8 Data Input and Output

8.1 Introduction to Reading and Writing Text Files

A SAS® data set is often created by reading data directly from an external text file. Observations (rows) and variables (columns) in the SAS data set correspond to records and the fields in the text file. You can go in the opposite direction and create a text file from a SAS data set. This section provides an overview for reading and writing text files, as well as an introduction to SAS informats and formats.

8.1.1 Steps for Reading Text Files

To read an external text file into SAS, you need to use an INFILE statement followed by the INPUT statement. The INFILE statement has the following form:

```
INFILE file-specification <options>;
```

File-specification is used to specify the location of the input data set. It can take the form of *'external-file'* (with quotes) or *fileref* created with a FILENAME statement. The simple form of the FILENAME statement is as follows:

```
FILENAME fileref 'external-file'
```

The FILENAME statement associates fileref with the external file. The following code shows how INFILE and FILENAME work together:

```
filename ex 'c:\example.txt';
data sasdat;
    infile ex;
    input...;
run;
```

If you don't want to use INFILE with FILENAME, the following is also perfectly acceptable:

```
data sasdat;
    infile 'c:\example.txt';
    input...;
run;
```

In addition, if you use the DATALINES or DATALINES4 statements, the INFILE statement can be omitted when options are not specified. To specify options in the INFILE statement with DATALINES or DATALINES4, just enter either *file-specification* as

```
INFILE DATALINES | DATALINES4 <options>;
```

Once the location of the external file is identified by the INFILE statement, you need to use the INPUT statement to read the data.

SAS provides many methods of reading external text files, which include the *column*, *formatted*, *list*, and *named* input methods. These input methods can also be mixed when reading an external file. In addition to these input methods, many options in the INFILE statement assist in importing the data. Choosing the correct input method and the option in the INFILE statement depends upon how the data is arranged or formatted in the text file. Thus, before reading a text file into SAS, one needs to examine the data carefully and then choose the correct method based on how data are stored in the text.

There are many aspects of the data that you need to inspect. Here is a checklist that should help you out:

- Are the data arranged in a fixed field or free format?
- Is the length of each record the same or varied?
- Does any numeric data field contain any nonstandard numeric values?
- Are delimiters used for separating each field?
- Is the maximum length of character data field greater than 8 bytes?
- Does the character data field contain any embedded blanks?
- Are the missing values located in the beginning, middle, or end of the record?
- Are you creating one observation based on multiple records or are you creating multiple observations based on one record?

The text files shown in this chapter contain data formats that represent the most commonly encountered scenarios. Some rarely seen data formats will not be presented. For example, the *named* input method is not presented. You can use the *named* input method to read a data file that contains values preceded by the name of the variable and an equal sign (=).

8.1.2 Steps for Writing Text Files

Creating a text file from a SAS data set often starts with the FILE statement and is followed by the PUT statement in the DATA step. You use the FILE statement to specify the name of the output file that you want to create, which has the following form:

```
FILE file-specification <options>;
```

If *file-specification* equals an *'external file'* or a *fileref*, output from the PUT statement (introduced in Chapter 3) is written to an external file. On the other hand, if you specify PRINT as the *file-specification*, SAS will redirect the PUT output to the output window. As with INPUT, SAS provides four types of PUT statements: *column*, *formatted*, *list*, and *named.* Again, the *named* output method, like the *named* input method, is not covered in this chapter.

8.1.3 Data Informat

An informat tells SAS how to read a field from a text file. Its use depends upon the type of INPUT statement being used. The SAS informat takes the following general form:

```
<$>informat<w>.<d>
```

The optional dollar sign ($) is used for reading character data. The *informat* component is the name of the informat. The *w* value is used to specify the width (or the number of columns) of the input field. The *d* value is used for dividing the input numeric data by 10 to the power of *d*.

The *$w.* informat is for reading standard character data values. When using the *$w.* informat, SAS trims the leading blanks of the character value and left-aligns the values. The *$w.* informat treats blank fields or a single period (.) as missing and converts the period (.) to a blank when the missing value is stored. If, however, the *$w.* informat encounters two consecutive periods in the external file, the field is treated as a non-missing character value.

SAS also provides a wide variety of informats for reading other types of character data. For example, if you do not want to trim leading blanks or do not want to treat a single period as a missing value when reading in character values, you can use the *$CHARw.* informat. Or if you want to convert all the character data to uppercase, you can use the *$UPCASEw.* informat. Using the *$QUOTEw.* informat removes matching quotation marks from character data, and so on.

In SAS, standard numeric values can contain only numbers, decimal points, numbers in scientific notations, and plus or minus signs. Nonstandard numeric values can consist of fractions, values that contain special characters (such as the percent sign, %), the dollar sign ($), commas (,), etc.

TABLE 8.1

Examples of Using *w.d* Informat

Raw Data Value	Informat	Value That Read into SAS	Note
3.14159	`7.`	3.14159	
3.14159	`6.`	3.1415	Only the first 6 fields are read
3.14159	`7.5`	3.14159	5 in the informat is ignored
314159	`6.5`	3.14159	5 is used to specify 5 decimal points

The *w.d* informat is used for reading standard numeric data. When using the *w.d* informat, the *d* value will be ignored if the input data contains decimal points. The *w.d* informat can handle scientific notation and will interpret a single period as a missing value. Table 8.1 shows by example how the *w.d* informat works.

