**CHAPTER 8 Reading and Writing Text Files with SAS**

a. Introduction to using text files with SAS

1. Reading text files with the INPUT statement; also, the INPUT function
2. Writing text files with the PUT statement; also, the PUT function

Thus far, all the DATA step programs we have seen have involved reading and writing only SAS data sets. In this chapter we will present techniques to read and write external or "raw" files in the DATA step using the INPUT and PUT statements. Any combination of these inputs and outputs can be used in a DATA step.

raw data raw data

INFILE FILE

OR

PUT

INPUT

## DATALINES

INPUT DATA step OUTPUT

MERGE

SET

UPDATE

OUTPUT

DATA

SAS data set SAS data set

Reading External Files with SAS

## The following example creates a temporary SAS data set from the raw data file CLASS.ASC .

RAW DATA FILE CLASS.ASC

CHRISTIANSEN M 37 71 195

HOSKING J M 31 70 160

HELMS R M 41 74 195

PIGGY M F . 48 .

FROG K M 3 12 1

GONZO 14 25 45

FILENAME in 'E:\BIOS511\CLASS.ASC';

DATA class;

INFILE in;

INPUT Name $ 1-12 Sex $ 15 Age 18-19 Ht 22-23 Wt 26-28 ;

RUN;

PROC PRINT DATA=class;

RUN;

RAW DATA

DATA STEP

WORK.CLASS

Other file extensions commonly used for raw data files are DAT and TXT.

10 FILENAME in 'E:\BIOS511\CLASS.ASC';

11

12 DATA class;

13 INFILE in;

14 INPUT Name $ 1-12 Sex $ 15 Age 18-19 Ht 22-23 Wt 26-28 ;

15 RUN;

NOTE: The infile IN is:

File Name=E:\BIOS511\CLASS.ASC,

RECFM=V,LRECL=256

NOTE: 6 records were read from the infile IN.

The minimum record length was 28.

The maximum record length was 28.

NOTE: The data set WORK.CLASS has 6 observations and 5 variables.

NOTE: DATA statement used:

real time 0.11 seconds

cpu time 0.04 seconds

16

17 PROC PRINT DATA=class;

18 RUN;

NOTE: There were 6 observations read from the data set WORK.CLASS.

NOTE: PROCEDURE PRINT used:

real time 0.22 seconds

cpu time 0.05 seconds

The SAS System

Obs Name Sex Age Ht Wt

1 CHRISTIANSEN M 37 71 195

2 HOSKING J M 31 70 160

3 HELMS R M 41 74 195

4 PIGGY M F . 48 .

5 FROG K M 3 12 1

6 GONZO 14 25 45

**Compilation Phase**

During the compilation of the DATA step, the INPUT statement:

* Creates an input buffer to provide temporary storage for the current input record being processed.
* Creates the PDV containing the variables listed in the INPUT statement.
* Generates the machine language instructions for reading the data values from the input buffer and converting them into the corresponding SAS variable values in the PDV.

### Notes:

* The length of the input buffer is determined by the LRECL (logical record length) of the file referenced by the INFILE statement.
* INFILE and INPUT statements are used only to read non-SAS files. To read SAS data sets, use a SET, MERGE, or UPDATE statement.

#### Execution Phase

During the execution phase of the DATA step, the INPUT statement:

* Reads the next record from the input file into the input buffer.
* Reads each data value from the appropriate field in the input buffer, converts it to the specified SAS variable, and stores the result in the PDV.
* Notes:
* The PDV is initialized to missing each time control returns to the top of the DATA step.
* Once variable values have been read into the PDV, they can be modified with DATA step programming statements in the same manner as values read from an existing SAS data set.

