, VAR,
                SUM, and FORMAT keywords';
      run;
```

3. PROC SORT

PROC SORT is used to sort a data set on specified variables. PROC PRINT is used here to illustrate the results of different ways of using PROC SORT (PROC SORT by itself does not produce any output in the Output window). It is important to note that sort order sequence (i.e., whether numbers or alphabetic characters are sorted first) and how missing values are dealt with can vary with the operating system. In PC SAS, numeric values are ordered before alphabetic values.

Example 3: PROC PRINT AND PROC SORT

```
*******************************************************
*This program sorts the data by name and creates a    *
*listing of the values of 3 variables (name being     *
*placed in the first column)for the first 10 records.*
*The resulting output is displayed                    *
*in alphabetical order of name.                       *
*******************************************************;

proc sort data=htwt;
  by name;
run;

proc print data=htwt (obs=10);
  id name;
  var sex age;
  title1 'PROC PRINT: Example 3';
  title2 'Where the data set is sorted by name';
run;
```

Example 4: PROC PRINT AND PROC SORT

```
*******************************************************
*This program sorts the data in reverse order of name*
*and creates a listing of the values of 3 variables  *
*(name being placed in the first column) for the      *
*first 10 records.  This output is displayed          *
*in reverse alphabetical order of name.               *
*******************************************************;

proc sort data=htwt;
  by descending name;
run;

proc print data=htwt (obs=10);
  id name;
  var sex age;
  title1 'PROC PRINT: Example 3';
  title2 'Where the data set is sorted by DESCENDING name';
run;
```

Example 5: PROC PRINT AND PROC SORT

```
*****************************************************
*This program creates another data set called "other"*
*which is sorted by sex and, for each value of sex, *
*is sorted by age.  The PROC PRINT step is identical*
*to Example 4 except the newly created data set is  *
*specified to produce output instead of
*the "htwt" data set.                               *
*****************************************************;

proc sort data=htwt out=other;
  by sex age;
run;

proc print data=other (obs=10);
  id name;
  var sex age;
  title1 'PROC PRINT: Example 5';
  title2 'Where the data set is sorted by sex and age';
run;
```

4. PROC FORMAT

PROC FORMAT is an extremely useful SAS procedure for creating formats that can be used to label data values or to group them. The PROC FORMAT statement is usually placed prior to a DATA step (although it can be run separately, creating formats that can be used at any time during the SAS session). Separate VALUE statements are required for each format; multiple VALUE statements can be specified under one PROC FORMAT statement. A data set is not specified when using a PROC FORMAT statement. PROC FORMAT does not change, manipulate or do any calculations on the data. It simply creates formats that the user can use in PROC or DATA steps after PROC FORMAT has ran.

Format names are assigned by the user; they must be no longer than 32 characters and cannot end in a number (In older versions of SAS, format names can only be 8 characters long). Formats that will be used with **character** variables MUST start with "$" (the “$” counts as one of the 32 characters allowed). The format name can also be used to distinguish grouping formats (e.g., ending in "F" or "G") from labeling formats (e.g., ending in "L"). Another useful convention is to repeat the original value in the new label being created (e.g. *'A' = 'A.Winnipeg'* instead of *'A'='Winnipeg')*. The output could then display not only the label for the value, but the original value as well.

Once PROC FORMAT is submitted, only the log indicates that the program has executed; it should show the names of the formats that have been created. The log will add an additional note indicating that the format "is already on the library" if the format already exists (e.g., was previously submitted), and indicating that the previously existing format has been overwritten. This is not a problem unless the user wishes to keep the pre-existing format as well - in that case,

the new format should be given a new name before submission (and before SAS overwrites the pre-existing format).

No output is produced in the Output window when submitting PROC FORMAT. The formats, however, are now available for use at anytime during the current SAS session, and can be used for labeling values (using the FORMAT statement) or for creating new variables by grouping values using the e.g., PUT statement.

```
*************************************************
*This program creates several formats.          *
*All values on the left side of "=" refer to values *
*that must already exist in the data set.  All  *
*values on the right side are created by the user.  *
*The keywords LOW, HIGH, and OTHER are illustrated. *
*************************************************;

proc format;
/*1.Create format to be used to label CHARACTER values*/
 /* Create $SEXL format (need $ and quotes)*/
  value $sexl
         'M' = 'M.Male'
         'F' = 'F.Female';
/*2.Create format to be used to label NUMERIC values */
  value sexl
           1 = '1.Male'
           2 = '2.Female';
/*3.Create format to be used to group CHARACTER values */
              /* Group values of A and B into value 1*/
  value $regionf 'A','B' = '1'
              /* Group values C to E into value 2 */
                'C'-'E' = '2'
              /* Group all other values into value 3 */
                Other   = '3';
/*4.Create format to be used to group NUMERIC values */
  value agef
  /*Note that missing values would be included in
       the <30 category. 0-29 could be specified instead
       of low-29 to exclude the missing values from
       the grouping. */
            low-29 = '1'
             30-39  = '2'
             40-49  = '3'
           50-high  = '4';
run;
```

VIEW THE DATA: PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data (a link to this data is found on the website under sample data sets). Examples are given for how program, log, and output might look.

1. Generate a list of variables and their attributes.

2. Generate the following listings of variable values:

   - All variables for all observations in the data, displaying their original values.

   - The first 5 observations, printing values for the following 3 variables: gender, diastolic blood pressure, and systolic blood pressure. Display labels for the variable names in the output, and add value labels for the gender variable.

   - Re-run the same program on all observations, except this time display the data for the 3 variables sorted by gender. (2 procedures required.) If the original sort order is desired to be kept in the clinical data, the user has the option of creating an output data set, sorted by gender, with a different name.

   - Re-run the same program, except this time sort the data by both gender and systolic blood pressure, and display gender in the first column (rather than having the observation number showing). (2 procedures required.)

3. Change how output is displayed for the gender variable and display a listing for only this variable. Instead of displaying *Male* and *Female,* have the values read *Male adult* and *Female adult*. (2 procedures required.)

III. EXPLORE THE DATA

This section describes how to generate some statistics for numeric data and how to use tables to display either numeric or categorical data.

STATISTICS FOR NUMERIC DATA

Certain SAS procedures can only be performed on numeric data. Two such procedures - PROC MEANS and PROC UNIVARIATE - are illustrated here using the height/weight SAS data set. (Note that PROC SUMMARY generates output similar to PROC MEANS.)

1. PROC MEANS

PROC MEANS: Example 1