A commonly used informat for reading nonstandard numeric values is the *COMMAw.d* informat. The *COMMAw.d* informat removes embedded characters, including commas, blanks, dollar signs ($), percent signs (%), dashes (-), and closed parentheses, from an input data field. The *COMMAw.d* informat also converts the open parenthesis at the beginning of a field to a minus sign.

8.1.4 Data Format

A format tells SAS how to write a SAS data set value out to a text file. The general form of a SAS format is as follows:

```
<$>format<w>.<d>
```

The dollar sign ($) is used to indicate a character format. The *format* is the name of the format, which can be either a SAS format or a user-defined format. The *w* value is the format width, which is the number of columns in the output data in most cases. The *d* value is used to indicate the decimal scaling factor in the numeric formats. A format always contains a period (.) as a part of the format name.

Chapter 1 introduced some of the commonly used formats, such as the standard character format ($*w*.), standard numeric format (*w.d*), and a few nonstandard character and numeric formats. Readers can refer to SAS documentation for other types of formats.

8.1.5 SAS Date and Time Values

In SAS, date or time values are stored as numeric values. A SAS date value, which accounts for all leap-year days, is a value that represents the number of days between January 1, 1960, and a specified date. Dates after

January 1, 1960, are positive numbers, and dates before January 1, 1960, are negative numbers. Because dates are stored as numbers, you can easily perform any kind of computation on dates, such as calculating the number of days between two specified dates.

SAS time value is a value that stands for the number of seconds since midnight of the current day and ranges between 0 and 86,400. SAS datetime values correspond to the number of seconds between midnight January 1, 1960, and a specified second within a specified date.

SAS provides a large selection of informats to read different notations of date and time and stores the data to a SAS date, time, or datetime value. Date and time values can be read into SAS successfully provided that the date and time are separated by either a blank or the following delimiters: ! # $% & () * + −./: ; < = > ? [\] ^ _ {|} ~. However, only one type of delimiter can be used; for example, 10/01/49 is a valid input date value, while 10-01/49 is not.

The *DATEw.*informat is used for reading in date values in the form of *ddmmmyy* or *ddmmmyyyy*. The *dd* component is an integer ranging from 01 to 31 that denotes the day of the month, whereas *mmm* is the first three letters of the month. The *yy* or *yyyy* component is the year in either a two-digit or four-digit format. Each of the three components can be separated by blanks or the delimiter that was listed in the previous paragraph. The *MMDDYYw.* informat is similar to the *DATEw.* informat except that the *MM* value, representing month, is an integer from 01 to 12.

Examples of date informats are listed in Table 8.2. Readers can refer to SAS documentation for other types of date and time informats.

A SAS date value can contain either a two-digit or four-digit year component. SAS interprets the two-digit year based on a 100-year span that starts with the value specified in the YEARCUTOFF = option. By default, the YEARCUTOFF = option is set to the year 1920, meaning that the 100-year span is from 1920 to 2019.

Thus, a two-digit value of 49 will be interpreted as 1949. If you set the YEARCUTOFF = option to 1950, then the 100-year span is from 1950 to 2049. Thus, 49 will be treated as 2049. To avoid confusion, you should try to avoid using a two-digit year option if possible. Also, if your data spans more than 100 years, you must use the four-digit year option.

TABLE 8.2

Examples of Using Date Informats

Date Input	Informat	Result
01oct1949	`date9.`	-3744
01oct49	`date7.`	-3744
01-oct-1949	`date11.`	-3744
10-01-49	`mmddyy8.`	-3744
10/01/1949	`mmddyy10.`	-3744
01 10 1949	`ddmmyy10.`	-3744

TABLE 8.3

Examples of Date Formats

Numeric Value	Format	Formatted Value
-3744	`date5.`	01OCT
-3744	`date7.`	01OCT49
-3744	`date9.`	01OCT1949
-3744	`date11.`	01-OCT-1949
-3744	`mmddyy6.`	100149
-3744	`mmddyy8.`	10/01/49
-3744	`mmddyy10.`	10/01/1949
-3744	`mmddyyp10.`	10.01.1949

Dates and times are stored as numbers in SAS, so you need to use a format to see what the numbers represent. For example, *DATEw.* writes date values in the form of *ddmmmyy*, *ddmmmyyyy*, or *dd-mmm-yyyy*, where *dd* is an integer that represents the day of the month, *mmm* is the first three letters of the month, and *yy* or *yyyy* is a two-digit or four-digit integer that represents the year. The *w* value represents the width of the output field.

Another commonly used date format is the *MMDDYYw.* format that writes date values in the form of *mmdd<yy>yy* or *mm/dd/<yy>yy*, where *mm* is an integer value that represents month, *dd* is an integer value that represents day, and *yy* or *yyyy* is a two-digit or four-digit integer that represents the year. The forward slash is a separator between month, day, and year.

The *MMDDYYxw.* format is similar to *MMDDYYw.* except that you can use the following *x* field to specify a separator: B (a blank), C (a colon), D (a dash), N (no separator), P (a period), and S (a slash). Some examples of date formats are listed in Table 8.3.