**The DATALINES/CARDS Statement**

* The DATALINES and CARDS statements are interchangeable.
* Used to mark the start of a group of "raw" data lines to be read by SAS.
* Must be placed at the end of the DATA step (immediately following the RETURN statement).
* The DATALINES statement is followed by the data lines, and then a line containing only a semicolon.
* The general form is:

DATA new;

INPUT ................................ ;

\*

\*

\* (SAS statements)

\*

\*

OUTPUT ;

RETURN ;

DATALINES ; /\* or CARDS; \*/

\*

\* (data lines)

\*

;

**Example: The DATALINES Statement**

DATA new;

INPUT A B C $;

D = A + B;

OUTPUT;

RETURN;

DATALINES; /\* or CARDS; \*/

2 5 M

3 6 F

7 9 F

;

**The FILENAME Statement**

The FILENAME statement associates a *fileref* (a valid SAS name up to eight characters long) with an external file or device. An external file is a non-SAS file residing or to be created on a host operating system (such as Windows) which you need for a purpose such as the following:

* You want to read data from the file.
* The file contains SAS programming statements that you want to run in a particular way.
* You want to write output to the file.

Once you associate a fileref with an external file, the fileref can be used as a shorthand reference for that file in SAS programming statements (INFILE, FILE, and %INCLUDE) and windowing environment commands (FILE and INCLUDE) that access external files. You can associate a fileref with a single external file or an aggregate storage location (such as a directory) that contains many external files.

SYNTAX:

FILENAME fileref 'external file' <host options> ;

fileref is any valid SAS name (up to eight characters long)

'external file' is the physical name of an external file or an aggregate location such as a directory.

EXAMPLES:

Example 1: Associating a fileref with a single external file

This example reads data from a file that has been associated with the fileref BIOS.

FILENAME bios 'd:\bios511\class.asc' ;

DATA one ;

INFILE bios ;

INPUT x y z ;

RUN;

Example 2: Associating a fileref with an aggregate storage location

If you associate a fileref with an aggregate storage location, you then use the fileref, followed in parentheses by an individual filename, to read from or write to any of the individual external files stored there.

FILENAME bios 'd:\bios511' ;

DATA one ;

INFILE bios(class.asc) ;

INPUT x y z ;

RUN;

**The INFILE Statement**

The INFILE statement connects an external, non-SAS file with a DATA step in order to read the raw data records contained in the file with an INPUT statement.

Syntax:

INFILE *file-specification* <options>:

where file-specification identifies the source of input data records to be read. Many of the INFILE options are technical in nature and have to do with operating system features and file organizations that are beyond the scope of this course.

The *file-specification* can have the following forms:

'external-file' specifies the physical path and name of an external file. It must be enclosed in quotes.

fileref gives the fileref of an external file. The fileref must have been previously associated with an external file in a FILENAME statement.

Selected options of interest include:

-- END=*variable* *variable* is set to 1 when you are processing the last record

-- FIRSTOBS=*n*, OBS=*n* *n* on FIRSTOBS is first input record to be read, *n* on OBS is last input record to be read

-- LINESIZE=*ls*, N=*nl* set number of columns (*ls*) and lines (*nl*) available to the input buffer

-- COLUMN=*var*, LINE=*var* name special variables to hold the current column and line location of the pointer in the input buffer

-- MISSOVER, STOPOVER, to determine what to do if too few data values are found in an input record

TRUNCOVER

-- DLM=, DSD to control the delimiters for list input; default delimiter is a blank space but DSD makes comma the default; other common delimiters are ‘,’ (comma) and ‘09’x (tab)

-- PRINT to read a print file without having to remove the carriage control characters

Notes:

* The INFILE statement must be executed before the INPUT statement that reads the raw data lines.
* More than one INFILE statement may be used in a single DATA step. INPUT statements will read from the file defined by the most recently executed INFILE statement.

##### The INPUT Statement

* The INPUT statement reads raw data lines from an external (non-SAS) file specified in the INFILE statement and converts the values of the fields into the corresponding values of SAS variables in the Program Data Vector.
* Alternatively, it tells SAS to read one or more lines from the data provided in the DATALINES section.
* The INPUT statement works as follows:
* The contents of the line are copied into the computer's memory.
* SAS moves to a column indicated by INPUT and reads to the end of the data field.
* SAS converts the data to a SAS numeric or character variable and stores the SAS-formatted data in the PDV.
* The process is repeated for each variable indicated in the INPUT statement.

Notes:

* INPUT initiates the reading of data; INFILE or DATALINES just tells SAS where to look for the data.
* You do not have to read every field of raw data into your SAS data set.
* You can read the same field more than once.
* The general form of this statement is

INPUT list of variable names and specifications;

The variable names are new SAS variables being created by the INPUT statement. The specifications define the location of the input fields and how the values are to be converted to SAS's internal representation. The form of the specifications will vary depending on the input mode (our next topic).