```
      ***************************************************
      *This program creates output (Example 1)          *
      *using the default setting of PROC MEANS.         *
      ***************************************************;
       proc means data=htwt;/* Begin the PROC step */
                            /* Add 2 titles */
        title1 'PROC MEANS:  Example 1';
        title2 'No keywords specified';
      run;             /* End the PROC step */
```

PROC MEANS: Example 2

```
      ***************************************************
      *This program specifies a series of keywords and  *
      *optional statements to create output (Example 2) *
      *using PROC MEANS. The CLASS statement avoids having*
      *to sort the data first, but the CLASS statement is *
      *more suited to smaller data sets or when just a few*
      *CLASS variables are to be used.                  *
      ***************************************************;
        /*Some of the keywords available with PROC MEANS:
                    N - number of observations
                    MEAN - mean value
                    MIN - minimum value
                    MAX - maximum value
                    SUM - total of values
                    NMISS - number of missing values
                    MAXDEC=n - set maximum number of
                     decimal places */
      proc means data=htwt n
         mean min max sum nmiss maxdec=1;
         /*Apply analysis only to "age" variable*/
        var age;
         /*Separate the analysis by values of sex*/
        class sex;
         /* Add 3 titles */
        title1 'PROC MEANS:  Example 2';
        title2 'Use of VAR, CLASS, and TITLE statements';
        title3 'CLASSED by gender';
      run;
```

PROC MEANS: Example 3

```
      ***************************************************
      *This program generates output (Example 3)        *
      *similar to Example 2 but displays the output     *
      *slightly differently and also creates another SAS  *
      *data set. Additional resources are used because    *
      *the data must be sorted first.                   *
      ***************************************************;
```

```
 /* Sort the data first because a BY statement
    is being used in the next PROC step */
       /*Sort by sex */
proc sort data=htwt;
  by sex;
run;

proc means data=htwt n
    mean min max sum nmiss maxdec=1;
     /*Separate the output by sex*/
  var age;
  by sex;
 /*Create a temporary SAS data set containing
      the information generated by PROC MEANS */
  output out=agedata;
  /* Add 3 titles */
  title1 'PROC MEANS:  Example 3';
  title2 'Use of VAR, BY and OUTPUT statements';
  title3 'SORTED by gender';
run;

   /*Display values of the new data set*/
proc print data=agedata;
   /* Add a 4th title*/
  title4 'A print of the OUTPUT data set';
run;
 /*Remove Titles 2-4 from the next set of output*/
title2;
title3;
title4;
```

2. PROC UNIVARIATE

PROC UNIVARIATE provides additional statistics, some of which are not available from PROC MEANS (e.g. mode).

```
PROC UNIVARIATE: Example
      ****************************************************
      *This program uses PROC UNIVARIATE to create         *
      *detailed output of numeric statistics               *
      *(Univariate example) on the "age" variable.         *
      ****************************************************;

      proc univariate data=htwt;
      var age;   /*Apply analysis only to "age" variable*/
        title1 'PROC UNIVARIATE example';
      run;
```

EXPLORE NUMERIC DATA: PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data. Continue to use the same program as used for the other practice exercises. The format file does not need to be included for this section. Examples are given for how program, log, and output might look.

1. Generate numeric statistics using the default setting for PROC MEANS.

2. Obtain the mean values for heart rate and systolic and diastolic blood pressure, limiting the decimal places to 2, and indicating how many missing values there may be.

3. Re-submit the question, this time obtaining the mean values for the 3 variables for each gender and for whether or not the patient is pregnant. Save these values to a separate data set, and display a listing of these values. (3 procedures)

4. Obtain mean, median, and mode values for systolic and diastolic blood pressure.

CREATING TABLES FOR NUMERIC OR CATEGORICAL DATA

The SAS procedure **PROC FREQ** is commonly used to produce summary data in tabular form. Five examples are shown here using this procedure on the height/weight data set. It can be used on either character or numeric data, although a procedure specifically for numeric data (like PROC MEANS or PROC UNIVARIATE) may be more appropriate for numeric variables having many different values.

The following is a summary of options and optional statements that can be used with PROC FREQ. ***Optional statements*** can be in any order, while ***options*** are entered at the end of the TABLES statement, following "/" and before ";" Note that this list represents only a portion of all available to the user from SAS:

- **TABLES** - optional statement for specifying the variables to be included in the analysis.

- **WEIGHT** - optional statement for specifying the variables to be summed for each value of the variables specified in the TABLES statement.

- **CHISQ** - option to obtain chi-square statistic to test for significant differences.

- **ALL** - option to obtain all statistics available with PROC FREQ.

- **MISSING** - option to include missing values in the calculations within the table.

- **MISSPRINT** - option to display the missing values in the tables without including them in the calculations.

- **LIST** - option to list values of variables side by side rather than in tabular form.
- **OUT=** - option to create a data set containing the output generated by the TABLES statement.

PROC FREQ: Example 1