You can use the *DATETIMEw.d* format to write datetime values in the form of *ddmmmyy:hh:mm:ss.ss.* The *hh* value is an integer that represents the hour in 24-hour clock time, *mm* is an integer that represents the minutes, and *ss.ss* is the number of seconds to two decimal places. The *w* value is used to specify the width of the output field, and the optional *d* value is used to indicate the number of digits to the right of the decimal point in seconds value.

8.2 Reading Text Files

The INPUT statement is used for reading the external text file into SAS. The INPUT statement first copies the data from the external text file to the input buffer and then controls how values are transferred from the input buffer into the program data vector (PDV). The DATA step execution for reading a text file is illustrated in Chapter 3.

8.2.1 Column Input

The column input method was discussed in Chapter 1 and again in Chapter 3. You can use the column input method only to read a text file that contains variables in a fixed field with character or standard numeric values. The syntax for reading a text file using the column input method is as follows:

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

During the execution phase of the DATA step, each record from the text file is read into the input buffer. SAS uses the input pointer to read data from the input buffer to the PDV that follows the instruction from the INPUT statement, which then creates observations for the SAS data set. The compilation and execution phases for reading an external text file via the column input method were shown in Program 3.1 in Chapter 3.

Using the column input method does not require that each field in the text file be separated by a delimiter. The character values in the text file can contain embedded blanks. The input values in the text file can also be read in any order. Any blank fields in the text file are treated as missing by the INPUT statement. In addition, the INPUT statement treats a single period in either numeric or character fields as missing.

A text file that contains records of varied length will cause problems when the column input method is used. For example, CH8_1.TXT contains data in a fixed-field format. However, the length of each record varies, because the last variable CITY is left-aligned, and CITY names have different lengths. A description of the data is presented in Table 8.4.

By default, SAS uses the FLOWOVER option in the INFILE statement. That is to say, if a record is shorter than expected, the INPUT statement will read the values from the following data record. To read the data correctly, you can use the TRUNCOVER option in the INFILE statement. TRUNCOVER accommodates variable-length records.

Alternatively, you can use the PAD option in the INFILE statement. The PAD option pads each record with blanks so that all the records have the same length. Using either the TRUNCOVER or the PAD option is especially useful when you read records that contain missing values at the end. Program 8.1 uses the column input method along with the TRUNCOVER option to read CH8_1.TXT.

TABLE 8.4
Field Description for CH8_1.TXT

Variable Name	Locations	Variable Type
Name	Columns 1–14	Character
Gender	Column 16	Character
Age	Columns 17–18	Numeric
City	Columns 20–30	Character

CH8_1.txt:

```
123456789012345678901234567890
James Miller   M26 New York
Patricia Davis F30 Phoenix
John Jackson   M31 Dallas
Mary Martinez  F32 Los Angeles
```

Program 8.1:

```
data ex8_1;
    infile "W:\SAS_Book\dat\ch8_1.txt" truncover;
    input Name   $ 1 - 14
        Gender   $ 16
           Age     17 - 18
          City   $ 20 - 30;
run;

title 'Using the column input method and the TRUNCOVER option';
proc print data = ex8_1;
run;
```

Output from Program 8.1:

```
    Using the column input method and the TRUNCOVER option
     Obs        Name          Gender     Age     City
      1     James Miller        M         26     New York
      2     Patricia Davis      F         30     Phoenix
      3     John Jackson        M         31     Dallas
      4     Mary Martinez       F         32     Los Angeles
```

8.2.2 Formatted Input

If the input data set in fixed field contains nonstandard numeric values, you can use the formatted input method to read data. Here is the syntax of the formatted input method:

```
INPUT <pointer-control> variable informat;
```

The *pointer-control* can be either column or line pointer-control. Column pointer-control is used to move the input pointer in the input buffer to a specified column. Here are two commonly used column pointer-controls:

```
@n
+n
```

By default, the column input pointer is located at column 1 in the input buffer. The @*n* pointer-control moves the input pointer to column *n* in the input buffer, while the +*n* moves the pointer *n* columns forward to a column relative to the current position.

TABLE 8.5

Field Description for CH8_2.TXT

Variable Name	Locations
Name	Columns 1–14
Ethnic	Column 16
DOB	Columns 18–27
Income	Columns 30–39

For example, data in CH8_2.TXT are in fixed field with the last two fields containing nonstandard numeric values (description of the data is presented in Table 8.5). Program 8.2 reads CH8_2.TXT by using the formatted input method. The PAD option is also used because the length of each record is not the same. In the INPUT statement, the @1pointer-control sets the input pointer at column 1. Because the default position for the input pointer is at column 1, @1 can be omitted. The informat $14. instructs SAS to copy the value from the input buffer to the variable NAME in the PDV. Next, the input pointer is located at column 15. The +1 pointer-control moves the input pointer to column 16. The $1. informat then creates the ETHNIC variable. The pointer-control @18 positions the input pointer to column 18, and the *MMDDYY10.* informat instructs SAS to read values for the DOB variable. The last pointer-control +2, along with the *COMMA10.* informat, reads the value for the INCOME variable:

CH8_2.txt:

```
1234567890123456789012345678901234567890
James Miller   W 03-02-1986  $1,100,000
Patricia Davis B 11-13-1982  $90,000
John Jackson   W 06-15-1981  $75,000
Mary Martinez  H 08-02-1980  $150,000
```