#### Column Input Mode

The most straightforward and commonly used mode of input specification is column mode. It should be used when the data values are in the same fields on all the data lines and the data values are in standard numeric or character form.

Syntax:

INPUT var1 start-end var2 $ start-end . . ;

where:

var1 is a numeric variable starting in column "start" and ending in column "end", and

var2 $ is a character variable starting in column "start" and ending in column "end."

Column Input Example: Create a temporary SAS data set from raw data

DATA student;

INPUT Name $ 1-6 HW1 10-11 HW2 15-17;

DATALINES;

JOHN B 9 90

ALBERT 10 95

SALLY 8

MARY 7 74

;

PROC PRINT DATA=student;

ID name;

TITLE 'WORK.STUDENT';

RUN;

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

input │

buffer└‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑┴‑‑

Name HW1 HW2

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑‑‑‑‑‑‑‑

PDV │

└‑┴‑‑┴‑‑┴‑‑┴‑‑┴‑‑‑┴‑‑‑‑‑‑‑‑‑‑‑‑┴‑‑‑‑‑‑‑‑‑‑‑‑

WORK.STUDENT

Name HW1 HW2

JOHN B 9 90

ALBERT 10 95

SALLY . 8

MARY 7 74

**Notes on Column Input Mode**

* The field positions for the variables are fixed.
* A $ is used to indicate character variables.
* Data fields can be read and reread in any order.   
    
  INPUT HW1 10-11 NAME $ 1-6 HW2 15-17;

INPUT NAME $ 1-6 HW1 10-11 HW2 15-17 HW2CHAR $ 15-17;

* Blanks are read as missing.
* For character variables:
* Embedded blanks are allowed in fields ( “JOHN SMITH”).
* Data values are left-justified.
* The length of the variable is determined by the column specification.
* Character data values can range from 1-32,767 in length starting with Version 7.0 of SAS.
* For numeric variables:
* The value can be anywhere within the column specification.
* Embedded blanks are not allowed in a numeric field.

List Input Mode

In list input the variables are specified in the order in which they appear on the data lines. No field specification is given, but each character variable name is followed by a $. A pointer is used to determine which bytes of the input buffer correspond to each variable in the INPUT statement. The pointer scans the input buffer until it encounters a non-blank character. That character and all consecutive non-blank characters on the line comprise the data value. The next blank signals the end of that data field and the process continues.

Syntax:

INPUT var . . . var $ . . . ;

List Input Example:

DATA class;

INFILE student;

INPUT NAME $ SEX $ AGE HT WT;

OUTPUT;

RETURN;

RUN;

HOSKING M 31 70 160

HELMS R M 41 74 195

OS file CHRISTIANSEN M 37 71 185

PIGGY F . 48 .

FROG F 3 12 1

GONZO . 14 25 45

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

input buffer │

└‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

NAME SEX AGE HT WT

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑┬‑‑‑‑‑┬‑‑‑‑‑

PDV │

└‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑┴‑‑‑‑‑┴‑‑‑‑‑┴‑‑‑‑‑

**WORK.CLASS**

NAME SEX AGE HT WT

SAS data set HOSKING M 31 70 160

HELMS R . 41 74

CHRISTIA M 37 71 185

PIGGY F . 48 .

FROG F 3 12 1

GONZO 14 25 45

**Notes on List Input Mode:**

* List input is needed when data values are stored with one blank between each field instead of being stored in fixed fields.
* Data lines can be free format.
* A $ is used to indicate character variables.
* Every field must be specified in order.
* Data fields must be separated from each other by at least one blank.
* Character variables default to 8 bytes. If you want character variables to be longer than 8, include a LENGTH statement before the INPUT statement.
* Embedded blanks are not allowed in character variables.
* Missing values must be coded as "." (as a place holder) for numeric and character variables.
* Blank fields (missing values not coded as ".") cause the matching of variable names and values to get out of sync.

**Formatted Input Mode**

The most flexible (and complex) input mode is formatted input. This mode specifies explicit pointer controls and informats for each variable in the INPUT statement. Formatted input mode is absolutely needed if you are reading in non-standard numeric data (dates, numbers with commas).

