1

Introduction to SAS®

1.1 SAS Program and Language

A SAS program can consist of one or more DATA steps and procedures (PROC steps), which can be in any sequence depending upon the programming purposes. The building blocks of the DATA and PROC steps are *statements* and are case insensitive. A statement is made up from one or a series of elements, such as SAS names, DATA step functions, operators, operands, special characters, and/or SAS keywords. Each statement must end with a semicolon. As a matter of fact, a missing semicolon in a statement is one of the most common programming errors for novice programmers. Some statements can only be used in either DATA or PROC steps, while some other statements, such as BY, WHERE, LABEL, and FORMAT statements, can be used in both steps. Furthermore, some statements can also be placed outside the DATA or PROC steps; these statements are referred to as being global in scope.

SAS statements that exist in the DATA step mainly serve as an instruction to read external data, create a new SAS data set, or perform data manipulation. Statements that are in the PROC step are mainly used to analyze data or create reports. Global statements are often used to provide information to the SAS system, such as modifying the appearance of the SAS log, adding titles or footnotes to the output, or controlling the methods as to how SAS processes your program.

In addition to statements, there are other types of SAS language elements, such as data set options, expressions, formats, and informats. These language elements are mostly used within a statement and will be introduced when they are used within a certain SAS statement throughout this book.

In order to be consistent in your SAS documentation, the syntax convention in this book follows the same convention provided in the *SAS® 9.2 Language Reference: Dictionary* (2009). The description of the syntax convention is described in the "Syntax Conventions for the SAS Language" article in SAS documentation.

1.2 Reading Data into SAS

The starting point for most projects that a programmer becomes involved with is reading data into the SAS system. The most frequently used input file formats a programmer will encounter are SAS data sets that already exist, raw text files, and Microsoft® EXCEL spreadsheets.

A SAS data set is often created by reading, extracting, or combining data from text files, EXCEL files, or some other SAS data sets. Once a SAS data set is created, programmers tend to save it as a permanent file. Not only is re-reading a SAS data set simple, but the SAS data set is also easier to manage than other types of file formats. External text files or EXCEL spreadsheets, for example, can be confusing if not enough attention is paid to how the data are arranged on a page. This section covers reading a SAS data set and reading a raw data file with fixed fields. Reading raw data with other types of formats will be covered in Chapter 8.

1.2.1 The SAS Data Set and SAS Library

A SAS data set is a SAS file that is created by SAS software. A SAS data set can be either a SAS data file or a SAS view. A SAS data view, which is not covered in this book, is a virtual data set pointing to the data from other sources. A SAS data file contains data and the descriptor information of the data. In this book, the terminologies for *data set* and *data file* are used interchangeably.

The values stored in the SAS data set are arranged in a table of rows and columns. Rows are referred to as observations or records, and columns are referred to as variables. In SAS, there are two types of variables: character and numeric. Character variables can contain alphanumeric values or special characters. The missing values for character values are presented as blanks. Numeric variables can contain floating-point numbers. The missing values for numeric values are represented as periods (.). In SAS, variables that contain date and time values are also considered numeric variables.

The descriptor information of a data set includes information about the creation date of the data set, the number of observations, and the attributes of each variable, such as a variable's name, length, type, label, format, informat, etc.

All files, including the SAS data sets on your computer, are stored and organized in different directories and subdirectories. In SAS, the directory that stores SAS data sets is referred to as a *library*. To access or create a SAS file in a library, you need to begin with the LIBNAME statement, which has the following form:

```
LIBNAME libref 'SAS-library';
```

The LIBNAME statement starts with the keyword LIBNAME and is followed by *libref* and '*SAS-library*'. *Libref* is the logical library name, while '*SAS-library*' is the physical location of the file folder. The basic idea of using the LIBNAME statement is associating a logical SAS library name, *libref*, with the directory path in which permanent SAS data sets are stored. Thus, in your SAS program, instead of referring to the file's directory with the complete path name, you will use *libref.* For example, in the following SAS statement, the *libref* is SASLIB, which is associated with the physical location 'C:\SAS Book\dat':

```
libname saslib 'C:\SAS Book\dat';
```

LIBNAME is a global statement and is used outside the DATA and PROC steps. Being *global* in this situation means that the name of the library, *libref*, is only in effect until you change it, cancel it, or terminate your current SAS session. The contents of the library always exist unless you delete them.

To access the data set in a library, you need to use a two-level name, which has the following form:

```
libref.filename
```

The first component of the two-level name is the library name, and the second component is the name of the SAS data set; the two names are separated by a period (.). For example, SASLIB.MYDAT refers to the SAS data set MYDAT that is stored in the SASLIB library.

The rules for naming *libref* and *filename* are similar. The starting position of both names must begin with either an underscore or a letter. The nonstarting position of the names can contain an underscore, letters, and numbers. The length of the *libref* cannot be more than eight characters, and the length of the *filename* can be up to 32 characters.

You can also use a single-level name by providing *filename* only and omitting *libref.* In this situation, you are accessing or creating a SAS data set stored in the WORK library. The WORK library is automatically created when you start your SAS session. The WORK library is a temporary library that is used to store temporary SAS files. What this means is that all files stored in the WORK library are deleted when the current session terminates. You can refer to the data set that is stored in the WORK library by using either a single-level name (*filename*) or a two-level name (WORK.*filename*). However, to refer to a data set that is stored in the permanent library, you must always use a two-level name (*libref.filename*).

1.2.2 Reading a SAS Data Set

Regardless of your programming purpose, writing a SAS program utilizing a DATA step will create one or more SAS data sets. For example, when you are reading a SAS data set into SAS, you are creating a SAS data set that is based on the SAS data set that you are reading.

A DATA step always starts with the DATA statement and usually ends with a RUN statement. To read a SAS data set into SAS, you need to use at least the following three statements:

```
DATA output-data-set-name;
   SET input-data-set-name;
RUN;
```

In the DATA statement, *output-data-set-name* is the name of the data set that you are creating, and the *input-data-set-name* in the SET statement is the name of the data set that you are reading. Both *output-data-set-name* and *input-data-set-name* can be either a one-level or two-level name. In this book, the data set being created after the keyword DATA is called the *output data set*, and the data set you are reading after the keyword SET is called the *input data set*. The RUN statement is used to execute the previously entered statements.

Notice that the SET statement is indented. SAS is an indentation-insensitive language. Indenting only serves to add visual clarity to your code. Other than the first and last statements of a DATA or a PROC step, programmers often indent the remainder of the statement so that each DATA or a PROC step can be distinguished individually.

To further clarify your program, you can provide documentation to your program by using the COMMENT statement. The COMMENT statement takes one of the following forms:

```
*message;
/*message*/
```

The *message* component in both formats is used to specify the text that explains the statement or program. SAS ignores text in COMMENT statements during processing. The *message* in the first format is enclosed between an asterisk (*) and a semicolon (;), and the *message* in the second format is enclosed between a slash star (/*) and star slash (*/). The main difference between these two formats is that the first one, starting with an asterisk and ending with a semicolon, cannot contain internal semicolons or unmatched quotations, while the second format can. Furthermore, you should avoid placing /* comment symbols in columns 1 and 2 of your code, because in some operating environments, SAS might interpret a /* in columns 1 and 2 as a request to terminate your SAS program.