Program 8.2:

```
data ex8_2;
    infile "W:\SAS_Book\dat\ch8_2.txt" pad;
    input @1 Name $14.
          +1 Ethnic $1.
          @18 DOB mmddyy10.
          +2 Income comma10.;
run;

title 'Using the formatted input method and the PAD option';
proc print data = ex8_2;
run;

proc contents data = ex8_2 varnum;
run;
```

Output from Program 8.2:

```
Using the formatted input method and the PAD option
  Obs        Name        Ethnic    DOB     Income
   1    James Miller        W      9557   1100000
   2    Patricia Davis      B      8352     90000
   3    John Jackson        W      7836     75000
   4    Mary Martinez       H      7519    150000
```

Partial output from PROC CONTENTS:

```
Variables in Creation Order
 #    Variable   Type   Len
 1    Name       Char    14
 2    Ethnic     Char     1
 3    DOB        Num      8
 4    Income     Num      8
```

Program 8.2 also contains a CONTENTS procedure. Based on the partial output from PROC CONTENTS, the lengths for the NAME and ETHNIC variables are 14 and 1, respectively, which have the same *w* values in the character informats ($14. and $1.). However, both lengths for the DOB and INCOME variables are 8, which is different from the *w* values in the *MMDDYY10.* and *COMMA10.* informats. When using a formatted input method, the informat for the character variable is used to specify the number of columns to read in the input buffer; it is also used to specify the length of the variable. However, this rule does not apply to numeric variables. By default, SAS stores numeric values as floating-point numbers in 8 bytes, which is the default length for numeric variables. The *w* value in the informat for the numeric value does not affect the default length of the numeric variable but only serves to specify the number of columns to read.

8.2.3 List Input

The list input method is used to read free format data. Here is the syntax of the list input method:

```
INPUT variable <$>;
```

The optional dollar sign ($) in the INPUT statement is used to read character variables. To use the list input method, data in all the fields must contain either character values without any embedded blanks or standard numeric values. Each field needs to be separated by at least one blank or some other type of delimiter. The INPUT statement reads all the fields in each record from left to right, and you cannot skip or re-read any fields from the input data.

CH8_3.TXT contains data in free format with each field separated by a blank. The first and second fields contain first and last name, followed by age, number of people in the household, and the number of children in the household. Program 8.3 reads the CH8_3.TXT file by using the list input method.

CH8_3.txt:

```
123456789012345678901234567890
James Miller   26 3 1
Patricia Davis 30 4 1
John Jackson   31 2 0
Mary Martinez  32 4 2
```

Program 8.3:

```
data ex8_3;
    infile "W:\SAS_Book\dat\ch8_3.txt";
    input fname $
          lname $
          age
          family
          children;
run;

title 'Using the list input method';
proc print data = ex8_3;
run;
```

Output from Program 8.3:

```
                 Using the list input method
      Obs    fname       lname      age   family  children
       1     James       Miller      26     3        1
       2     Patricia    Davis       30     4        1
       3     John        Jackson     31     2        0
       4     Mary        Martinez    32     4        2
```

The data execution for reading data by using the list input is different from the column input method. Instead of searching for specific columns, SAS scans the input buffer from left to right to copy the values from the input buffer to the PDV. When the INPUT statement begins to execute, the input pointer is located at column 1 and then reads the first field until it encounters a blank space. The blank space is used to indicate the end of the field. Once the first field is copied from the input buffer to the PDV, SAS starts to scan the input buffer until the next nonblank column is found, and then the second value is read from the input buffer until another blank column is reached. This process continues until all the fields are read from the input buffer.

By default, all the character variables are assigned with default length of 8. Thus, character values longer than 8 will be truncated. To correctly read character values longer than 8, you can use the LENGTH statement to preset the variable length before reading the data.

The list input method can read missing values correctly if the missing value is represented as a period. If the missing value is represented as blanks and is located at the end of the record, you can use the MISSOVER option in the INFILE statement to read the missing value correctly.

The data in CH8_4.TXT are in free format. Notice that the fourth record has "Christopher" (11 characters) as the first name, and the last two fields of the fourth record are missing. Program 8.4 reads the data correctly by using the MISSOVER option and sets the length for FNAME to 11 by using the LENGTH statement.

CH8_4.txt:

```
123456789012345678901234567890
James Miller      26 3 1
Patricia Davis    30 4 1
John Jackson      31 2 0
Christopher Jones 25
Mary Martinez     32 4 2
```

Program 8.4:

```
data ex8_4;
    infile "W:\SAS_Book\dat\ch8_4.txt" missover;
    length fname $11.;
    input fname $
          lname $
          age
          family
          children;
run;

title 'Using the LENGTH statement and the MISSOVER option';
proc print data = ex8_4;
run;
```

Output from Program 8.4:

```
      Using the LENGTH statement and the MISSOVER option
     Obs   fname         lname      age   family   children
     1     James         Miller      26      3         1
     2     Patricia      Davis       30      4         1
     3     John          Jackson     31      2         0
     4     Christopher   Jones       25      .         .
     5     Mary          Martinez    32      4         2
```

