

# Reading and Writing Data in SAS

- 1) Basic input/output (i/o) commands:
  - in the introduction, we went over basic i/o commands
  - basic input:
    - to read a SAS data set, you would use the SET command
    - to read a space-delimited ASCII data set, you would use the INFILE and simple (variable list only) INPUT commands
  - basic output:
    - to create a SAS data set, you would use the DATA statement
    - to create a space-delimited ASCII data set, you would use the OUTPUT and simple PUT commands
  - this class will cover more advanced operations, including
    - creating and reading *permanent* SAS data sets
    - options for creating and reading SAS data sets
    - different types of formatted ASCII data sets
    - working with SAS transport files
- 2) What does a DATA step *normally* do?<sup>1</sup>
  - Compilation Phase
    - checks syntax of submitted statements
    - creates input buffer, program data vector, and descriptor information
  - Execution Phase—implicit loop over observations being read; observations processed
    - starting from DATA statement
    - a record is either read into the input buffer (if INPUT is used) or directly into the program data vector if a SAS data set is read (through a SET or MERGE)
    - additional statements in the DATA step are executed on *that observation*
    - at end of DATA step,
      - ◊ observation is output
      - ◊ processing returns to the top of that step
      - ◊ all information in the program data vector is reset
    - iteration counter (\_N\_) is updated
    - flow continues until the end of the input file is reached
  - processing is best-suited for “rectangular” data sets but can accommodate other structures
  - lots of variations possible
- 3) the OUTPUT and DELETE commands
  - unless told otherwise, SAS outputs one record to the SAS data set specified in the DATA statement at the end of the DATA step for each record read
    - if you do not want SAS to output a record to the SAS data set, you would type the DELETE command

---

<sup>1</sup> See <http://support.sas.com/documentation/cdl/en/lrcon/59522/HTML/default/a000985872.htm>.

- ◇ when this command is used, the current observation is not output
- ◇ also, SAS immediately returns to the top of the DATA step to begin a new iteration (the commands following a DELETE statement are not executed)
- ◇ the DELETE command shortens the length of a SAS data set (recall that the DROP and KEEP commands reduced the “width” of a data set)
- if you want to write an observation to a SAS data set somewhere other than the end of a DATA step, you would use the OUTPUT command

#### 4) Temporary and permanent SAS data sets

- SAS distinguishes between temporary and permanent SAS data sets
- temporary SAS data sets
  - only last as long as a SAS session; they are automatically deleted when the SAS session closes
  - have a one-part name, e.g., <temp\_data\_set\_nm>
  - to read a temporary data set, you would use the command:

```
SET <temp_data_set_nm>;
```

- to create a temporary data set, you would begin the DATA step with the statement:

```
DATA <temp_data_set_nm>;
```

- permanent SAS data sets
  - are stored in *libraries* and can “outlive” a SAS session
  - have a two-part name, <lib\_ref>.<perm\_data\_set\_nm>, where
    - ◇ the first part, <lib\_ref>, is the internal library name (defined as the first argument in a LIBNAME statement), and
    - ◇ the second part, <perm\_data\_set\_nm>, is the permanent data set name
  - in Windows, the permanent data set would be stored as <perm\_data\_set\_nm>.sas7bdat in the directory defined in the second argument of the LIBNAME statement
  - to read a permanent data set, you would use the command:

```
SET <lib_ref>.<perm_data_set_nm>;
```

- to create a permanent data set, you would begin the DATA step with the statement:

```
DATA <lib_ref>.<perm_data_set_nm>;
```

#### 5) SAS data set options<sup>2</sup>

- options can be used when a SAS data set is created or read
- most options can be used for either operation, but some are specific to creating or reading
- the options would appear in parentheses after the SAS data set name

---

<sup>2</sup> See <http://support.sas.com/documentation/cdl/en/lrdict/59540/HTML/default/a002295655.htm>.

- some common options are:

Option	Description	Input/Output
DROP =	drops listed variables from data set	both
FIRSTOBS =	data processing begins with specified observation number	input
KEEP =	keeps only the listed variables	both
RENAME =	renames the listed variables	both
WHERE =	only keeps observations that satisfy the listed condition	input

- some examples
  - SET <sas\_data> (DROP = <var1> <var2>);  
drops <var1> and <var2> from <sas\_data>
  - SET <sas\_data> (KEEP = <var1> <var2>);  
only inputs <var1> and <var2> from <sas\_data>
  - SET <sas\_data> (RENAME = (<var1>=<new1>));  
renames input variable <var1> as <new1>

## 6) changing general specifications of input and output files

- we have previously discussed the syntax for input and output statements for simple space-delimited text/ASCII files
- changing the length of the line – one way that text input and output can become more complicated has to do with the length of the line being read
  - the Windows version of SAS “assumes” that input and output will take up no more than 256 character spaces on each line
  - unless it is told otherwise, SAS will ignore input past the 256<sup>th</sup> character position of a line and jump to the next line to continue reading a record
  - similarly, when writing output, SAS will begin a new line (insert a line break) after the 256<sup>th</sup> position
  - to specify an alternative *logical record length*, or LRECL, you would add the LRECL= option to the end of either the INFILE or FILE statement
- changing delimiters – another potential complication is the use of alternative delimiters in a file, such as commas
  - to change the delimiters that SAS will recognize, add the DELIMITER= or DLM= option
  - the new delimiter or delimiters should appear as a character constant (e.g., DLM=',')
  - or as a character variable
- reading tabbed data – some datasets include tabs as delimiters