An informat is a set of directions for converting the characters in an input field to a variable's value in one of SAS's two types of internal representation. The wide variety of informats available allows SAS to read data written in "virtually any form."

With formatted input you specify the starting location and field width by moving an input “pointer” to the starting position of the field and then specifying the variable name and an informat.

Syntax:

INPUT *pointer-control* variable *informat* . . . ;

where *pointer-control* directs the pointer to the specified byte of input buffer and

*informat* defines the input format and field width to be used to read the variable's value.

###### Pointer Controls

@n (absolute) go to byte n of input buffer

+n (relative) move pointer n bytes to the right

w (informat) format width specification moves pointer to the right as it reads the data value

###### Selected Informats

w. Reads standard numeric data

w.d Reads standard numeric data with decimal

COMMAw.d Removes embedded commas, dollar signs,

and other special characters from

incoming numeric data

$w. Reads standard character data

$CHARw. Reads character data with leading blanks

$UPCASEw. Converts character data to uppercase

DATEw. Reads date values in the form *ddmmmyy* or *ddmmmyyyy*

MMDDYYw. Reads date values in the form *mmddyy* or *mmddyyyy* or

*mm/dd/yy* or *mm/dd/yyyy*

**Formatted Input Example 1**

INPUT @1 NAME $6. +3 HW1 2. @l5 HW2 3.;

JOHN B 9 90 raw data ALBERT 10 95

SALLY 8

MARY 5 8

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

input buffer │

└‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

NAME HW1 HW2

┌‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑┬‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

│

│

└‑┴‑┴‑┴‑┴‑┴‑┴‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑┴‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

**WORK.STUDENT**

NAME HW1 HW2

JOHN B 9 90

ALBERT 10 95

SALLY . 8

MARY 5 8

**Formatted Input Example 2**

The postal codes and 1980 population for five states in the southeast are shown in the table below. Read the data and create a SAS data set.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | L |  | 9 | , | 7 | 3 | 9 | , | 9 | 9 | 2 |
| G | A |  | 5 | , | 4 | 6 | 4 | , | 2 | 6 | 5 |
| N | C |  | 5 | , | 8 | 7 | 4 | , | 4 | 2 | 9 |
| S | C |  | 3 | , | 1 | 1 | 9 | , | 2 | 0 | 8 |
| V | A |  | 5 | , | 3 | 4 | 6 | , | 2 | 7 | 9 |

FILENAME xyz ‘c:\bios511\sepop.asc’;

DATA pops;

INFILE xyz;

INPUT @1 state $2. +1 pop COMMA9.;

RUN;

**WORK.POPS**

Obs state pop

1 FL 9739992

2 GA 5464265

3 NC 5874429

4 SC 3119208

5 VA 5346279

**Formatted Input Example 3: Dates**

DATA one;

INPUT @1 Date1 MMDDYY6. @8 Date2 6. @15 Date3 DATE7. @15 Date4 $7. ;

DATALINES;

011593 011593 15JAN93

;

RUN;

TITLE 'Formatted Input / Dates Example';

PROC PRINT;

RUN;

PROC CONTENTS;

RUN;

Formatted Input / Dates Example

Obs Date1 Date2 Date3 Date4

1 12068 11593 12068 15JAN93

Formatted Input / Dates Example

The CONTENTS Procedure

-----Alphabetic List of Variables and Attributes-----

# Variable Type Len Pos

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

1 Date1 Num 8 0

2 Date2 Num 8 8

3 Date3 Num 8 16

4 Date4 Char 7 24

**Input Example Including an Error**

DATA one;

INPUT X Y;

DATALINES;

3 4

5 6

7 F

;

132 DATA one;

133

134 INPUT X Y;

135

136 DATALINES;

**NOTE: Invalid data for Y in line 139 3-3.**

RULE: ----+----1----+----2----+----3----+----4----+----5----+----6----+--

139 7 F

X=7 Y=. \_ERROR\_=1 \_N\_=3

NOTE: The data set WORK.ONE has 3 observations and 2 variables.