When data are arranged in free format, each field can be separated by a delimiter. In this situation, you need to use the DLM= option in the INFILE

TABLE 8.6

Use of INFILE Options for Listing Input when the Text File Contains Delimiter(s) and Missing Values (Missing Values Are Represented as Blanks)

Delimiter	Missing Value Location	INFILE Statement Option
Nonblank	None	`DLM=`
blank(s)	End of the record	`MISSOVER`
comma (,)	End of the record	`DSD`
Other[a]	End of the record	`DSD and DLM=`
comma (,)	Beginning of the record	`DSD`
Other[a]	Beginning of the record	`DSD and DLM=`
comma (,)	Middle of the record	`DSD`
Other[a]	Middle of the record	`DSD and DLM=`

[a] Delimiters other than blanks and commas.

statement when reading the data. For example, suppose that commas (,) are used as delimiters to separate each file; you can write the following statement:

```
infile "W:\SAS_Book\dat\ch8_4.txt" DLM = ",";
```

If missing values are represented as blanks and exist in either the beginning or the middle of a record in a free-format data set, each field in the text file needs to be separated by a nonblank delimiter in order to be read into SAS correctly. Table 8.6 summarizes the use of options in the INFILE statement when text files contain delimiter(s) and missing values (represented as blanks).

When using the DSD (Delimiter-Sensitive-Data) option in the INFILE statement, SAS sets the delimiter to a comma, treats two consecutive delimiters as a missing value, and removes quotation marks from values. When the missing value exists at the beginning of the record, only one delimiter is needed.

CH8_5.TXT is similar to CH8_3.TXT except that a field for the new GROUP variable has been added to the beginning of each record. Each field in CH8_5.TXT is also separated by commas. Notice that there is a missing value at the beginning of the first record, the end of the second record, and the middle of the third record. In this situation, the DSD option added to the INFILE statement is used to read in the data. DSD works correctly without the DELIMITER = option because DSD expects a comma by default. (See Program 8.5.)

CH8_5.txt:

```
123456789012345678901234567890
,James,Miller,26,3,1
B,Patricia,Davis,30,4,
B,John,Jackson,,2,0
A,Mary,Martinez,32,4,2
```

Program 8.5:

```
data ex8_5;
    infile "W:\SAS_Book\dat\ch8_5.txt" dsd;
    input group $
          fname $
          lname $
          age
          family
          children;
run;

title 'Using the DSD option';
proc print data = ex8_5;
run;
```

Output from Program 8.5:

```
                      Using the DSD option
   Obs   group    fname       lname       age   family  children
    1             James       Miller       26      3         1
    2      B      Patricia    Davis        30      4         .
    3      B      John        Jackson      .       2         0
    4      A      Mary        Martinez     32      4         2
```

8.2.4 Modified List Input

The modified list input is a more flexible method that can read data in free format and can read character values containing embedded blanks and nonstandard numeric values. The general syntax of modified list input is as follows:

```
INPUT variable <:|&|~> <informat>;
```

You can use the colon (:) format modifier along with an informat to read nonstandard numeric values and character values with more than eight characters. However, the character values cannot contain any embedded blanks. When using the colon (:) format modifier, SAS reads values until it reaches a blank column, a defined length of a variable (character only), or the end of the data line, whichever comes first.

The ampersand (&) format modifier enables you to read character values with embedded blanks with or without specifying a character informat. If the character values have more than 8 bytes, you can either use the LENGTH statement to prespecify the length or insert the $*w.* informat after the ampersand (&).

The ampersand (&) format modifier works for values that have one or more nonconsecutive blanks, because processing continues to the next field only when two or more *consecutive* blanks are encountered in the text file.

The tilde (~) format modifier enables you to read and retain single or double quotation marks, as well as delimiters within character values. When using the tilde (~) format modifier, you must use the DSD option in the INFILE statement; otherwise, this option will be ignored by the INPUT statement. Please refer to SAS documentation for an example.

CH8_6.TXT contains data in free format. The first field contains values for first name, followed by last name, city, and income information. In this text file, the first name of the fourth record, "Christopher", has 11 characters. Some values in the third field (city) contain embedded blanks. The last field (income) contains nonstandard numeric values. Thus, a modified input method needs to be used to read CH8_6.TXT.

In Program 8.6, the colon (:) format modifier and the informat $11. are used to read the value for the FNAME variable. Because all values for the LNAME variables are less than or equal to 8, only the regular list input method is used for reading LNAME. Next, because the city field contains embedded blanks and some city values are longer than eight characters, the ampersand (&) format modifier and the $11. informat are used to read values for the CITY variable. The ampersand (&) format modifier tells SAS that the value for CITY should be read until two consecutive blanks are reached. In the end, the colon (:) format modifier and the *COMMAw.d* informat are used to read nonstandard numeric values for the INCOME variable.

The use of an informat in modified list input is different from that for formatted input. In a formatted input, the informat is used to determine the length of the variable and the number of columns to read in the text file. However, the informat in a modified list input is used only to determine the length of the variable. The list input reads values in each field until the next blank is encountered. In Program 8.6, the *w* value is not specified in the *COMMAw.d* informat because the default length for numeric variables is eight, and specifying *w* in the *COMMAw.d* informat is not necessary. However, you need to specify the *w* value in the *COMMAw.d* informat in the formatted input because the *w* value is used to determine the number of columns to be read.