- in a file, tabs are simply stored as characters, so you could adjust the delimiter to recognize tabs
- another approach is to include the EXPANDTABS option, which converts tab characters to a series of spaces (converts tab-delimited input into space-delimited input)

## 7) formatted input and output files

- the syntax that we discussed for INPUT and PUT statements was very simplified; a more complete syntax is:

```
INPUT <specification_1> <specification_2> ... <@ | @@>;
```

```
PUT <specification_1> <specification_2> ... <@ | @@>;
```

- each “specification” can consist of several parts:
  - ◇ column and line pointer-controls
  - ◇ a variable name or list
  - ◇ column specifications
  - ◇ format modifiers and informats
- in addition @ and @@ characters can appear at the end of the statements to hold the line until the next INPUT or PUT statement in the same DATA step iteration (@) or possibly across iterations (@@)
- pointer controls
  - column pointer controls precede variable names in an INPUT or PUT statement; these controls direct SAS to begin reading or writing the variable information starting at the specified column
  - column point controls can be absolute or relative
    - ◇ absolute controls are written @*n*, where *n* is the column position where you want to next read or write
    - ◇ relative controls are written +*n*, where *n* is the number of additional column positions from the current position
  - examples:

```
PUT @12 variable1;
```

this would begin writing *variable1* in the 12<sup>th</sup> column position of an output line

```
INPUT @4 variable1 4. +5 variable2;
```

this would read *variable1* from the 4<sup>th</sup> – 7<sup>th</sup> column positions and begin reading *variable2* in the 13<sup>th</sup> column position of the input line

- line pointers advance the input or output by a line; the symbol for a line advance is /

```
INPUT variable1 / variable2;
```

- would read *variable1* from one line and *variable2* from the next line
- line holds – normally at the end of an INPUT or PUT command, SAS advances to the next line; line hold commands prevent this; these commands are useful if the formatting of data (the record lay-out) varies across lines; you can determine what type of record you are reading with one INPUT command and then specify the correct format for the rest of the record using a subsequent command
  - ◊ to hold a line within the current DATA step iteration, you would leave an @ at the end of the line; for example,

```
INPUT variable1 @;

IF variable1 EQ 1 THEN
  INPUT variable2;
ELSE
  INPUT variable3;
```

would read *variable1* from the beginning of a record line; depending on the value of *variable1*, the code would read and interpret the next piece of data on the same line as either *variable2* or *variable3*

- ◊ to hold a line across DATA step iterations, you would leave an @@ at the end of the line
- informat specifications
  - informat specifications immediate follow either a variable or a list of variables
  - to specify the input or output as character data, you would use \$ as the informat specification; to specify something as a character of a particular length, you would use \$. as the informat, where *n* gives the length of the character
  - to specify the input or output as an integer of a particular length, you would use *n*.
  - to specify the input or output as a real number of a particular length and precision, you would use *n.d*, where *n* is the total length of the number (including the decimal point) and *d* is the number of decimal places in the number

```
INPUT variable1 $8. variable2 5. variable3 8.3;
```

- would read *variable1* as a character variable of length 8, *variable2* as an integer of length 5, and *variable3* as a real variable of length 8 and with 3 decimal places
- SAS has MANY other informat specifications, see <http://support.sas.com/documentation/cdl/en/lrdict/59540/HTML/default/a000309877.htm> for a complete list
- for lists of variables with the same informat, you would group the variables and informat into parentheses; for example

```
INPUT (variable1 variable2 variable3) (5.);
```

- column specifications
  - another way to read or write data is to specify the beginning and ending column positions

- these specifications would follow the variable name in an INPUT or PUT command

```
PUT variable1 12-15 variable2 16-18;
```

writes *variable1* in the 12<sup>th</sup> – 15<sup>th</sup> column positions and *variable2* in the 16<sup>th</sup> – 18<sup>th</sup> positions

#### 8) SAS transport files

- the structure of SAS files varies across computer platforms; a SAS file created for a Windows computer will not be readable on a UNIX computer
- SAS does have a facility for creating files that can be moved across systems; these are called *transport* files
- transport files cannot be read directly in a DATA step; you need to first create a copy of the file in your system's format
- suppose that you had a SAS transport file called *newdata.xpt* in a directory *C:\mysasdata*, and suppose that you wanted to create a SAS permanent data based on this to the same directory; you could do this by typing the code

```
LIBNAME outlib "C:\mysasdata";  
LIBNAME trandata XPORT "C:\mysasdata\newdata.xpt";
```

```
PROC COPY IN=trandata OUT=outlib;  
RUN;
```

- transport files can also be read (and in some cases must be read) using the CIMPORT procedure; if the COPY procedure does not work, you might try the CIMPORT
- to perform the same steps using the CIMPORT procedure, you would type

```
LIBNAME outlib "C:\mysasdata";  
FILENAME tran_in "C:\mysasdata\newdata.xpt";
```

```
PROC CIMPORT LIBRARY=outlib INFILE=tran_in;  
RUN;
```

- note: transport files can be tricky to use, and some of the specifications for these files can vary across versions of SAS