```
************************************************
*This program creates output (Example 1)       *
*using the default setting of PROC FREQ, which  *
*produces 1-way tables of ALL the variables in  *
*the data.                                      *
************************************************;

          /* Begin the PROC step */
proc freq data=htwt;
          /* Add 2 titles */
  title1 'PROC FREQ:  Example 1';
  title2 'No keywords specified';
          /* End the PROC step */
run;
```

PROC FREQ: Example 2

```
*****************************************************
*This program creates 1-way tables for two variables*
*(Example 2).                                       *
*****************************************************;

proc freq data=htwt;
     /* Produce tables for 2 variables */
  tables sex age;
  title1 'PROC FREQ:  Example 2';
  title2 '1-way tables for variables specified
 by TABLES keyword';
run;
```

PROC FREQ: Example 3

```
*****************************************************
*This program creates a 2-way table (a "cross-tab"),*
*from a subset of the data (Example 3) by adding an *
*asterisk between the two variables. The            *
*values for the first variable specified appear on  *
*the left side of the table while the values for the*
*second variable appear across the top of the table.*
*A statistic is requested and a new data set is also*
*created.                                          *
*****************************************************;

proc freq data=htwt;
            /* Produce cross-tab with chi-square
```

```
                statistic and create a new data set
                containing the output generated by
                the TABLES statement*/
  tables sex * age /chisq out=freqtbl;
             /*Keep only ages 0 to 29 */
  where 0<=age<=29;
  title1 'PROC FREQ:  Example 3';
  title2 '2-way table using the CHISQ,
            WHERE, and OUT= keywords';
  title3 'Subsetting ages 0 to 29';
run;

      /* Produce a listing of the new data set*/
proc print data=freqtbl;
  title4 'A PRINT of the OUTPUT data set';
run;
```

PROC FREQ: Example 4

```
*******************************************************
*This program creates a 2-way table listing the      *
*values of the variables side by side (Example 4).   *
*This is a useful way of checking the values         *
*of existing variables against those of new          *
*variables to ensure they have been accurately       *
*created.                                            *
*******************************************************;
proc freq data=htwt;
        /* Use the LIST keyword to list the values
             side by side, and the MISSING keyword to
           indicate which variable(s) may have missing
             values*/
  tables sex * age /list missing;
  title1 'PROC FREQ:  Example 4';
  title2 '2-way table using LIST and MISSING options';
       /*Remove previous TITLE3 and TITLE4 */
  title3;
  title4;
run;
```

PROC FREQ: Example 5

```
*******************************************************
*This program creates a 3-way table using three      *
*variables on a subset of the data (Example 5).      *
*The first variable represents the control variable, *
*for which separate output (cross-tabs of the other  *
*two variables)is created for each of its values.    *
*******************************************************;

proc freq data=htwt;
    /* Controlling for "name", produce cross-tabs
          of "height" by "weight"*/
```