CH8_6.txt:

```
1234567890123456789012345678901234567890
James Miller New York           $1,100,000
Patricia Davis Phoenix          $90,000
John Jackson Dallas             $75,000
Christopher Jones San Francisco $95,000
Mary Martinez Los Angeles       $150,000
```

Program 8.6:

```
data ex8_6;
    infile "W:\SAS_Book\dat\ch8_6.txt";
    input fname : $11.
```

```
          lname $
          city & $13.
          income : comma.;
run;

title 'Using the modified list input method';
proc print data = ex8_6;
run;
```

Output from Program 8.6:

```
             Using the modified list input method
    Obs   fname         lname      city              income
     1    James         Miller     New York         1100000
     2    Patricia      Davis      Phoenix            90000
     3    John          Jackson    Dallas              7500
     4    Christopher   Jones      San Francisco      95000
     5    Mary          Martinez   Los Angeles       150000
```

8.2.5 Mixed Input

You can mix input styles when reading a text file. For example, in CH8_7.TXT, the first field, occupying columns 1 to 14, contains values for the NAME variable. The second field, occupying column 16, contains values for the ETHNIC variable. The last two fields, INCOME (nonstandard numeric value) and CITY (containing embedded blanks), are in free format. Program 8.7 uses the column input method to read the first two fields, NAME and ETHNIC, and then reads INCOME and CITY by using the modified list input method.

CH8_7.txt:

```
1234567890123456789012345678901234567890
James Miller   W $1,100,000 New York
Patricia Davis B $90,000    Phoenix
John Jackson   W $75,000    Dallas
Mary Martinez  H $150,000   Los Angeles
```

Program 8.7:

```
data ex8_7;
    infile "W:\SAS_Book\dat\ch8_7.txt";
    input name   $ 1 - 14
       ethnic    $ 16
       income    : comma.
       city &    $13.;
run;

title 'Using the mixed input method';
proc print data = ex8_7;
run;
```

Output from Program 8.7:

```
                 Using the mixed input method
    Obs          name         ethnic     Income   city
     1     James Miller          W      1100000   New York
     2     Patricia Davis        B        90000   Phoenix
     3     John Jackson          W        75000   Dallas
     4     Mary Martinez         H       150000   Los Angeles
```

8.2.6 Creating Observations by Using the Line Pointer-Controls

In some situations, you may need to create one observation that is based on multiple records in a text file. You can move the input pointer of a specific record by using the following line pointer-controls:

```
/(forward slash)
#n
```

The forward slash (/) pointer-control moves the input pointer to column 1 of the next record. The #*n* pointer-control moves the input pointer to the record *n*.

Using the forward slash (/) pointer-control enables you to read multiple records sequentially or in any order. To use the line pointer-control, however, you must check that the text file contains the same number of records for each observation. The line pointer-control can be used with any input method.

For example, in the CH8_8 data set, the information required to create one observation is stored in three records. Program 8.8 uses the forward slash (/) pointer-control to read the data.

CH8_8.txt:

```
1234567890123456789012345
James Miller
W New York
$1,100,000
Patricia Davis
B Phoenix
$90,000
```

Program 8.8:

```
data ex8_8;
    infile "W:\SAS_Book\dat\ch8_8.txt";
    input fname : $11. lname $/
          ethnic $ city & $/
          income : comma.;
run;

title 'Using forward slash (/) pointer-control';
proc print data = ex8_8;
run;
```

Output from Program 8.8:

```
          Using forward slash (/) pointer-control
   Obs    fname      lname    ethnic     city      income
    1     James      Miller     W      New York   1100000
    2     Patricia   Davis      B      Phoenix      90000
```

Program 8.9 shows how the #*n* pointer-control is applied. First, the second record is read, followed by the first and finally by the last.

Program 8.9:

```
data ex8_9;
    infile "W:\SAS_Book\dat\ch8_8.txt";
    input #2 ethnic $ city & $
          #1 fname : $11. lname $
          #3 income : comma.;
run;

title 'Using #n pointer-control';
proc print data = ex8_9;
run;
```

Output from Program 8.9:

```
                 Using #n pointer-control
     Obs   ethnic     city     fname       lname     income
      1      W      New York   James       Miller   1100000
      2      B      Phoenix    Patricia    Davis      90000
```

8.2.7 Creating Observations by Using Line-Hold Specifiers

The INPUT statement has optional line-hold specifiers that provide greater flexibility for reading a text file into a SAS data set. Line-hold specifiers have either a single trailing "at" sign (@) or double trailing "at" signs (@@). Trailing "at" signs are always placed at the end of an INPUT statement:

```
INPUT <specification(s)><@|@@>;
```

Double trailing "at" signs (@@) are frequently used for creating multiple observations from a single record. With double trailing "at" signs (@@) you can hold data in the input buffer across multiple iterations of the implicit loop of the DATA step. The data record held in the input buffer by double trailing "at" signs (@@) is not released until either the input pointer moves past the end-of-record marker or another INPUT statement without a line-hold specifier executes. You cannot use double trailing "at" signs (@@) with the column input method, @ column pointer-control, or the MISSOVER option in the INFILE statement.