NOTE: DATA statement used:

real time 0.03 seconds

cpu time 0.03 seconds

140 ;

**Trailing @**

An INPUT statement ending with "@" (before the ";") holds the current data line in the input buffer for the next INPUT statement to read. This is extremely useful when reading

data files containing mixed record types. For example, consider records from two versions of a form in the same file.

DATA one;

INPUT Version 1 @;

IF Version EQ 1 THEN INPUT Name $ 2-9 Sex $ 11 Age 13-14;

IF Version EQ 2 THEN INPUT Name $ 2-9 Sex $ 10 Age 12-13 Ht 15-16;

DATALINES;

1HOSKING M 31

## 2HELMS M 40 70

1CHRISTIA M 37

2PIGGY MF 48

;

###### Trailing @ Example

###### The raw data file LIPA.ASC contains data with multiple record types. Read in data where form=LIP and create a SAS data set.

FILENAME raw 'C:\BIOS511\LIPA.ASC' ;

DATA one;

INFILE raw;

INPUT Form $ 1-3 @ ;

IF Form='LIP' THEN DO;

INPUT X 4-5 Y 6-7 Z 8-9 ;

OUTPUT;

END;

RUN;

**Double Trailing @**

An INPUT statement ending with "@@" (before the ";") also holds the current data line in the input buffer. However, this double trailing @ holds the current data line in place even when SAS starts building a new observation. This feature enables DATA steps such as the following.

DATA new;

INPUT x y @@;

DATALINES ;

57 8 55 10 31 10 17 8 90 16 18 3

;

Data Set New

Obs x y

1 57 8

2 55 10

3 31 10

4 17 8

5 90 16

6 18 3**Format Lists**

A variation of formatted input is called "format list mode," in which a list of variable names and a list of pointer controls and formats are grouped separately by parentheses:

INPUT (NAME SEX AGE HT WT)

($8. @11 $1 2. +1 4. @21 5.);

This is really only an advantage when there is a series of variables with a repeating format.

For example,

INPUT X1 8. X2 8. X3 8. X4 8. X5 8.;

can be rewritten as:

INPUT (X1-X5) (5\*8.);

The format list cycles to satisfy the variable list:

INPUT (X1-X5) (8.);

INPUT (X1 Y1 X2 Y2 X3 Y3) (2. 3.);

INPUT (X1-X2) (2. +3) @1 (Y1-Y3) (+2 3.);

In general, this mode of input is the most error prone; avoid it when possible.

#### Mixing Input Styles

You can mix the three INPUT styles (list, column, and formatted) in one INPUT statement.

INPUT name $ 1-11 sex $1. age 13-14 ht @21 wt 3.;

Ethel F52 65 145

Fred M55 70 170

**Named Input Mode**

Named input is a special input mode that allows the data lines to contain variable names as well as their values.

Syntax:

INPUT var1 . . . namedvar1= namedvar2= . . . ;

Data lines can then specify the values of the named variables in any order as long as named variables are after regular input fields. This mode is useful in constructing transaction data sets for subsequent use with the UPDATE statement.

###### Named Input Example

DATA trans;

INFILE fixes;

INPUT ID AGE= WT= HT= ;

RUN;

## 243 AGE=12 WT=99 HT=50

OS file 475 WT=120 AGE=11 HT=45

649 HT=60

**WORK.TRANS**

ID AGE WT HT

243 12 99 50

475 11 120 45

649 . . 60

**Multiple Records Per Observation**

There are several ways to read an observation that continues over multiple input records. Consider input data consisting of three lines per person.

* Use multiple INPUT statements to advance to the next input line. Each INPUT statement advances to the next record.

DATA class;

INFILE rawdata;

INPUT Name $ 1-8 Sex $ 11;

INPUT Age;

INPUT Ht 1-7 Wt 8-10;

* Use / to advance to the next input line. The / advances the pointer to the next line of raw data.