Program 1.1 creates two SAS data sets by reading NOISE.SAS7BDAT from the SASLIB library. All the SAS data sets end with ".SAS7BDAT". For purposes of simplicity, the SAS file extension will be omitted for the remainder of this book. In Program 1.1, the first DATA step creates a temporary data set (NOISE1) that is stored in the library WORK. The second DATA step creates a permanent data set (NOISE1) that is stored in the DESKTOP library. Notice that both data sets being created have the same data set names but are stored in different libraries. This program also utilizes two types of COMMENT statements that provide further explanation of the program.

Program 1.1:

```
libname saslib 'W:\SAS Book\dat';
libname desktop 'C:\Documents and Settings\Desktop';
data noise1; *creates NOISE1.SAS7BDAT in the library WORK;
    set saslib.noise;
run;

data desktop.noise1;/*becomes NOISE1.SAS7BDAT in w:\SAS Book\
dat */
    set saslib.noise;
run;
```

The purpose of Program 1.1 is to provide more illustrations, because you would never actually write such a program since you are simply making a copy of the data set. If you would like to test this program, you need to download the data set NOISE on your computer and specify the correct directory path in which NOISE is located on your computer in the LIBNAME statement. The information regarding downloading all the testing data sets and the program is described in the Preface.

Once a program is submitted, the first thing that you should check is the SAS log that is generated from the submitted SAS code. The SAS log often contains information about submitted programs, including warning or error messages. For example, the SAS Log from Program 1.1 contains the message about the number of observations being read from the NOISE data set, the number of observations and variables in the output data sets, and DATA step computing times.

Log from Program 1.1:

```
13 libname saslib 'W:\SAS Book\dat';
NOTE: Libref SASLIB was successfully assigned as follows:
      Engine:      V9
      Physical Name: W:\SAS Book\dat
14 libname desktop 'C:\Documents and Settings\Desktop';
NOTE: Libref DESKTOP was successfully assigned as follows:
      Engine:      V9
      Physical Name: C:\Documents and Settings\Desktop
15 data noise1; *creates NOISE1.SAS7BDAT in the library WORK;
16     set saslib.noise;
17 run;

NOTE: There were 32 observations read from the data set
SASLIB.NOISE.
NOTE: The data set WORK.NOISE1 has 32 observations and 4
variables.
```

```
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds

18 data desktop.noise1;/*becomes NOISE1.SAS7BDAT in W:\SAS
Book\dat
19 ! */
20    set saslib.noise;
21 run;

NOTE: There were 32 observations read from the data set
SASLIB.NOISE.
NOTE: The data set DESKTOP.NOISE1 has 32 observations and 4
variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

1.2.3 Reading a Raw Data File with Fixed Fields

In this book, a raw data file is also referred to as a text file. A raw data file with fixed fields means that the values of the variables occupy the same location for all the observations. For example, HEARING.TXT contains variables in the fixed fields (only the first five observations are shown). The first row contains the column numbers; they are not part of the data file. The description of each variable, along with their column information, is shown in Table 1.1.

HEARING.TXT:

```
12345678901234567890123456789 0
629F H past    26 0       35000
656F W never   26 1 no    48000
711F W never   32 1 no    30000
733F W current 17 0       59000
135F B current 29 1 no   120000
```

TABLE 1.1

Variable Information for HEARING.TXT

Variable Name	Description	Locations	Variable Type
ID	Subject's four-digit ID	Columns 1–4	Character
RACE	Ethnicity	Column 6	Character
SMOKE	Smoking status	Columns 8–14	Character
AGE	Age	Columns 16–17	Numeric
PREG	Pregnancy	Column 19	Numeric
HEARING	Hearing loss	Columns 21–23	Character
INCOME	Annual income	Columns 25–30	Numeric

Reading an external text file often starts with the INFILE statement, which is used to identify the location of the text file. The INFILE statement has the following form:

```
INFILE file-specification <OBS=record-number>;
```

File-specification is used to specify the location of the input data; the common form is the physical file name along with its full path. The language elements that are enclosed between a less than sign (<) and a greater than sign (>) are *options* for the SAS statement. In this example, the optional OBS option is used to specify the first number of records to be read. This option is especially useful for reading a data set with a large number of observations. For example, you can read the first few observations to verify if the data is being read correctly before continuing to read the entire data set.

Once the file location is identified from the INFILE statement, you need to use the INPUT statement to read the external text file. In SAS, there are four types of input methods: *column*, *formatted*, *list*, and *named*. In this section, only the column input method is presented. The expansion of this topic, along with other input methods, will be covered in Chapter 8.

You can use the column input method to read raw data that contains variables in a fixed field with character or standard numeric values. The standard numeric data values include numbers, decimal points, numbers in scientific notations (e.g., 1.2E4), and plus or minus signs. Nonstandard numeric data can include date and time values, fractions, integers, real binary numbers in hexadecimal forms, and values that contain special characters, such as %, $, and comma (,). Here is the syntax for the column input method:

```
INPUT variable <$> start-column <- end-column>
```

In the INPUT statement, *variable* is the name of the variable that you are creating by associating it with the input values from specified columns. The naming convention for the *variable* is the same as the rules for naming a SAS data set. The optional dollar sign ($) is used when creating a character variable from character values. *Start-column* and *end-column* are the starting and ending positions of the input values. *End-column* is optional if the variable value occupies only one field. Program 1.2 reads the HEARING.TXT file by using the column input method.

Program 1.2:

```
data hearing;
    infile "W:\SAS Book\dat\hearing.txt";
    input id $ 1 - 4
          race $ 6
          smoke $ 8 - 14
          Age 16 - 17
```

```
          Preg 19
          Hearing $ 21 - 23
          Income 25 - 30;
run;
```

1.2.4 Reading Data Entered Directly into the Program

For a data set with a small number of observations and variables, you can enter the data directly into the DATA step by using the DATALINES statement, which has the following form:

```
DATALINES;
```

When reading data directly from the DATA step, you need to place the DATALINES statement in the last statement of the DATA step and enter your data immediately after the DATALINES statement. In the end, you need to write a single semicolon (a NULL statement) to indicate the end of the input data. If the data that you entered contains semicolons, you must use the DATALINES4 statement and use four consecutive semicolons (;;;;) instead of one at the end of the input data. Program 1.3 illustrates the use of the DATALINES statement by reading only the first 16 observations from the HEARING data set.

Program 1.3:

```
data hearing_small;
    input id $ 1 - 4
          race $ 6
          smoke $ 8 - 14
          Age 16 - 17
          Preg 19
          Hearing $ 21 - 23
          Income 25 - 30;
datalines;
629F H past    26 0      35000
656F W never   26 1 no   48000
711F W never   32 1 no   30000
733F W current 17 0      59000
135F B current 29 1 no  120000
982F W past    26 1 yes 113000
798F W never   19 0      28900
494F W never   36 0      65000
748F W never   34 1 no   39000
904F W never   25 0      76200
244F W never   28 1 yes  58000
747F A current 18 0      39000
796F A past    35 0     134000
713F H never   26 1 no   29000
```