For example, CH8_9.TXT contains two records with each record containing first and last names plus age for two persons. Program 8.10 creates two observations in the SAS data set from each record in the text file by using double trailing "at" signs (@@) in the INPUT statement. During the first iteration of the implicit loop of the DATA step, the INPUT statement copies the first record from CH8_9.TXT to the input buffer. Once the record is copied to the input buffer, the INPUT statement copies the first group of values for FNAME, LNAME, and AGE from the input buffer to the PDV. Then the input pointer is positioned at column 16. The first observation is created at the end of the DATA step. At the beginning of the second iteration of the implicit loop, the double trailing "at" signs (@@) prevent the data value in the input buffer from being released and keep the input pointer at the current location (at column 16). Next, the INPUT statement copies the second group of values in the input buffer to the PDV and the input pointer reaches the *end-of-record* maker. When this happens, the first data record is released from the input buffer. At the beginning of the third iteration of the DATA step, the INPUT statement copies the second record from CH8_9.TXT to the input buffer. The input pointer moves to column 1 of the input buffer and the execution process is repeated until all the values in the second record are read.

CH8_9.txt:

```
123456789012345678901234567890123456789
James Miller 26 Patricia Davis 30
John Jackson 31 Mary Martinez  32
```

Program 8.10:

```
data ex8_10;
    infile "W:\SAS_Book\dat\ch8_9.txt";
    input fname $ lname $ age @@;
run;

title 'Using the double at sign(@@)';
proc print data = ex8_10;
run;
```

Output from Program 8.10:

```
             Using the double at sign(@@)
            Obs   fname      lname      age
            1     James      Miller     26
            2     Patricia   Davis      30
            3     John       Jackson    31
            4     Mary       Martinez   32
```

Similar to the double trailing "at" signs (@@), the single trailing "at" sign (@) enables multiple INPUT statements to be applied to a single record in the input buffer. However, the essential difference between the single and the double trailing “at” signs is that the single trailing “at” sign (@) also releases a record in the input buffer at the beginning of every DATA step execution, while double trailing “at” signs (@@) can hold a record in the input buffer across multiple iterations of the implicit loop of the DATA step until the end-of-record marker is encountered.

Suppose that you would like to create a SAS data set from CH8_1.TXT but only keep records of the people with AGE > 30. You can use the single trailing “at” sign (@) to accomplish this task. (See Program 8.11.) During the DATA step execution, the first INPUT statement reads the data value from columns 17 and 18 from the input buffer and assigns the AGE variable in the PDV. The single trailing “at” sign (@) holds the records in the input buffer. The subsetting IF statement evaluates the condition (AGE > 30). If the condition is not true, then the current DATA step iteration terminates, and SAS returns to the beginning of the DATA step for the next iteration of the implicit loop while at the same time releasing the contents of the input buffer. In the second iteration, the next record in the input buffer is read in with the first INPUT statement. Again, AGE > 30 is checked, and if TRUE, the second INPUT statement executes with values being copied from the input buffer to the NAME, GENDER, and CITY variables. Because the second INPUT statement doesn’t contain a line-hold specifier, the records in the input buffer will then be released after the INPUT statement executes. This process continues until all the records have been processed.

Program 8.11:

```
data ex8_11;
    infile "W:\SAS_Book\dat\ch8_1.txt" truncover;
    input age 17 - 18 @;
    if age > 30;
    input Name $ 1 - 14
          Gender $ 16
          City $ 20 - 30;
run;

title 'Using the single at sign(@)';
proc print data = ex8_11;
run;
```

Output from Program 8.11:

```
                 Using the single at sign(@)
         Obs   age         Name        Gender   City
           1    31   John Jackson        M      Dallas
           2    32   Mary Martinez       F      Los Angeles
```

8.3 Creating Text Files

The idea behind choosing an optimal output method is similar to choosing an input method. For example, to generate a text field in a fixed field, you can use the column output or formatted output methods. Using the formatted output method enables you to associate a format with a variable when you create your output file. To create a free format text file, you can use the list output method. In addition, you can mix different output methods to create a text file. Due to these similarities, generating text files in each of the following sections will not be presented in detail.

8.3.1 Column Output

The column output method is used to write variable values in the specified columns in the output line. The general form of the column output method is as follows:

```
PUT variable start-column <- end-column>;
```

The *variable* is the name of the SAS variable whose value is written to the text file. The *start-column* and the *last-column* are used to specify the first column and the last column of the field where the value is to be written in the text file. The *last-column* can be omitted if the value only copies one column. If the number of columns in a specific value is less than specified, a character variable will be left-aligned at *start-column*, and a numeric variable will be right-aligned at *end-column*. There is no need to write a dollar sign ($) after the character variable in the PUT statement.

In the previous section, Program 8.1 was used to create a SAS data set, EX8_1, with four variables by reading in CH8_1.TXT. The four variables were NAME, GENDER, AGE, and CITY. Program 8.12 shows how to create a text file from EXE8_1 with the column output method that contains fields for only the NAME, AGE, and CITY variables. GENDER is being excluded.

In Program 8.12, a DATA _NULL statement is used because the purpose of the DATA step is to create an external text file, not a SAS data set. The SET statement copies the contents from EX8_1 to the PDV. Next, the FILE statement specifies the name of the output file. Then, the PUT statement writes the values from variables NAME, AGE, and CITY to the specified columns in the text file. The generated file, EX8_1_OUT.TXT, is listed after Program 8.12.