INPUT Name $ 1-8 Sex $11 / Age / Ht 1-7 Wt 8-10;

rawdata HOSKING M

31

70 160

┌‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

input │

buffer └‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑‑

Name Sex Age Ht Wt

PDV

**Multiple Line Input Buffers: The Line Pointer**

SAS can be instructed to set up a multiple-line input buffer. A line pointer is moved from line to line to control input processing. Use #n to advance to the first column of the nth record in the group (i.e., to set the line pointer to the nth line in the input buffer):

INPUT #1 NAME $ 1-8 SEX $ 11

#2 AGE 1-2

#3 HT 1-7 WT 8-10;

or another order:

INPUT #3 HT 1-7 WT 8-10

#1 NAME $ 1-8 SEX $11

#2 AGE 1-2;

The highest #n will be the number of lines in the input buffer. In this example, the input buffer has three lines:

HOSKING M

31

70 160

NAME SEX AGE HT WT

PDV

The INFILE statement option N= can also be used to declare the number of lines in the buffer.

**Summary: Pointer Controls**

column line

absolute @n #n

relative +n /

Notes:

* n can be a positive numeric constant or a numeric variable.
* For the relative column shift (+n), n can also be a numeric variable with a negative value.

## **The INPUT Function**

The INPUT function is similar to the INPUT statement in that it reads a string of characters and converts them to a SAS variable value according to a specified informat. It differs from the INPUT statement in that it "reads" the string of characters from a SAS character variable rather

than from an external file.

Syntax:

variable = INPUT (charactervariable, informat);

The length and type (character or numeric) of "variable" will be determined by "informat".

**INPUT Function Example**

DATA one;

INPUT @1 Var1 $2. @4 Date1 $6.;

NVar1 = INPUT(Var1,2.);

NDate1 = INPUT(Date1,MMDDYY.);

NDate1Copy = NDate1;

DATALINES;

10 011593

20 103193

;

TITLE 'INPUT Function Example';

PROC CONTENTS DATA=one;

RUN;

PROC PRINT DATA=one;

FORMAT NDate1 MMDDYY6.;

RUN;

INPUT Function Example

-----Alphabetic List of Variables and Attributes-----

# Variable Type Len

ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ

2 Date1 Char 6

4 NDate1 Num 8

5 NDate1Copy Num 8

3 NVar1 Num 8

1 Var1 Char 2

INPUT Function Example

Obs Var1 Date1 NVar1 NDate1 NDate1Copy

1 10 011593 10 011593 12068

2 20 103193 20 103193 12357

**Writing External Files with SAS**

In the previous sections raw data records were read into a data set using INFILE and INPUT statements. This section will present the opposite case: existing SAS variables will be written to an external file using the FILE and PUT statements. The following example reads an existing SAS data set and writes raw data to an external file.

FILENAME raw ‘C:\BIOS511\CLASS.RAW’ ;

DATA temp;

SET bios511.class;

FILE raw;

PUT name $ 1-8 age 10-12 ht 15-20;

RUN;

OBS NAME SEX AGE HT WT

class

1 CHRISTIANSEN M 37 71 195

2 HOSKING J M 31 70 160

3 HELMS R M 41 74 195

4 PIGGY M F . 48 .

5 FROG K M 3 12 1

6 GONZO 14 25 45

DATA

step

raw CHRISTIA 37 71

HOSKING 31 70

HELMS R 41 74

PIGGY M . 48

FROG K 3 12

GONZO 14 25

#### The FILE Statement

FILE *file-specification* <options> <host options> ;

* The FILE statement specifies the output file for PUT statements in the current DATA step.
* By default PUT statement output is written to the SAS log.
* The FILE statement specifies an external file, not a SAS data set.
* More than one FILE statement can be used in a DATA step. PUT statements will write to the file defined by the most recently executed FILE statement.
* The FILE statement can be executed conditionally.
* “File-specification”

identifies an external file to which you want to write output with a PUT statement.

“File-specification” can have the following forms:

* *'external-file'*

specifies the physical name of an external file and must be enclosed in quotes.

* *fileref*

gives the fileref of an external file. The fileref must have been previously associated

with an external file in a FILENAME statement.

* LOG

directs lines produced by PUT statements to the SAS log. PUT statements are by

default written to the LOG. To write PUT statements to the LOG a FILE statement is

not needed. A FILE LOG statement is only used to restore the default action or to

specify additional options.

* PRINT

directs lines produced by PUT statements to the same print file as the output produced

by SAS procedures. When PRINT is the fileref, SAS uses carriage control

characters and writes the lines with the characteristics of a print file.