```
745F A never   36 1 no   76000
184M W past    19 .      13900
;
```

1.3 Creating and Modifying Variables

After data is read into the SAS system, an immediate common task is to create or modify variables because the existing variables from the input data do not always contain the information that you need. This section covers creating variables by using the *assignment statement* and introduces how to create variables conditionally. Conditionally creating variables will be further discussed in Chapter 2.

1.3.1 The Assignment Statement and SAS Expression

The *assignment* statement is the most common method used for creating a variable. It has the following form:

```
variable=expression;
```

In the assignment statement, *variable* is either a new or existing variable, and *expression* is any valid SAS expression.

The purpose of using an expression in a SAS statement is to create variables, assign values, perform calculations, transform variables, and perform conditional processing. All expressions will return a result with a character, numeric, or Boolean value. An expression is formed by a sequence of *operands* and *operators*. Operands are either constants or variables. Operators include symbols for arithmetic calculations, comparisons, logical operations, SAS functions, or grouping parentheses.

Examples of operators, along with their evaluation orders, are illustrated in Table 1.2. Operations at priority 1 level are executed before priority 2 level, and priority 2 level operations are executed before priority 3 level, and so on. The order of operations can also be controlled by parentheses. You should always use parentheses when you are not sure about the operation order. Some operators have mnemonic-equivalent forms that provide alternative forms that you can use instead of the corresponding operators.

Comparison operators are often used with IF-THEN/ELSE statements. The use of comparison operators is for comparison or calculation purposes between two variables, constants, or expressions. If the comparison is true, the result is returned with 1; otherwise, the result is returned with 0. The IN operator is used to determine whether a variable's value is among the list of character or numeric values. When character values are used for comparison,

TABLE 1.2

Definition, Evaluation Order, and Examples of Operators

Priority	Evaluation Order	Type	Operator	Mnemonic Equivalent	Definition	Example
1	Right to left	Arithmetic	–		Negation prefix	`b = -a;`
		Arithmetic	**		Exponentiation	`b2 = a**2;`
		Logical	^ ~[a]	NOT	Logical not	`~z`
2	Left to right	Arithmetic	*		Multiplication	`c = a*b;`
		Arithmetic	/		Division	`c = a/b;`
3	Left to right	Arithmetic	+		Addition	`c = a+b;`
		Arithmetic	–		Subtraction	`c = a-b;`
4	Left to right	Comparison	<	LT	Less than	`a<10`
		Comparison	< =	LE	Less than or equal to	`a le b`
		Comparison	=	EQ	Equal to	`b = 2`
		Comparison	^ =	NE	Not equal to	`z ne 'A'`
		Comparison	> =	GE	Greater than or equal to	`g> = c`
		Comparison	>	GT	Greater than	`g gt a`
				IN	equal to one of a list	`z in ('A', 'B', 'G')`
5	Left to right	Logical	&	AND	Logical and	`a<10 & a>5`
6	Left to right	Logical	|!b	OR	Logical or	`z = 'A'|z = 'B'`

[a] Use either ^ or ~ for a "logical not" operator.

[b] Use either | or ! for a "logical or" operator.

they must be written in the same case as they appear in the original data set and must be enclosed in either single or double quotation marks.

An expression can be categorized into *simple*, *compound*, and *WHERE* (discussed in Section 1.4.1). A simple expression can contain no more than one operator, while a compound expression can contain more than one. You can connect one or more simple expressions with logical operators to form a compound expression.

A SAS function can also be considered an operator because a function performs a certain type of calculation and returns a value. More details about SAS functions are discussed in Chapter 9. You can use SAS functions in DATA step statements or in a WHERE expression. Here's the general form of a SAS function:

```
function-name(argument-1<,...argument-n>)
```

For example, to calculate the sum of a list of numeric variables, you can use the SUM function, which has the following form:

```
SUM(argument-1<,...argument-n>)
```

Program 1.4 reads five observations by using the DATALINES statement and creates two variables: SCORE_SUM1 and SCORE_SUM2. SCORE_SUM1 and SCORE_SUM2 are created by using the addition (+) operator and the SUM function, respectively. When adding two or more variables, the SUM function treats the missing values as zero; hence, the resulting value will not be missing. However, calculations that result by the use of the arithmetic operator will result in a missing value if any operand contains a missing value for an observation.

Program 1.4:

```
data score;
    input ID $ 1-4 score1 6-7 score2 9-10 score3 12-13;
    score_sum1 = score1 + score2 + score3;
    score_sum2 = sum(score1, score2, score3);
datalines;
629F 5  6  9
656F 6 10  9
711F 0  .  3
511F 9  4 10
478F .  5  3
;

title 'Adding Three Scores By Using + Operator and SUM
Function';
proc print data = score;
run;
```

Output from Program 1.4:

```
Adding Three Noise Scores By Using + Operator and SUM Function
Obs    ID    noise1   noise2   noise3   noise_ sum1   noise_ sum2
1     629F      5        6        9         20            20
2     656F      6       10        9         25            25
3     711F      0        .        3          .             3
4     511F      9        4       10         23            23
5     478F      .        5        3          .             8
```

The PRINT procedure immediately following the DATA step is used to print the contents of the data set. The TITLE statement is used to add the title for the output generated from using PROC PRINT. Further details concerning the use of PROC PRINT and the TITLE statement will be presented in Section 1.4.

1.3.2 Creating Variables Conditionally

In many situations, you need to create variables conditionally, which can be done by using the IF-THEN/ELSE statement. This topic is expanded upon in Chapter 2. The IF-THEN/ELSE statement has the following form:

```
IF expression THEN statement;
<ELSE statement;>
```

The *expression* in the IF-THEN/ELSE statement can be any valid SAS expression and often contains a comparison operator. In the situation where you need to have more than one expression to form a condition, you can use logical operators to create compound expressions. If the *expression* is evaluated to be true, the IF-THEN statement executes the *statement* after the keyword THEN for observations that are read from a data set. If there is an optional ELSE statement and if the *expression* is evaluated to be true, the ELSE statement is not executed; otherwise, the ELSE statement is executed. To use the ELSE statement, place it immediately after the IF-THEN statement.

Suppose that you would like to create a variable, named OVER10K. If income is greater then 10,000, OVER10K will be assigned to 1; otherwise OVER10K will be assigned to 0. There are multiple ways of creating this variable, for example:

```
if income > 10000 then over10k = 1;
if income < = 10000 then over10k = 0;
```

These two statements are legitimate; however, the second statement is executed even if the condition in the first statement is evaluated to be true. A more efficient way to create these variables is by writing the following code:

```
if income > 10000 then over10k = 1;
else over10k = 0;
```

By adding the ELSE statement, if the condition in the IF statement is evaluated to be true, the second statement will not be processed.

In SAS, a missing value is the smallest value. Thus, in the example above, if the INCOME variable contains any missing values, the observations with the missing values will be assigned to 0 for the OVER10K variable. How best to handle missing values when creating variables is discussed in Section 2.1.2.

1.4 Base SAS Procedures

Base SAS software provides a large selection of procedures that allow you to examine the contents of a SAS data set. Detailed documentation can be found in *Base SAS® 9.2 Procedures Guide* (2009). These procedures can be grouped into three categories: report writing, statistics, and utilities. This chapter covers only a few commonly used procedures: CONTENTS, SORT, PRINT, MEANS, and FREQ. The procedures in this section will be frequently used throughout the book as data-checking tools.

1.4.1 Common Statements in SAS Procedures: TITLE, BY, and WHERE Statements

This section covers only the two most commonly used statements, the BY and WHERE statements, to reduce repeated or redundant future explanations when introducing other procedures that use these statements. In addition, the TITLE statement, which is a global statement, is discussed in this section because it is frequently used when generating output.

Most SAS procedures will generate output in the SAS output window after procedures are submitted. By default, the generated output will have "The SAS System" as its title. To override the default title, you can use the TITLE statement, which has the following form:

```
TITLE <'text' | "text">;
```

The optional *text* is the new title that you may want to display for the procedure, which can be in either single or double quotations. You can place the TITLE statement within or outside the procedure. Once a title statement is submitted, it will be used for all subsequent output until you cancel or create a new title. To cancel a title, you can simply submit a TITLE statement without adding any text.

Many SAS statements have the same functionality across procedures, such as the BY and WHERE statements. Using the BY statement will categorize

the output by each level of the variable that is used in the BY statement. The BY statement has the following form:

```
BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
```

You can list more than one variable in a BY statement. The *variable* specified in the BY statement is called a *BY variable*. If the BY statement is used in a procedure, the output will be grouped by each level of the variable(s) used in the BY statement. The DESCENDING option is used to sort the *variable* that immediately follows the DESCENDING keyword in descending order. Here are some of the procedures that support the BY statement: REPORT, SORT (required), COMPARE, CORR, FREQ, TABULATE, MEANS, PLOT, TRANSPOSE, PRINT, UNIVARIATE, etc.

The WHERE statement subsets the input data set by specifying conditions a record has to meet for inclusion. Only those observations that meet the specified conditions will be used for processing. The WHERE statement has the following form:

```
WHERE where-expression;
```

The *where-expression* needs to be a legitimate arithmetic or logical expression. Here are some of the procedures that support the WHERE statement: REPORT, COMPARE, SORT, CORR, FREQ, TABULATE, MEANS, PLOT, TRANSPOSE, PRINT, UNIVARIATE, etc.

1.4.2 The CONTENTS Procedure

All SAS data sets contain a descriptor portion that contains information about the data set. PROC CONTENTS not only displays the descriptor information of a specified data set, it can also show the contents of a specified SAS library. The basic syntax for PROC CONTENTS is as follows:

```
PROC CONTENTS <DATA=SAS-file-specification> <VARNUM>;
RUN;
```

All SAS procedures contain a DATA=option, which is used to specify the name of the input data set for the procedure. Without using the DATA = option, SAS will use the most recently created data set. The *SAS-file-specification* can be in one of the following forms:

```
<libref.>SAS-data-set
<libref.>_ALL_
```

When <*libref.*>*SAS-data-set* is used, PROC CONTENTS displays the contents of *SAS-data-set* within the given library. Without specifying *libref* explicitly, *libref* is referred to as the WORK library. Using <*libref.*>_ALL_, PROC

CONTENTS will list all SAS data sets along with their information specified by the given library.

The VARNUM option prints the variable names by their created order. By default, variable names are listed alphabetically. Program 1.5 uses PROC CONTENTS to display the descriptor portion of the HEARING data set.

Program 1.5:

```
title 'The Contents of Hearing Data';
proc contents data = hearing varnum;
run;
```

Output from Program 1.5:

```
                The Contents of Hearing Data
                   The CONTENTS Procedure

Data Set Name       WORK.HEARING       Observations          34
Member Type         DATA               Variables             7
Engine              V9                 Indexes               0
Created             Sunday, January    Observation Length    40
                    29, 2012 09:36:46 AM
Last Modified       Sunday, January    Deleted Observations  0
                    29, 2012 09:36:46 AM
Protection                             Compressed            NO
Data Set Type                          Sorted                NO
Label
Data Representation WINDOWS_32
Encoding            wlatin1 Western
                    (Windows)

               Engine/Host Dependent Information
Data Set Page Size          4096
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            101
Obs in First Data Page      34
Number of Data Set Repairs  0
Filename                    C:\Users\Arthur\AppData\Local\
                            Temp\SAS Temporary
                            Files\_TD3988\hearing.sas7bdat
Release Created             9.0202M0
Host Created                W32_VSPRO

                 Variables in Creation Order
                  #    Variable   Type    Len
                  1    id         Char      4
                  2    race       Char      1
                  3    smoke      Char      7
```

```
4    Age         Num        8
5    Preg        Num        8
6    Hearing     Char       3
7    Income      Num        8
```

1.4.3 The SORT Procedure

The SORT procedure is used to order the data set observations by the values of one or more variables. In SAS, a missing value, either numeric or character, is the smallest value. The general syntax for PROC SORT is as follows:

```
PROC SORT <DATA=SAS-data-set> <OUT=SAS-data-set>;
   BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
RUN;
```

In the PROC SORT statement, the DATA=*SAS-data-set* option is used to identify the input SAS data set for sorting, and the OUT=*SAS-data-set* option is used to name the output data set that contains newly sorted data. If the OUT= option is omitted, the PROC SORT procedure will replace the original data with the sorted data.

At least one variable is required in the BY statement. By default, the SORT procedure orders the data set by the values of the first BY variable in ascending order. If there is more than one BY variable, the SORT procedure orders the observations having the same value of the first BY variable by the values of the second BY variable in ascending order. The sorting scheme will continue for all the BY variables. The DESCENDING option sorts the variable that follows the DESCENDING keyword in descending order. For example, Program 1.6 sorts RACE in ascending order, PREG in descending order, and AGE in descending order.

Program 1.6:

```
proc sort data = hearing out = hearing_sort;
    by race descending preg descending age;
run;
```

1.4.4 The PRINT Procedure

The PRINT procedure is one of the report-writing procedures that you can utilize to generate a customized report. Only the basic usage of PROC PRINT is presented in this section. More often, you will use PROC PRINT to simply print the observations for a specified SAS data set using either all or some of the variables. The basic syntax for PROC PRINT is as follows:

```
PROC PRINT <DATA=SAS-data-set> <NOOBS>;
   BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
```

```
    VAR variable(s);
    WHERE where-expression;
RUN;
```

By default, SAS prints the observation numbers in the output. Using the NOOBS option will suppress the observation numbers.

If the optional BY statement is used, PROC PRINT will create a separate section of the report for each level of the variables specified in the BY statement. Furthermore, the data set needs to be previously sorted (by the same order) as listed in the BY statement. You can use multiple variables in the BY statement.

The optional VAR statement is used to select variables to be printed in the output. If the VAR statement is omitted, all the variables will be printed in the output. The optional WHERE statement is used to subset the input data set by specifying certain conditions.

For example, Program 1.7 prints the contents of the HEARING_SMALL data set by each category of the RACE variable.

Program 1.7:

```
proc sort data = hearing_small out = hearing_small_sort;
    by race;
run;

title 'Print ID SMOKE and AGE variables by RACE';
proc print data = hearing_small_sort noobs;
    by race;
    var id smoke age;
run;
```

Output from Program 1.7:

```
            Print ID SMOKE and AGE variables by RACE

- - - - - - - - - - - - - - - - race = A- - - - - - - - - - - - - -
                id          smoke         Age
                747F        current       18
                796F        past          35
                745F        never         36

- - - - - - - - - - - - - - - - race = B- - - - - - - - - - - - - -
                id          smoke         Age
                135F        current       29

- - - - - - - - - - - - - - - - race = H- - - - - - - - - - - - - -
                id          smoke         Age
                629F        past          26
                713F        never         26
```

```
- - - - - - - - - - - - - - - - race = W- - - - - - - - - - - - - - -
                  id          smoke         Age
                  656F        never         26
                  711F        never         32
                  733F        current       17
                  982F        past          26
                  798F        never         19
                  494F        never         36
                  748F        never         34
                  904F        never         25
                  244F        never         28
                  184M        past          19
```

1.4.5 The MEANS Procedure

You can use PROC MEANS to calculate descriptive statistics for numerical variables across all observations in the input data set. PROC MEANS has the following form:

```
PROC MEANS <DATA=SAS-data-set> <MAXDEC=number>
           <statistic-keyword(s)>;
   BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
   CLASS variable(s);
   VAR variable(s);
   WHERE where-expression;
RUN;
```

The MAXDEC=*number* option in the PROC MEANS statement is used to specify the maximum number of decimal places to display statistics in the output. Without specifying this option, the default decimal place is seven. You can specifically request the statistics that you would like to compute by specifying statistic keyword(s) in the *statistic-keyword(s)* option. The commonly used keywords are NMISS, N, RANGE, STD, MAX, MEAN, MIN, VAR, MEDIAN, Q1, Q3, etc. Without specifying *statistic-keyword(s)*, the default statistics (N, MEAN, STD, MIN, and MAX) will be computed.

Similar to other procedures, the optional BY statement computes separate statistics for each BY group; likewise, the data set needs to be sorted previously in the same order as listed in the BY statement.

The optional CLASS statement is used to specify the variables whose values define the subgroup combinations for the analysis. The use of the CLASS statement is similar to the BY statement; however, the output created by using the CLASS statement has a different layout. You don't need to sort the *variable(s)* specified in the CLASS statement first before using the MEAN procedure.

The optional VAR statement is used to list the variable(s) for computing statistics. Omitting the VAR statement will result in SAS computing statistics for all the numeric variables that are not listed in the other statements.

The optional WHERE statement is used to subset the input data set by specifying certain conditions.

Program 1.8 calculates minimum, median, and maximum values for AGE variables just for the White and Black groups from the HEARING data set. These computed statistics are separated by each level of the SMOKE variable. The first PROC MEANS uses the CLASS statement for the SMOKE variable, and the second PROC MEANS uses the BY statement for the SMOKE variable. The output generated from Program 1.8 shows the differences between using the CLASS and BY statements.

Program 1.8:

```
title 'The Mean Procedure - Class Statement';
proc means data = hearing min median max maxdec = 2;
    where race = 'W' or race = 'B';
    class smoke;
    var age;
run;

proc sort data = hearing;
    by smoke;
run;

title 'The Mean Procedure - By Statement';
proc means data = hearing min median max maxdec = 2;
    where race = 'W' or race = 'B';
    by smoke;
    var age;
run;
```

Output from Program 1.8:

```
              The Mean Procedure - Class Statement
                      The MEANS Procedure
                    Analysis Variable : Age
smoke        N Obs         Minimum        Median         Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
current          6           17.00         28.50           33.00
never           14           19.00         27.00           36.00
past             5           19.00         21.00           34.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                The Mean Procedure - By Statement
- - - - - - - - - - - - - - - - - - - smoke = current- - - - - - - - - - -
                      The MEANS Procedure
                    Analysis Variable : Age
          Minimum        Median         Maximum
        - - - - - - - - - - - - - - - - - - - - - -
              17.00          28.50           33.00
        - - - - - - - - - - - - - - - - - - - - - -
```

```
- - - - - - - - - - - - - - - - smoke = never- - - - - - - - - - - - - -
                    Analysis Variable : Age
          Minimum        Median       Maximum
        - - - - - - - - - - - - - - - - - - - - - -
            19.00         27.00         36.00
        - - - - - - - - - - - - - - - - - - - - - -


- - - - - - - - - - - - - - - - smoke = past- - - - - - - - - - - - - -
                    Analysis Variable : Age
          Minimum        Median       Maximum
        - - - - - - - - - - - - - - - - - - - - - -
            19.00         21.00         34.00
        - - - - - - - - - - - - - - - - - - - - - -
```

1.4.6 The FREQ Procedure

PROC FREQ is often used to create one- to *n*-way frequency and contingency tables for categorical variables. Here is the basic syntax for PROC FREQ:

```
PROC FREQ <DATA=SAS-data-set>;
   BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
   WHERE where-expression;
   TABLES requests </options> ;
RUN;
```

The concepts for using the BY and WHERE statements are the same as the ones for using PROC PRINT and PROC MEANS.

In the TABLES statement, *requests* are used to specify frequency and contingency tables. A *request* can be one variable name or a list of variable names separated by asterisks (*). For example, to produce a one-way frequency table, simply use one variable name; to produce a two-way contingency, put an asterisk between two variables. Without using any *options* in the TABLES statement, PROC FREQ will list frequencies, cumulative frequencies, percentages of the total frequency, and cumulative percentages in the table for a one-way frequency table. For *n*-way tables, PROC FREQ will create a contingency table that includes cell frequencies, cell percentages of the total frequency, cell percentages of row frequencies, and cell percentages of column frequencies if you are not specifying any *options*. A few useful *options* for the TABLES statement to control displayed output are listed in Table 1.3.

By default, PROC FREQ will only display the total frequency of missing observations below each table. Program 1.9 illustrates the difference in the output when the computed variable contains missing values if no option is used and if the MISSING and MISSPRINT options are used.

TABLE 1.3

Selected *options* for the **TABLES** Statement for PROC FREQ

Options	Description
LIST	*n*-Way tables are created in list format
NOFREQ	Suppresses frequencies
NOCUM	Suppresses cumulative frequencies and percentages
NOPERCENT	Suppresses percentages
NOCOL	Suppresses column percentages
NOROW	Suppresses row percentages
MISSING	Treats missing values as a group
MISSPRINT	Displays missing value frequencies without percentages

Program 1.9:

```
title 'No option';
proc freq data = hearing;
    tables preg;
run;

title 'Using MISSING option';
proc freq data = hearing;
    tables preg/missing;
run;

title 'Using MISSPRINT option';
proc freq data = hearing;
    tables preg/missprint;
run;
```

Output from Program 1.9:

```
                          No option
                       The FREQ Procedure
                                          Cumulative    Cumulative
Preg     Frequency     Percent     Frequency      Percent
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   0           19       63.33            19        63.33
   1           11       36.67            30       100.00
                     Frequency Missing = 4