```
  tables name * height * weight;
        /*Keep only ages 0 to 27 */
  where 0<=age<28;
  title1 'PROC FREQ:  Example 5';
  title2 '3-way table: height by weight,
     controlling for name';
run;
```

Note: SAS can create tables that cross any amount of variables (i.e., 'n'-way table), but interpretations can get complicated with too many variables.

EXPLORE DATA: PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data and that the format file has been included (the data and format file can be found on the website under sample data sets). The default setting for PROC FREQ is would generate a lengthy list of all numeric and character variables; instead the variables for analysis should always be specified using a TABLES statement (similar to the VAR statement used in the numeric procedures MEANS and UNIVARIATE). Examples are given for how program, log, and output might look.

1. Create one-way tables for each of the following variables: gender, pregnant, primary DX and secondary DX. Add value labels for each of them; the format names are found in the format file for the clinical data set. These one-way tables display the distribution of values for each of the specified variables.

2. Create separate two-way tables (or cross-tabs), i.e., one variable against the other, for each of the following questions; label the values of each variable using the available formats:

   - What proportion of pregnant women were taking vitamins, compared with non-pregnant women? In this case, only women should be kept for analysis.
   - How does primary diagnosis differ by gender? (Suggestion: put gender as the last variable in the TABLES statement because it has only 2 values. Recall that values for the last variable are displayed across the width of the table.)
   - Create a side-by-side listing to check the values of gender against the values of pregnant.

3. Controlling for gender, how does the distribution of primary diagnosis differ for those taking vitamins versus those not taking vitamins? This can be answered using a 3-way table.

IV. DATA MANIPULATION

BASIC TECHNIQUES

SAS provides for many optional statements and keywords that can be used in SAS programs to

facilitate manipulation and display of the data. Statements can often (but not always) be entered in any order within DATA and PROC steps, while options must usually be placed in a specific position within a SAS statement. The **data=** option, for example, can be added to most procedures to specify the data set on which it should be run, e.g., ***proc freq data=test;***. Unless the data set is specified (in this case, "test"), SAS will automatically go to the most recently created data set.

Two broad categories of statements/keywords are described here - those that can be used to: a) create subgroups of data, and b) customize display of output. A SAS program incorporating the use of these statements/keywords follows.

A. CREATE SUBGROUPS OF DATA

Analysis and space requirements will often dictate whether to create separate SAS data sets (temporary or permanent) for analysis or to simply split the output by the desired values of a variable. If all analysis is to be conducted on individuals age 65+, for example, it might be desirable to create a separate, permanent SAS data set, removing all records having an age of less than 65. Not only is the data set being tailored to meet analysis needs, program efficiency is also enhanced by reducing the amount of time and space being used to carry out SAS runs - extremely important if computer resources are limited in terms of both physical and memory space.

Three approaches to creating subgroups of data are described here, specifying 1) data values, 2) variable names, or 3) number of observations to reduce the data set.

1. **Specify data values** with the **WHERE** or **IF** statements to keep a subset of **records**, or observations. Although only the observations containing the specified value(s) will be kept, all other variables associated with these observations will also be kept.

   where age>=65; This can be used within a PROC or DATA step to keep only those records having an age of 65 or older within a DATA step.

   if age>=65; This can only be used in a DATA step; however, the advantage of using ***IF*** is that a number of conditions can be specified:

   - ***if age>=65 and gender='F';*** This statement tells SAS to keep all females who are 65 years of age and older.
   - ***if age>=65 or gender='F';*** This statement tells SAS to keep all females and all people (both male and female) who are 65 years of age and older.

   IF statements can be used on variables created in the current data step. WHERE statements can only be applied to variables that pre-exist in the data.

2. **Specify variable names** using the **KEEP** keyword/statement (to keep selected variables) or the **DROP** keyword/statement (to drop selected variables) to create a subset of **variables**.

data new;
set test (keep=regionre los);

run;

The above is an example of using KEEP as a keyword; it keeps 2 variables from *test* in the "new" data set as the data set is being read in (the other variables, however, are still accessible for other data steps). (Alternatively, the KEEP statement can be placed at the end of the DATA statement, but this is less efficient because all variables in the *test* data set will be processed.)

data test2;
set test;
drop drg drgrgn drgw;
run;

The KEEP and DROP keywords can also be used as a statement. In this case only the DROP keyword is being used and 3 variables are dropped from the new *test2* data set. This statement is often used when new variables have been created from existing variables, and the existing variables are no longer necessary for analysis.

input age 7-9 regionre 51 deathsep 55-58;

When reading in raw data (e.g., the simulated Manitoba Health data), only the variables necessary to the analysis need to be read in.

Note that the DROP and KEEP keywords only affect variables; they do not affect the number of observations.

2. **Specify number of observations** with the **(OBS=)** option to keep a subset of records.

data test;
set test (obs=10);
run;

This option is very useful when testing/debugging SAS programs or portions of code. It is easily removed, and the program can be re-submitted to obtain output for all the observations. The following illustrates how the option can also be used when reading in ***raw*** data.

data test;
infile rawfile (obs=10);

B. CUSTOMIZE DISPLAY OF OUTPUT

Output can be enhanced in a number of ways, only a few of which are presented here:

1) LABEL, 2) FORMAT, 3) TITLE, and 4) FOOTNOTE statements.

1. Change how ***variable*** information is displayed by using a **LABEL** statement.

   e.g., ***label height = 'Height in inches'***
   weight = 'Weight in pounds';

   This code will attach labels to *height* and *weight* so that the labels rather than just the variable names will be displayed in any output. Labels currently can be up to 256 characters long, and must be enclosed in single quotes (double quotes if there is an apostrophe in the label). The LABEL statement usually goes in the DATA step, near the end, and is very helpful for explaining what the variables represent, particularly if other users will be accessing the data.

2. Change how data ***values*** are displayed by using the **FORMAT** statement.

   e.g., ***format gender $genderl. regionre regionh $regionl.;***

   This FORMAT statement assumes that formats called *$gender* and *$regionl* have been created (using PROC FORMAT). The variable(s) is specified first, and then the format, which always has a period at the end of it (at least one space between each). The $ denotes a character format; it can only be used with character variables. The FORMAT statement will result, for example, in the values of the variables *regionre* and *regionh* being displayed as full region names rather than '1' through '8' although certain SAS procedures may truncate the formatted values from the maximum allowable 32 characters long to 8 or 16 characters in length).

   The FORMAT statement can be used within a PROC or DATA step. If used within a DATA step, the format is applied in all procedures referencing the data set. If used within a PROC step (generally preferred), the format is only applied for that specific procedure. The original, underlying values are NOT permanently changed; only how they appear in output is changed.

   Note that SAS also has its own library of formats that are available throughout any SAS session.

3. Enhance ***output*** by adding a title(s) using the **TITLE** statement.

   e.g., ***title1 'An example of a title';*** can be used within a PROC or DATA step, or by itself, to place a title on each page of subsequent output. Note that:

   - Titles are enclosed in single quotes; anything within the quotes is not processed by SAS. Double quotes are used if the title contains any apostrophes.

   - Up to 10 titles can be added, using up to 10 TITLE statements. Each additional title is numbered, e.g., ***title2 'An example of a 2nd title';***

   - TITLE statements are global statements which mean that they are displayed on every subsequent page of output in a SAS session. They can be either overwritten (by specifying new TITLE statements) or cleared (by specifying in the program e.g., ***title1;*** to clear a

TITLE1 statement, ***title2;*** to clear a TITLE2 statement, and so on).

4. Enhance ***output*** by adding a footnote(s***)*** using the **FOOTNOTE** statement.

   e.g., ***footnote1 'An example of a footnote';*** can be used within a PROC or DATA step, or by itself, to place a footnote on each page of subsequent output. The above notes for the TITLE statement also apply to the FOOTNOTE statement.