* Selected FILE options:

HEADER=*label*, NOTITLES to have user-supplied titles for external print files.

COLUMN=*var*, LINE=*var* to get the current locations of the pointer in the output buffer.

PAGESIZE=*ps*, LINESIZE=*ls*, N=*nl* to specify the number of lines per page and the number of columns and lines in the output buffer.

DLM=, DSD to control the delimiters used to write output in list format.

**The PUT Statement**

The PUT statement is almost exactly the opposite of the INPUT statement. The PUT statement writes the values of SAS variables to the output buffer, which is then written to the external file defined by the FILE statement.

SYNTAX:

PUT <specification1> <specification2> .. ;

Where specification can have any of the following forms:

Column variable start-end

List variable

Format variable format

Named variable =

Pointer-control @n, +n, /, #n, @

\_INFILE\_ writes contents of the input buffer to the output buffer

\_ALL\_writes the values of all variables in the PDV using named

output

\_PAGE\_ advances to the top of a new page in the output file (this

option is only available for print files)

* Column, list, formatted, named modes, and format lists are the same as defined in the INPUT statement.
* \_INFILE\_ and \_ALL\_ are useful for debugging your programs.
* A PUT statement can write lines to:
* the SAS log
* the SAS procedures output file (such as the Output window)
* an external file
* the location specified in the most recently executed FILE statement
* If no FILE statements are executed before a PUT statement, lines are written to the SAS log.
* A PUT statement can write lines containing:
* variable values
* character strings
* a combination of the above
* With specifications you can list items to be written, indicate their position, and specify how they are to be formatted.
* You can write variable values in list, column, formatted, or named output mode.

**Column Output**

PUT name 1-20 var1 22-23 var2 25-28 ;

**List Output**

PUT name var1 var2 ;

**Formatted Output**

PUT @1 name $char20. @22 var1 2. @25 var2 $3. ;

**Named Output**

PUT name= var1= var2= ;

Note:

If a DATA step exists for the sole purpose of writing values with PUT statements, the special data set name \_NULL\_ can be used on the DATA statement to prevent a SAS data set from being created.

**PUT Statement Example 1**

174 DATA one;

175 SET bios511.class;

176

177 PUT name sex wt;

178 PUT name= sex= wt= ;

179

180 RUN;

CHRISTIANSEN M 195

NAME=CHRISTIANSEN SEX=M WT=195

HOSKING J M 160

NAME=HOSKING J SEX=M WT=160

HELMS R M 195

NAME=HELMS R SEX=M WT=195

PIGGY M F .

NAME=PIGGY M SEX=F WT=.

FROG K M 1

NAME=FROG K SEX=M WT=1

GONZO 45

NAME=GONZO SEX= WT=45

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: The data set WORK.ONE has 6 observations and 5 variables.

NOTE: DATA statement used:

real time 0.05 seconds

**PUT Statement Example 2**

181 DATA \_NULL\_;

182 SET bios511.class;

183

184 IF sex = 'M' THEN PUT 'Data Dump:' \_ALL\_;

185

186 RUN;

Data Dump:NAME=CHRISTIANSEN SEX=M AGE=37 HT=71 WT=195 \_ERROR\_=0 \_N\_=1

Data Dump:NAME=HOSKING J SEX=M AGE=31 HT=70 WT=160 \_ERROR\_=0 \_N\_=2

Data Dump:NAME=HELMS R SEX=M AGE=41 HT=74 WT=195 \_ERROR\_=0 \_N\_=3

Data Dump:NAME=FROG K SEX=M AGE=3 HT=12 WT=1 \_ERROR\_=0 \_N\_=5

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: DATA statement used:

real time 0.04 seconds

cpu time 0.03 seconds

###### PUT Statement Example 3

194 DATA \_NULL\_;

195 SET bios511.class;

196

197 IF sex NOT IN ('M','F') THEN PUT

198 'Bad value for sex ' name= +2 sex= ;

199

200 IF age <= .z THEN PUT name +3 "isn't telling";

201

202 IF ht < 20 THEN PUT name= ht= 8.2;

203

204 RUN;

PIGGY M isn't telling

NAME=FROG K HT=12.00

Bad value for sex NAME=GONZO SEX=

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: DATA statement used:

real time 0.03 seconds

cpu time 0.03 seconds

#### PUT Statement Example 4

205 DATA one;

206 SET bios511.class;

207

208 IF sex NOT IN ('F','M') THEN DO;

209 PUT 'Invalid value for sex ' name sex;

210 DELETE;

211 END;

212

213 RUN;

Invalid value for sex GONZO

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: The data set WORK.ONE has 5 observations and 5 variables.