Program 8.12:

```
data _null_;
    set ex8_1;
    file "W:\SAS_Book\dat\OUT\ex8_1_out.txt";
    put name 1 - 14
```

```
        age 16 - 17
        city 19 - 29;
run;
```

EX8_1_out.txt:

```
123456789012345678901234567890
James Miller   26 New York
Patricia Davis 30 Phoenix
John Jackson   31 Dallas
Mary Martinez  32 Los Angeles
```

8.3.2 Formatted Output

You can use the formatted output method to write formatted variables out to a text file. The general form of the formatted output method is as follows:

```
PUT <pointer-control> variable format;
```

Similar to the formatted input method, two commonly used column pointer-controls are @*n* and +*n*. The @*n* control moves the pointer to column *n*, and +*n* moves the pointer *n* columns forward. You can use the following alignment parameters in the *format* to override the default alignment:

- -L: left-align the value
- -C: center the value
- -R: right-align the value

Program 8.2 creates a SAS data set, EX8_2, by reading in CH8_2.TXT. Program 8.13 re-creates a text file by using the formatted output method. The alignment parameter -L is used to left-align the value of INCOME; otherwise, numeric variables are right-aligned by default.

Program 8.13:

```
data _null_;
    set ex8_2;
    file "W:\SAS_Book\dat\OUT\ex8_2_out.txt";
    put @1 name $14.
        +1 dob mmddyyc8.
        +2 income dollar13.2-l;
run;
```

EX8_2_out.txt:

```
1234567890123456789012345678901234567890
James Miller   03:02:86 $1,100,000.00
Patricia Davis 11:13:82 $90,000.00
John Jackson   06:15:81 $75,000.00
Mary Martinez  08:02:80 $150,000.00
```

8.3.3 List Output

You can use the list output method to write output to a text file. To use the list output method, simply list the variable(s) after the keyword PUT:

```
PUT variable;
```

When the list output method is used, a single-space character is inserted between each variable in the list that follows the PUT statement. Check the output from Program 8.14 for embedded spaces between NAME, AGE, and CITY.

Program 8.14:

```
data _null_;
    set ex8_1;
    file "W:\SAS_Book\dat\OUT\ex8_3_out.txt";
    put name age city;
run;
```

EX8_3_out.txt:

```
1234567890123456789012345678901234567890
James Miller   26 New York
Patricia Davis 30 Phoenix
John Jackson   31 Dallas
Mary Martinez  32 Los Angeles
```

Exercises

Exercise 8.1. Read DAT8_1.TXT into SAS. Descriptions of each field are listed in Table 8.7. The last field of the data contains quotation marks.

TABLE 8.7

Field Description for DAT8_1.TXT

Variable Name	Locations	Variable Type
ID	Columns 1–3	Character
Gender	Column 4	Character
Ethnic	Column 6	Character
City	–	Character
Income	–	Numeric
Smoking	–	Character

Please make sure to remove the quotation marks when you read the data.

DAT8_1.TXT

```
12345678901234567890123456789012345678901234567890123456789
629F H Los Angeles    $35,000.00 "past"
656F W New York       $48,000.00 "never"
711F W Chicago        $30,000.00 "never"
733F W Los Angeles    $59,000.00 "current"
135F B San Francisco $120,000.00 "current"
982F W Boston        $113,000.00 "past"
798F W Chicago        $28,900.00 "never"
494F W Seattle        $65,000.00 "never"
748F W Los Angeles    $39,000.00 "never"
904F W Seattle        $76,200.00 "never"
244F W Orlando        $58,000.00 "never"
747F A New York       $39,000.00 "current"
796F A San Francisco $134,000.00 "past"
713F H Chicago        $29,000.00 "never"
745F A Seattle        $76,000.00 "never"
184M W Boston         $13,900.00 "past"
```

Exercise 8.2. In DAT8_2.TXT, the first two fields contain ID (columns 1–3) and GENDER (column 4), and the last four fields contain quarterly income. Create a SAS data set by reading DAT8_2.TXT. There will be 12 observations in the generated SAS data set and it will contain the following variables: ID, GENDER, QTR (with values 1, 2, 3, or 4), and INCOME (contains monthly income). (Hint: Use a single trailing "at" sign (@) to hold the input pointer.)

DAT8_2.TXT

```
12345678901234567890123456789012345678901234567890123456789
629F $5,000.00 $5,500.00 $4,000.00 $3,500.00
656F $3,400.00 $5,600.00 $7,800.00 $3,100.00
711F $2,800.00 $3,100.00 $2,900.00 $3,800.00
```

The created SAS data set will be similar to the one below:

```
        The SAS data set that created from DAT8_2.TXT
           Obs       id      gender      qrt     income
            1       629        F          1       5000
            2       629        F          2       5500
            3       629        F          3       4000
            4       629        F          4       3500
            5       656        F          1       3400
```

```
6       656       F         2       5600
7       656       F         3       7800
8       656       F         4       3100
9       711       F         1       2800
10      711       F         2       3100
11      711       F         3       2900
12      711       F         4       3800
```