```
*******************************************************
*This program creates two data sets "men" and "women" *
*and generates 2 versions of tables showing the       *
*distribution of name separately for males and for    *
*females over age 40. It assumes that the data set    *
*"htwt" has been previously created. It also assumes  *
*that labels have not yet been created for the        *
*variables.                                           *
*******************************************************;

proc format;          /*Create label format for sex*/
  value $sexl 'M'='Male'
              'F'='Female';
run;

/*Create 2 new temporary SAS data sets*/
data men women;

        /*Read in the "htwt" data set, keeping only
           3 of the variables and the first 10 records*/
  set htwt (keep=sex name age obs=10);

      /*For the "men" data set keep only the records
         that have a value of "M" for sex */
  if sex='M' then output men;

     /*For the "women" data set keep only the records
        that have a value of "F" for sex */
      /*(Note that records missing values for sex
           would not go into either data set) */
  else if sex='F' then output women;

  /*Create labels for the variables being kept*/
  label name = 'Name of individual'
        age    = 'Age at admission'
        sex = 'Gender of patient';
run;

proc freq data=men;
   tables name;
   where age=40;
   format sex $sexl.
  title1 'Example re use of basic techniques';
  title2 'The Student Project';
```

```
  footnote1 'Limiting age to 40+';
run;
proc freq data=women;
   tables name;
   where age=40;
   format sex $sexl.;
run;

*********************************************;
*Instead of the 2 PROC steps used above, the  *
*same analysis could be done as follows using *
*PROC SORT and BY variable processing.        *
*********************************************;

/*Sort the data by sex prior to creating
tables that are split by sex */


proc sort data=htwt;
  by sex;
run;

proc freq data=htwt;
  /* Create a table of distribution of name */
  tables name;
  /* Do this separately for each value of sex*/
  by sex;
   /* Do this only for age 40+*/
 where age=40;
    /* Display formatted values*/
  format sex $sexl. ;
  title1 'Example re use of basic techniques';
  title2 'The Student Project';
  footnote1 'Limiting age to 40+';
run;
```

BASIC DATA MANIPULATION: PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data. Use the same program as used in the above practice exercises. The format file does not need to be included. Examples are given for how program, log, and output might look.

1. Create 3 separate temporary SAS data sets for each of the following and carry out any SAS procedures that will indicate only the specified data was kept.
    - only pregnant females.
    - only data on blood pressure and heart rate data, along with the id number (for the whole sample).
    - only the first 15 observations.
2. Display the distribution of pregnant by vitamins. Re-submit this table after adding the

following:

- A new label for the vitamins variable: *Patient on vitamin therapy*.
- New labels for the values of pregnant: label "1" as *3+ mos.pregnant* and "0" as *LT 3 mos. pregnant*
- A title showing the source of data and a footnote reflecting the type of exercise.

CREATE NEW VARIABLES

New variables can only be created within the context of a DATA step; they will be included in the new data set specified in the DATA statement. In the following example, the temporary SAS data set *addvar* will contain 2 new variables: *newvar* and *newvar2*:

```
data addvar;
   set test;
   newvar=(regionre='A');
   newvar2=(put(age,agefmt.));
 run;
```

When creating new variables, several guidelines are important:

- **Numeric vs. character**. It should always be determined whether the existing variables are character or numeric as this will affect how the values will be referenced.

- **Naming variables**. A recommended practice is to always give new variables new names. This enables others who may be using the data to feel confident that the original names represent the original variables. This convention also provides a way of checking the original variable against newly created ones to ensure their accuracy.

- **Repetitive tasks**. This is not currently a factor in the simulated Manitoba Health data, but it should be pointed out that alternate approaches are available for accomplishing repetitive tasks in SAS programming. If the same processing, for example, has to be done on the same kind of variable (e.g., diagnosis) and there are 16 fields, or variables, for this information (e.g., DIAG01 to DIAG16), a DO loop, combined with an ARRAY statement, is most useful.

Two broad categories of statements for creating new variables are illustrated here: 1) IF/THEN statements, and 2) assignment statements. One of the differences between these two categories is where the new variable name is placed. In IF/THEN statements, the new variable is referred to at the ***end*** of the statements that refer to the existing variable. The new variable name is followed by the equal ("=") sign and the value(s) to be assigned for the new variable. In assignment statements, the new variable is referenced at the ***beginning*** of the SAS statement, followed by the "=" sign, and then the existing variable(s).

Descriptions and programs are provided for each of the two categories, illustrating their use on

the height/weight data set. Program 1 compares and contrasts the use of IF/THEN statements with an assignment statement that uses the PUT function. It also illustrates the use of an assignment statement to create a dichotomous variable. Program 2 illustrates the use of two other types of assignment statements, one using arithmetic operators and another using the SAS function, SUBSTRING.

DATA MANIPULATION - CREATING NEW VARIABLES: PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data, including the format file. Use the same program as used in the above practice examples. Examples are given for how program, log, and output might look.

1. Calculate a new variable (*bpratio*) that represents a ratio of systolic to dystolic blood pressure. Round it to the nearest single decimal place. Do a frequency distribution of the new variable.

2. Assuming that the 2-digit diagnosis for the variable *prim_dx* can be meaningfully collapsed to 1-digit diagnosis, create a new variable (prim_sub) that will only contain the 2nd digit. Check the new variable against the values of the original variable (using PROC FREQ with a LIST MISSING option).

3. Create a new blood pressure variable (*bpnorm*) that simply denotes normal/not normal using a dichotomous assignment statement based on both readings of blood pressure. Consider the norm for diastolic to be 60 to 90 and for systolic to be 100 to 140; the norm must be present for both variables. Check the new variable (which will have 1/0 values) against the values of the two original variables.

4. Create two new heart rate variables (*rateif* and *rateput*, each of which groups the same values of heart rate into 3 categories: low (less than 70), moderate (70-85), and high (86 and over). Use IF/THEN statements to create one variable, and the PUT function to create the other. In addition to creating the grouping format required for the latter, create a labeling format for the 3 different groups. Do frequency distributions (labeling the new values) for the 2 variables - they should be identical; however, the differing distributions illustrate the importance of identifying missing values prior to creating new variables and determining how to deal with them.

V. ADDING VARIABLES AND OBSERVATIONS TO DATA SETS

I. ADDING VARIABLES USING THE SET STATEMENT

a) Concatenating Data Sets

The SET statement when used with one data set can allow you to read or modify the data. If the SET statement is used with two or more data sets it can not only allow you to read and modify the data but also it can concatenate or stack the data sets on top of each other. The SAS system will read all observations from the first data set then the second and so on until all observations are read. This process is useful when you want to combine data sets that have most or all of the same variables with different observations.

The number of observations in the new data set will be the sum of all the observations from the original data sets. The order of the observations is based on the order of the list of the original data sets. If any of the data sets has a variable that is not contained within another data set, the observations from that data set will have missing values for that particular variable.