                       Using MISSING option
                       The FREQ Procedure
                                          Cumulative    Cumulative
Preg     Frequency     Percent     Frequency      Percent
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   .            4       11.76             4        11.76
   0           19       55.88            23        67.65
   1           11       32.35            34       100.00
```

```
                    Using MISSPRINT option
                      The FREQ Procedure
                                    Cumulative    Cumulative
Preg     Frequency     Percent      Frequency      Percent
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   .            4           .              .            .
   0           19       63.33             19        63.33
   1           11       36.67             30       100.00
                   Frequency Missing = 4
```

Program 1.10 illustrates the difference in the output when using (or not using) the LIST option.

Program 1.10:

```
title 'Without LIST option';
proc freq data = hearing;
    tables preg*race*smoke;
run;

title 'With LIST option';
proc freq data = hearing;
    tables preg*race*smoke/list;
run;
```

Output from Program 1.10:

```
                    Without LIST option
                     The FREQ Procedure
                   Table 1 of race by smoke
                   Controlling for Preg = 0
      race      smoke
      Frequency|
      Percent  |
      Row Pct  ,
      Col Pct  |current |never  |past   | Total
      - - - - -^- - - - ^- - - -^- - - -^
      A        |      2 |     1 |     1 |     4
               |  10.53 |  5.26 |  5.26 | 21.05
               |  50.00 | 25.00 | 25.00 |
               |  33.33 | 11.11 | 25.00 |
      - - - - -+- - - - + - - - + - - - +
      B        |      1 |     2 |     0 |     3
               |   5.26 | 10.53 |  0.00 | 15.79
               |  33.33 | 66.67 |  0.00 |
               |  16.67 | 22.22 |  0.00 |
      - - - - -+- - - - + - - - + - - - +
      H        |      0 |     1 |     1 |     2
               |   0.00 |  5.26 |  5.26 | 10.53
               |   0.00 | 50.00 | 50.00 |
               |   0.00 | 11.11 | 25.00 |
```

W	3	5	2	10
	15.79	26.32	10.53	52.63
	30.00	50.00	20.00	
	50.00	55.56	50.00	
Total	6	9	4	19
	31.58	47.37	21.05	100.00

Without LIST option
The FREQ Procedure
Table 2 of race by smoke
Controlling for Preg = 1

race Frequency Percent Row Pct Col Pct	smoke current	never	past	Total
A	0	1	0	1
	0.00	10.00	0.00	10.00
	0.00	100.00	0.00	
	0.00	12.50	0.00	
B	1	0	0	1
	10.00	0.00	0.00	10.00
	100.00	0.00	0.00	
	100.00	0.00	0.00	
H	0	1	0	1
	0.00	10.00	0.00	10.00
	0.00	100.00	0.00	
	0.00	12.50	0.00	
W	0	6	1	7
	0.00	60.00	10.00	70.00
	0.00	85.71	14.29	
	0.00	75.00	100.00	
Total	1	8	1	10
	10.00	80.00	10.00	100.00

Frequency Missing = 1

With LIST option
The FREQ Procedure

Preg	race	smoke	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	A	current	2	6.90	2	6.90
0	A	never	1	3.45	3	10.34

```
0  A      past          1     3.45          4     13.79
0  B      current       1     3.45          5     17.24
0  B      never         2     6.90          7     24.14
0  H      never         1     3.45          8     27.59
0  H      past          1     3.45          9     31.03
0  W      current       3    10.34         12     41.38
0  W      never         5    17.24         17     58.62
0  W      past          2     6.90         19     65.52
1  A      never         1     3.45         20     68.97
1  B      current       1     3.45         21     72.41
1  H      never         1     3.45         22     75.86
1  W      never         6    20.69         28     96.55
1  W      past          1     3.45         29    100.00
                  Frequency Missing = 5
```


1.5 Subsetting Data by Selecting Variables

In SAS, *subsetting* by data refers to selecting a portion of a SAS data set by keeping only certain variables or selected observations (or both). This section covers how to subset a data set by selecting certain variables. Subsetting a data set by selecting observations is covered in Chapter 3.

When you subset data by selecting variables, you often need to keep or drop a list of variables. SAS provides a convenient method that allows you to refer to a list of variables by using the SAS *variable-lists* notation. Variable list notation can be used in many SAS statements and data set options. There are four types of variable-lists: *numbered range lists*, *name range lists*, *name prefix lists*, and *special SAS name lists*.

The variables that are referenced in *numbered range lists* must have the same name, except for the last one, two, or more characters. The last one or more characters need to be consecutive numbers. You need to use a single dash to connect the beginning and ending variables. For example, writing VAR1 - VAR3 is equivalent to writing VAR1, VAR2, VAR3 or VAR1 VAR2 VAR3.

To use *named range lists*, you need to know the creation order of the data set, which can be found via the VARNUM option from PROC CONTENTS. For example, ID -- PREG refers to all the variables in order of variable creation from ID to PREG.

Use *name prefix lists* to refer to all the variables that begin with a specified character string. For example, DRUG: refers to all the variables that begin with "DRUG."

Last, *special SAS name lists* uses special SAS names, _NUMERIC_, _CHARACTER_, and _ALL_, to refer to all numeric, character, or any variables that are already defined in the current DATA step, respectively. Examples of using *variable-lists* are summarized in Table 1.4.

TABLE 1.4

Examples of SAS Variable-Lists

Variable List	Example	Equivalence
Numbered range lists	`VAR1 - VAR3`	`VAR1, VAR2, VAR3` or `VAR1 VAR2 VAR3`
Named range lists	`ID -- PREG`	All the variables in order of variable creation from ID to PREG
	`ID -NUMERIC- PREG`	All numeric variables from ID to PREG
	`ID -CHARACTER- PREG`	All character variables from ID to PREG
Name prefix lists	`DRUG:`	All the variables that begin with "DRUG"
Special SAS name lists	`_NUMERIC_`	All numeric variables already defined in the current DATA step
	`_CHARACTER_`	All character variables already defined in the current DATA step
	`_ALL_`	All variables already defined in the current DATA step

1.5.1 Selecting Variables with the KEEP= Data Set Option or KEEP Statement

To create a data set by selecting a number of variables, you can use either the KEEP= data set option or the KEEP statement. Using data set options allows you to specify certain actions that apply to either the input or output data sets. To use the data set options, you need to place the data set options in parentheses after the data set name. If you need to specify more than one option, you need to separate them with spaces. The general syntax for data set options is as follows:

```
(option-1=value-1<...option-n=value-n>)
```

You can select the variables that you would like to keep by using the KEEP= data set option to control either the input data set or the output data set. The KEEP = data set option has the following form:

```
KEEP=variable-list
KEEP=variable-1 <...variable-n>
```

In the KEEP= option, you can either list the variables that you would like to keep individually (separated by a space) or use the *variable-list* notation.

Specifying the KEEP= data set option after the data set name in the SET statement reads only the specified variables from the input data set. Specifying the KEEP= data set option in the DATA statement controls which variables are written to the output data set. For example, Program 1.11 creates

a data set by reading the variables ID, SMOKE, and AGE from the HEARING data set. Using the KEEP= option after DAT1 in the DATA statement will yield the same result; however, specifying the KEEP= option in the SET statement in this example is more efficient because SAS reads only the desired variables.

Program 1.11:

```
data dat1;
    set hearing(keep = id smoke age);
run;
```

The easiest way to verify whether the DAT1 data set was created correctly is to check your SAS log. Based on the SAS log from Program 1.11, you can see that DAT1 contains 34 observations and 3 variables. You can also use PROC CONTENTS to examine the contents of the newly created data set.

Log from Program 1.11:

```
136 data dat1(keep = id smoke age);
137     set hearing;
138 run;

NOTE: There were 34 observations read from the data set
      WORK.HEARING.
NOTE: The data set WORK.DAT1 has 34 observations and 3
variables.
NOTE: DATA statement used (Total process time):
      real time           0.07 seconds
      cpu time            0.01 seconds
```

Alternatively, you can select variables in the DATA step by using the KEEP statement, which has the following form:

```
KEEP variable-list;
KEEP variable-1 <...variable-n>;
```

Notice that there will be no equal sign after the KEEP keyword in the KEEP statement. Program 1.12 creates the same data set by using the KEEP statement instead of the KEEP= data set option.

Program 1.12:

```
data dat1;
    set hearing;
    keep id smoke age;
run;
```

1.5.2 Selecting Variables with the DROP= Data Set Option or DROP Statement

You can also use the DROP= data set option or DROP statement to select variables that you want to remove when creating a data set. The DROP= data set option has the following form:

```
DROP=variable-list
DROP=variable-1 <...variable-n>
```

Program 1.13 selects the same variables (ID, SMOKE, and AGE) from the HEARING data set by using the DROP= option.

Program 1.13:

```
data dat2;
    set hearing(drop = race preg -- income);
run;
```

In Program 1.13, PREG -- INCOME (a named range list) is equivalent to listing all the variables between PREG and INCOME.

You can also use the DROP statement to select variables, which has the following form:

```
DROP variable-list;
DROP variable-1 <...variable-n>;
```

Program 1.14 achieves the same result by using the DROP statement.

Program 1.14:

```
data dat2;
    set hearing;
    drop race preg -- income;
run;
```

If you have more variables to drop than you have to keep, it will be easier for you to use the KEEP= option or the KEEP statement to save you some typing. Conversely, if you have more variables to keep than to drop, you should use the DROP= option or the DROP statement.

1.5.3 Where to Specify the DROP= and KEEP= Data Set Options and DROP/KEEP Statements

You can specify the DROP= and KEEP= data set options in either the DATA statement or the SET statement after the name of the SAS data set, depending upon whether or not you want to process values of the variables in the DATA step. For example, Program 1.15 creates a data set that contains only two

variables, ID and INCOME_HI, which is used to indicate whether income level is above or below $50,000.

Program 1.15:

```
data dat3 (drop = income);
    set hearing (keep = id income);
    if income > 50000 then income_hi = 1;
    else income_hi = 0;
run;
```

In Program 1.15, the KEEP= option is used in the SET statement to read the ID and INCOME variables from the input data HEARING. The INCOME variable is required because you need to use the INCOME variable in the DATA step to create a new variable (INCOME_HI) by using the IF-THEN/ELSE statement. Because you don't need the INCOME variable in the output data set, you can then specify the INCOME variable in the DROP= option in the DATA statement.

In DATA steps, the DROP= and KEEP= data set options can be used in either the SET statement to apply to the input data set or the DATA statement to apply to the output data set. However, the KEEP and DROP statements apply only to output data sets. In DATA steps, when you create multiple output data sets, you can use the DROP= or KEEP= data set options to write different variables to different data sets. The DROP or KEEP statements apply to all output data sets. For example, Program 1.16 uses one DATA step to create two SAS data sets: DAT4 and DAT5. DAT4 is created by keeping only the ID, RACE, and SMOKE variables, and DAT5 is created by keeping only the ID, AGE, and PREG variables. The KEEP= option is used to control which variables are retained after each data set name.

Program 1.16:

```
data dat4 (keep = id race smoke)
     dat5 (keep = id age preg);
    set hearing;
run;
```

You can use the DROP= or KEEP= data set options only in PROC steps, not in the DROP and KEEP statements.

1.6 Changing the Appearance of Data

When you encounter a new data set, you will often notice that the variable names are frequently very brief. Shorter but more meaningful variable names are often a preferred format because they are easy to type and save

storage space. This idea also applies to variable values. For example, the numerical value 1 is often used to represent "yes" and 0 is used to represent "no" for some categorical variables.

Labeling variable names and formatting variable values are often conducted by SAS programmers. The purpose of doing so is to provide a more appealing look when printing the output from a procedure. Labeling and formatting variables only modify the appearance of the variables' names and values; these actions have no effect whatsoever on the variables' original names and values.

1.6.1 Labeling Variables

To create a descriptive label for a variable, you can use the LABEL statement, which has the following form:

```
LABEL variable-1=label-1... <variable-n=label-n>;
```

You can label one or more variables within one LABEL statement and separate them by space(s). The *variable(s)* in the LABEL statement are the names of the variables that you want to label and *label(s)* are their corresponding labels. The assigned labels can contain blanks and can be no more than 256 characters. If the assigned label contains semicolons (;) or equal signs (=), you can enclose the label in either single or double quotations. If the label contains a single quote ('), you must enclose the labels in double quotations.

To remove labels from variables, you need to use the LABEL statement and associate the variables with a single blank space in quotation marks. Here's the syntax for removing labels:

```
LABEL variable-1=' '... <variable-n=' '>;
```

You can use the LABEL statement in either the DATA step or PROC step. When using a LABEL statement in a DATA step, you associate labels with the variables permanently; in this situation, the assigned label becomes one of the variable's attributes. When using a LABEL statement in the PROC step, the assigned label is available only to the output that the current procedure generated. The LABEL statement in the PROC step does not add permanent labels in the input data set except when the LABEL statement is used with the MODIFY statement in the DATASETS procedure.

Program 1.17 creates a new data set, HEARING1_1. In the HEARING1_1 data set, the variables HEARING and INCOME are assigned with permanent labels. When using PROC PRINT to display the data set contents, in order to see variable labels in the output, you need to use the LABEL option in the PROC PRINT statement. The OBS= option is used to specify the last observation that SAS needs to process the input data set. Thus, using OBS=5 restricts the printed output to the first five observations. Similarly, you can

use the FIRSTOBS= option to specify the first observation that SAS processes in an input data set.

Program 1.17:

```
data hearing1_1;
    set hearing;
    label hearing = Hearing Loss
          income = "People's Income";
run;

title 'Assigning Labels Permanently';
proc print data = hearing1_1(obs = 5) label ;
    var hearing income;
run;

proc contents data = hearing1_1;
run;
```

Output from Program 1.17:

```
Assigning Labels Permanently
                Hearing    People's
 Obs             Loss       Income
   1              no         29000
   2                         28700
   3                         59000
   4              no        120000
   5                         29000
```

Partial Output from PROC CONTENTS from Program 1.17:

```
Alphabetic List of Variables and Attributes
#    Variable    Type     Len    Label
4    Age         Num        8
6    Hearing     Char       3    Hearing Loss
7    Income      Num        8    People's Income
5    Preg        Num        8
1    id          Char       4
2    race        Char       1
3    smoke       Char       7
```

Based on the partial output from PROC CONTENTS from Program 1.17, you can see that the HEARING and INCOME variables have permanent labels that are under the LABEL attribute column.

Program 1.18 calculates mean, median, and standard deviation for the INCOME variable by each category of ethnicity. In PROC MEANS, the LABEL statement is used to assign the RACE variable a temporary label,

"Ethnicity." Notice that the label attribute for the INCOME variable is listed in the output because the INCOME variable is assigned a permanent label in the DATA step in Program 1.17.

Program 1.18:

```
title 'Assign temporary label to RACE variable';
proc means data = hearing1_1 mean median std;
    label race = Ethnicity;
    class race;
    var income;
run;
```

Output from Program 1.18:

```
            Assign temporary label to RACE variable
                      The MEANS Procedure
          Analysis Variable : Income People's Income
Ethnicity      N Obs        Mean         Median         Std Dev
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
A                  5    61420.00       39100.00        45499.25
B                  5    61200.00       39100.00        44726.45
H                  4    44800.00       37000.00        21332.92
W                 20    53575.00       48550.00        25353.54
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

1.6.2 Formatting Variable Values Using SAS FORMATS

A *format* is an instruction to tell SAS how to write data values. You can use formats to control the written appearance of data values in the output. Variable values can be formatted by using either SAS formats or user-defined formats that are created from the FORMAT procedure. This section covers only the SAS format; the PROC FORMAT is covered in Chapter 10. To associate formats with variables, you need to use the FORMAT statement, which has the following form:

```
FORMAT variable-1 <... variable-n> format
       variable-1 <... variable-n> format;
```

In the FORMAT statement, you associate a *format* with one or more *variables* that are listed before the specified *format.* That is to say, you can associate the same format with multiple variables in a single FORMAT statement. You can also associate different formats with different variables.

SAS has a large selection of formats that allow you to format character and numeric variables. To format standard character data, you will use the following format:

```
$w.
```

The dollar sign ($) is required for formatting character values. The *w* field is used to specify total width of the character values. The period (.) is a required component of the format.

There are many formats to write nonstandard character values. The difference between standard and nonstandard character formats is that nonstandard character formats contain a keyword in between the dollar sign and the *w* field. For example, the *$UPCASEw.* format converts character values to uppercase when writing the data value, the *$QUOTEw.* format writes data values enclosed in double quotation marks, etc.

To write numeric values in the standard format, you need to use the following form:

```
w.d
```

The first field *w* is used to specify total width of the numerical values, including the decimal point. The second field is a period (.). The last field *d* is optional; it is used to specify the number of digits to the right of the decimal point in the numeric value. If *d* is omitted, *w.d* format writes the value without a decimal point.

A nonstandard numeric format contains a keyword in front of the *w* field. For example, the *DOLLARw.d* format will write a number as follows: a leading dollar sign in the starting position, a comma that separates every three digits, and a period that separates the decimal fraction. When using the *DOLLARw.d* format, the *w* field must be large enough to include the dollar sign, comma, and the decimal points.

If you associate a variable(s) with a format in the DATA step, the associated format will become permanent and the format will become the variable's attribute. Like the LABEL statement, if you use a FORMAT statement in some PROC steps, the associated format will only be available for the output of the current procedure.

To disassociate a format from a variable, use the variable in a FORMAT statement without specifying any formats in a DATA step. Here is the syntax to disassociate a format from a variable:

```
FORMAT variable-1 <... variable-n>;
```

Program 1.19 associates the SMOKE and HEARING variables with the $UPCASE5. format and the INCOME variable with the DOLLAR11.2 format. Because the width of the $UPCASE5. format is not long enough, only the first five characters in the SMOKE variable are printed in the output.

Program 1.19:

```
data hearing1_2;
    set hearing;
    format smoke hearing $upcase5. income dollar11.2;
run;
```

```
title 'Assigning formats to variable';
proc print data = hearing1_2(firstobs = 6 obs = 12);
    var smoke hearing income;
run;
```

Output from Program 1.19:

```
                Assigning formats to variable
Obs    smoke    Hearing        Income
  6    CURRE                $19,000.00
  7    CURRE                $23,900.00
  8    CURRE                $39,000.00
  9    CURRE                $39,100.00
 10    NEVER       NO       $48,000.00
 11    NEVER       NO       $30,000.00
 12    NEVER                $25,000.00
```

Exercises

Exercise 1.1. Create a SAS data set that is based on HEARING.SAS7BDAT. The information about downloading the testing data sets can be found in the Preface.

1. In this data set, create a variable M_INCOME (monthly income) based on the variable INCOME (yearly income).
2. Create another variable, HEARING_INFO, which is based on the variable HEARING. If the HEARING variable contains missing values, HEARING_INFO will be assigned with value 1; otherwise, HEARING_INFO will be assigned with value 0.
3. Label the variable M_INCOME with "monthly income".
4. Format the M_INCOME variable with the FRACT9. format. Note: You might need to check the SAS documentation for this format. If you have difficulty finding this document, you can search "Fractw. Format SAS" using your preferred online search engine.
5. In the resulting data set, you need to keep the ID, HEARING, HEARING_INFO, INCOME, and M_INCOME variables.

Once this data set is created, use PROC FREQ to create a two-way contingency table for the variables HEARING and HEARING_INFO to confirm that HEARING_INFO was created correctly. Explore the NOPERCENT, NOCOL, and NOROW options for this procedure.

TABLE 1.5

Variable Information for EX1_3.DAT

Variable Name	Description	Locations	Variable Type
ID	Subject's ID	Columns 1–4	Character
GENDER	Gender	Columns 6–11	Character
SMOKE	Smoking status	Columns 13–15	Character
AGE	Age	Columns 17–18	Numeric
DISEASE	Disease status	Column 20	Numeric

Exercise 1.2. The MEANS procedure is introduced in Section 1.4.5. There are other statements in this procedure that were not presented in this section. For example, you can use the OUTPUT statement to output the calculated statistics to a SAS data set. The detailed explanation of this statement can be found in the SAS documentation ("The MEANS Procedure" article). For this exercise, create a SAS data set, HEARING_STATS, via the OUTPUT statement. The HEARING_STATS data set will contain the minimum, first quartile (Q1), median, third quartile (Q3), and maximum values for the AGE and INCOME variables by each category of the RACE variable in the HEARING data set. Because the requested statistics are outputted to a SAS data set, you can use the NOPRINT option in the PROC MEANS statement to suppress the output being listed in the output window.

Exercise 1.3. Create two SAS data sets by using one DATA step. These two SAS data sets are created by reading the raw data set EX1_3.DAT. The description of the data set is listed in Table 1.5. One of the SAS data sets that you are creating will contain only the ID, GENDER, and SMOKE variables; the other one will contain only the ID, AGE, and DISEASE variables.