```
*This program assumes that the data set htwt has already been created*

      /*Create temporary data sets*/
data male_htwt;
set course.male_htwt;
run;

data female_htwt;
set course.female_htwt;
run;

      /*Add observations by creating a new data set*/
data concat;
      /*concatenate the data sets using a SET statement*/
      /*create variables that indicate whether the data set set contributed
data to the current observation, using in=*/
set male_htwt (in=m1)
    female_htwt (in=m2);
      /*Make the indicators permanent variables*/
inmale=m1;
infemale=m2;
run;

PROC PRINT data=concat;
title 'Data=Male and Data=Female Concatenated';
run;
```

b) Interleaving Data Sets

If you have data sets that are sorted by some variable, simply concatenating the data sets as shown previously, may unsort the data sets. If you want to concatenate observations from two or more data sets in a particular order, it is more efficient to use a BY statement with the SET statement outlined above. This process is called interleaving data sets.

Before you can interleave the data sets you must sort the data sets by the interleaving variable using PROC SORT. Like concatenated data sets, the number of observations in the new data set is equal to the sum of observations from the original data sets. If a data set does not have a variable contained within the other data sets, the observations will be set to missing.

```
*This program assumes that the data sets htwt, male_htwt, and female_htwt
have already been created*

      /*Sort the male and female data sets BY age*/
PROC SORT data=male_htwt;
by age;
run;

PROC SORT data=female_htwt;
by age;
run;

      /*Create a new data set interleaving the male and female data
      sets by age*/
data interleave;
      set male_htwt
          female_htwt;
      by age;
run;

PROC PRINT data=interleave;
title 'Interleaving Male and Female Data Sets by Age';
run;
```

II. ADDING VARIABLES USING THE MERGE STATEMENT

To match observations from one data set to another, you can use the MERGE statement in the DATA step. If you know that the two or more data sets are in exactly the same order then you do not need a common variable between the data sets (mismatched merge). However, data sets are usually merged together using a merge key (match merge). In this tutorial we will only look at match merges. A merge key is a variable that is common to both data sets (i.e. a variable that has the same name and length in both data sets). Before you can merge the data sets, they both must be sorted by the merge key using PROC SORT.

There are two types of merges: a) One-to-one merges and; b) One-to-many merges.

a) One-to-One Merge: Combines observations from two or more data sets into a single observation in a new SAS data set.

If you merge two or more data sets and they both have variables with the same names other than the merge key, the variables in the second data set will overwrite the variables with the same name in the first data set.

```
*This example assumes that the data set htwt has already been created*

      /*Create a temporary SAS data set*/
data htwt_reg;
set course.htwt;
run;

      /*Sort the data sets by the merge key*/
PROC SORT data=htwt;
by name;
run;

PROC SORT data=htwt_reg;
by firstname;
run;

data mer;
      merge htwt (in=m1)
          /*Merge keys from both data sets must have the same name, rename
            the merge key in the htwt_reg data set*/
            htwt_reg (in=m2 rename=(firstname=name));
      by name;
            /*Create variables that indicate which data set contributed the
            observations*/
      inone=m1;
      intwo=m2;
run;

PROC PRINT data=mer;
title 'Merged Data Set';
run;
```

b) One-to-Many Merge: Refers to the case where one data set has one observation for each value of the merge key and the other data set has more than one observation for each value of the merge key.

```
*This example assumes the data set htwt has already been created*

      /*Create a data set with one observation per value of sex*/
PROC MEANS data=course.htwt;
class sex;
var age;
      /*Create a temporary data set called mage*/
output out=mage mean=mean_age;
run;

PROC SORT data=course.htwt out=htwt;
by sex;
run;

PROC SORT data=mage;
by sex;
run;

data mer;
      merge htwt (in=m1)
            /*Create variables that indicate what data set contributed their
            observations.  In the data set mage keep only the variables
            sex and mean_age*/
            mage (in=m2 keep=sex mean_age);
      by sex;
run;
```

Caution: It is important to remember to use a BY statement in the merge. By default, SAS will not report an error and you may end up with a mismatched merge instead of the matched merge that was intended.

ADDING VARIABLES AND OBSERVATIONS TO DATA SETS – PRACTICE EXERCISES

These questions assume that a permanent SAS data set has been created from the sample clinical data, including the format file. Use the same program as used in the above practice examples. Examples are given for how program, log, and output might look.

1. Create two data sets containing:

   - Only Males
   - Females that are not pregnant

Concatenate the data sets and interleave the data sets by date of birth.

2. Create two new data sets keeping only the following variables:

   - id, gender, date of birth, primary DX, secondary DX
   - gender and heart rate

Merge the data sets together to add the 'heart rate' variable. Limit the data set to those who have a heart rate over 70.

3. Create a data set with one observation per value of gender using PROC MEANS. Use the variable 'heart rate'. Find the proportion of observations with heart rate greater than the mean heart rate.

VI. DATA PROCESSING

I. ARRAY STATEMENT

Purpose

Arrays are often used in conjunction with DO loops when performing actions for a series of variables. The following example illustrates the same action being performed on two separate diagnostic field variables. The study diagnosis of 820.0 can occur in either of these fields, and the statements are identical except for the name of the diagnostic field. The intent of the following statements is to flag all occurrences of the study diagnosis by creating a new variable - "HIPFRAC" - where '1' indicates the presence of the desired diagnosis.

```
If '82000'<=DX01<='82009' then HIPFRAC='1';
If '82000'<=DX02<='82009' then HIPFRAC='1';
```

Sixteen diagnostic fields (DX01-DX16), however, would require 16 lines of code.

Array processing can make the program more efficient by streamlining the code required to accomplish the task (depending on the situation, if-then/else statements can be faster; however, they are also more error-prone). A specified series of variables is associated with a collective name of your choice; for example, the diagnostic fields DX01 through DX16 could be associated with the name "DIAG", which will then operate similarly to variables in data step manipulations.

Syntax

Arrays are set up using an ARRAY statement. It can appear anywhere in the DATA step as long as it occurs prior to any reference to it. The variables that make up the array are called elements. Individual elements are identified by subscripts (numbers that identifies an element's position in the array).

ARRAY array-name {number of variables} variable-1, variable-2,...variable-n;

Array-name is a name you choose to represent the group of variables (must be 32 characters or fewer beginning with a letter or underscore).

Number-of-variables tells SAS how many variables are being grouped; it is represented by subscripts that are enclosed in brackets.

Variable-1, variable-2,...variable-n lists the names of the variables (the variable list does not have to begin at 1 - e.g., DX5-DX16)

Example

ARRAY diag{16} $ dx01-dx16;

This statement tells SAS to:

a) create a group or array name DIAG for the duration of the DATA step.

b) have DIAG represent 16 variables: diagnostic fields DX01 through DX16.

Note that DX01-DX16 are character variables and thus must be preceded by a "$".

You can refer to the entire array or just one of its elements when performing logical comparisons or arithmetic calculations. All variables listed in the ARRAY statement are assigned extra names with the form array-name{position}, where position is the position of the variable in the list (1,2,3,...,16 in the example). The additional name is called an array reference and the position is often called the subscript.

In the above ARRAY statement, DX01 is assigned the array reference DIAG{1}; DX02 the array reference DIAG{2}; etc. 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 DX01 and DIAG{1} are equivalent.

Caution

An array is simply a convenient way of temporarily identifying a group of variables; it exists only for the duration of the DATA step. **Arrays are not variables**.

II. DO LOOP

Purpose

The iterative DO statement is used to repeatedly execute a set of statements occurring between the DO statement and the END statement (the DO loop). It is often used in conjunction with ARRAY statements so that the repeated actions occur within the loop for each of a specified series of variables.

Syntax

A DO loop begins with an iterative DO statement, followed by other SAS statement(s), and completed with an END statement. This loop iterates (is processed repeatedly) according to the directions in the DO statement. Its basic form is:

```
DO <index-variable=1> TO <upper bound of array>;
      [or DO <index-variable=1> TO <upper bound of array> UNTIL
      <specified condition>]
      [or DO <index-variable=1> TO <upper bound of array> WHILE
      <specified condition>]
   <SAS statements>
END;
```

Index-variable is a name you choose (e.g., "I"). Its value changes with each iteration of the loop, or each time the loop is processed. By default, the value of the index variable (I) is increased by 1 before each new iteration of the loop, consecutively representing the values 1 to n (number of variables in array). DO loops can iterate by 2 or by 'n' with a BY statement (i.e., do i=1 to 10 by 2;).

Number-of-variables-in-array: If used in conjunction with an array, the loop will execute as many times as there are variables in the array. If there are 16 variables,

the loop will execute the statements on each of the 16 variables (i.e., DO I=1 to 16). The processing stops when the value of the index variable becomes greater than number-of-variables-in-array.

Example

The SAS statements in an iterative DO loop often contain references to an array. In the following example, the array name is "diag" and the number of variables represented in its subscript are 16. With each iteration of the loop, the value of the subscript is replaced with the current value of the index variable ("i"). Successive iterations of the loop process the statements on consecutive variables in the array. The intent of this example is to search all 16 diagnostic fields to flag any occurrence of a hip diagnosis of ICD-9-CM 820.

```
ARRAY diag{16} $ dx01 - dx16;                          (1)
      hipdiag=0;                                       (2)
      DO i=1 to 16;                                    (3)
      IF '820 '<=diag{i}<='82099' THEN hipdiag='1';    (4)
      END;                                             (5)
RUN;
```

(1) Create an array called "diag" to represent the group of diagnostic fields dx01-dx16 for the duration of the data step.

(2) Create a new variable named "hipdiag" and set it equal to zero. It will remain at 0 if none of the 16 diagnostic fields for that record have a hip diagnosis present.

3) Perform the actions in the loop sixteen times for each record in the data set. When the value of **"i"** is 1, SAS reads the array reference as DIAG{1} and processes the statements on DIAG{1}, that is, DX01. In each iteration of the loop, the subscript associated with DIAG is replaced with the index variable's (i) current value.

(4) When a diagnosis within the specified range is encountered, the variable "hipdiag" will be assigned a value of '1'.

(5) All iterative DO loops must end with an END statement.

Example: Output

```
OBS DX01 DX02 DX03 DX04 DX05 DX06 DX07 DX08...DX16 HIPDIAG
1    650 V270                                          0
2  71783                                               0
```

```
3   4549                                                    0
4   V664 82021 E8809 4538 3569 36250 7213 2859              1
5   V301 7746                                               0
6   8208 E888 4019 3310                                     1
7   4111 4140 4280 4011 42731 586 5990 V668                 0
8    486                                                    0
9    V72                                                    0
10   650 V270                                               0
```

Observation 4 has hipdiag set to ’1’ because DX02 has a value of ’82021’, thus falling within the ICD-9-CM range 820-820.99. Observation 6 similarly has hipdiag set to ’1’ because DX01 has a value of ’8208 ’.

III. BY-GROUP PROCESSING (FIRST. /LAST.)

Purpose

By-group processing refers to the use of a BY statement in a DATA step, which permits identification of the first- and last-occurring record for each of the specified BY variables. Two dichotomous (1/0) variables are automatically created for each variable specified in the BY statement when using SET, MERGE, or UPDATE: FIRST.varname and LAST.varname, where varname is the name of the BY variable(s). By creating these variables, a number of various calculations are possible, such as obtaining a count of records for each unique identifier.

Syntax

BY varname1 varname2...;

For the first record in a BY group, the value of the FIRST.varname1 is set to 1, with all other records in the BY group set to 0. For the last record in a BY group, the value of the LAST.varname1 is set to 1, with all other records set to 0. If the data are sorted by more than one BY variable, the FIRST.varname for each variable is set to 1 at the first occurrence of a new value for the variable. FIRST. and LAST. variables are temporary variables that are only available for the current data step. You can create permanent variables equal to the temporary FIRST. and LAST. variables (i.e., firstvar=FIRST.var;). These permanent variables will be available in subsequent PROC and DATA steps.

Example: Single BY Variable

```
PROC SORT DATA=hosp;
BY phin;
```

```
RUN;

DATA dup;
   SET hosp;
   BY phin;                          (1)
   firstfl=FIRST.phin;               (2)
   lastfl=LAST.phin;                 (3)
RUN;
```

(1) Set the data by PHIN (already previously sorted by this variable) in order to create FIRST.PHIN and LAST.PHIN.

(2) Create a new variable called FIRSTFL and assign it a value of 1 for every FIRST.PHIN=1 encountered.

(3) Create a new variable called LASTFL and assign it a value of 1 for every LAST.PHIN=1 encountered.

Example: Output

OBS	PHIN	FIRSTFL	LASTFL
1	562737	1	1
2	563850	1	1
3	563961	1	1
4	565858	1	1
5	566739	1	1
6	**568729**	**1**	**0**
7	**568729**	**0**	**0**
8	**568729**	**0**	**1**
9	569961	1	1
10	660861	1	1

In the above example, the person with PHIN 568729 has 3 records (observations 6-8). For the first record (#6), FIRSTFL is set to 1, indicating that it is the first record for that person and all other records for that PHIN show FIRSTFL values set to 0. For the third and last record, LASTFL is set to 1, indicating that it is the last record for that person and all other records show LASTFL values set to 0.

Caution

When conducting BY-group processing, DO NOT do any data exclusions; data manipulation is ok. Data exclusions can be done in a subsequent data step. Data exclusions conducted during a data step with FIRST. and LAST. processing can cause unexpected results by

eliminating the FIRST. and LAST. records for each BY-group. The only time data exclusions can be done with BY-group processing is with a subsetting WHERE statement, which is applied to the data set coming in, before any BY-group processing is carried out.

IV. RETAIN STATEMENT

Purpose

We can often do data calculations/manipulations within observations, but sometimes it is necessary to do calculations across observations. The RETAIN statement is used to keep a specified value (assigned by an INPUT or assignment statement) from the current iteration of the DATA step to the next. Otherwise, SAS automatically sets such values to missing before each iteration of the DATA step. The RETAIN statement allows values to be kept across observations; for example, computing a running total of values, counting the number of occurrences of a variable's value, setting indicators within a BY-group, and so on. RETAIN statements are often used with FIRST. and LAST. processing.

Syntax

The RETAIN statement can be used to specify initial values for variable(s) or elements of an array. All elements or variables will be initialized to the specified value.

RETAIN <varlist> [initial-value(s)];

Varlist: specifies the names of the variables, lists or arrays whose values you wish to retain.

Initial-value(s): the initial value(s) can be numeric or character (e.g.,'y') and is assigned to all listed variables. If the initial value is not specified, it is set to missing.

The following shows four variables specified in each retain statement.

RETAIN var1-var4 1; sets initial values of var1, var2, var3, var4 to 1.
RETAIN var1-var4 (1); only var1 is set to 1; var2-4 are set to missing.
RETAIN var1-var4 (1 2 3 4); OR
RETAIN var1-var4 (1,2,3,4); var1 is set to 1, var2 to 2, var3 to 3, var4 to 4.

For example, the statement ***RETAIN pop 1;*** within a DATA step will assign a value of 1 to each observation for the variable POP.

```
/*Use the retain statement to count the number of observations
```

```
      in each BY group.  An index weight is identified and each
      subsequent weight is compared to the index*/

PROC SORT data=htwt_long;
by name age;
run;

data w_compare (keep=name age index_weight weight over count last_name);
      set htwt_long;
      by name;
      retain count index_weight;
      if first.name then do;
            count=0;
      /*Set and retain the first weight*/
      index_weight=weight;
      end;
      /*Counter for number of records for each*/
      count=count+1;
      /*Indicator variable for increased weight*/
      over = (index_weight < weight);
      last_name=last.name;
run;

PROC PRINT data=w_compare;
title 'Retain Statement';
run;
```

DATA PROCESSING – PRACTICE EXERCISES

The following two questions assume that a permanent SAS data set has been created from the sample clinical data, including the format file. Use the same program as used in the above practice examples. Examples are given for how program, log, and output might look.

1. Create a BY-group by primary DX.

2. Count the number of observations in each BY-group using the RETAIN statement.

The following questions assume that a permanent SAS data set has been created from the simulated Manitoba health data available at MCHP. Examples are given for how program, log and output might look.

1. Use the ARRAY statement to find all records with a diabetes diagnosis (code 250). Hint: the SUBSTR function is useful here.