NOTE: DATA statement used:

real time 0.04 seconds

cpu time 0.04 seconds

#### PUT Statement Example 5: Checking Input Data

214 DATA one;

215 INPUT id sex ms $ ss;

216

217 IF NOT(1<=sex<=2) THEN PUT

218 'Invalid value for sex' +2 id= +2 sex= ;

219

220 IF NOT(ms IN ('M','N','O')) THEN PUT

221 'Invalid value for ms' +2 id= +2 ms= ;

222

223 DATALINES;

Invalid value for sex id=1 sex=0

Invalid value for ms id=3 ms=Q

NOTE: The data set WORK.ONE has 4 observations and 4 variables.

NOTE: DATA statement used:

real time 0.04 seconds

cpu time 0.04 seconds

#### PUT Statement Example 6: Creating an External File / Data \_NULL\_

229 FILENAME raw 'e:\bios511\putex.dat';

230

231 DATA \_NULL\_;

232 SET bios511.class;

233

234 FILE raw;

235

236 PUT @1 name $20. @22 sex $1. @24 age 3. @28 ht 3. @32 wt z3.;

237

238 RUN;

NOTE: The file RAW is:

File Name=e:\bios511\putex.dat,

RECFM=V,LRECL=256

NOTE: 6 records were written to the file RAW.

The minimum record length was 34.

The maximum record length was 34.

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: DATA statement used:

real time 0.11 seconds

###### PUT Statement Example 7: Checking Input Data

39 DATA \_NULL\_;

40 SET sc.v1(IN=in1)

41 sc.v2(IN=in2) END=eof ;

42 BY id;

43

44 count1 + in1 ;

45 count2 + in2 ;

46 count3 + first.id ;

47

48 IF (eof) THEN PUT

49

50 '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' /

51 ' # of records in v1 is: ' count1 /

52 ' # of records in v2 is: ' count2 /

53 ' # of unique IDs is: ' count3 /

54 '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' ;

55

56

57 RUN;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# of records in v1 is: 151

# of records in v2 is: 148

# of unique IDs is: 151

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NOTE: The DATA statement used 0.42 seconds.

###### PUT Statement Example 8: Correcting Data

239 DATA one;

240 SET bios511.class;

241

242 IF sex = ' ' THEN DO;

243

244 PUT 'Before corrected: ' name= sex= ;

245

246 sex = 'F';

247

248 PUT 'After corrected: ' name= sex= ;

249

250 END;

251

252 RUN;

Before corrected: NAME=GONZO SEX=

After corrected: NAME=GONZO SEX=F

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: The data set WORK.ONE has 6 observations and 5 variables.

NOTE: DATA statement used:

real time 0.12 seconds

cpu time 0.05 seconds

**PUT Function**

PUT(*source*,*format*) ;

* Writes the value of a variable to a character variable using the specified format.
* The result of the PUT function is always a character string.
* The format must be the same type as the source.
* If the source is numeric, the resulting string is right aligned.
* If the source is character, the resulting string is left aligned.
* The PUT function can be used to convert a numeric variable to a character variable.
* The PUT function is also an efficient way to recode a variable.
* To preserve the result of the PUT function, you must assign it to a new variable.
* Examples:

x=44 ;

y=put(x,4.) ; \*\* y=' 44' \*\* ;

a=123;

b=put(a,z4.) ; \*\* b='0123'\*\* ;

**PUT function vs. INPUT function:**

PUT(source,format): char OR num 🡪 char (so use for num 🡪 char)

INPUT(source,informat): char 🡪 char OR num (so use for char 🡪 num)**PUT Function Example 1: Recoding**

## Recode AGE to a character variable with values Group 1, Group 2, or Invalid.

253 PROC FORMAT;

254 VALUE aft 0-40 = 'Group 1'

255 40<-HIGH = 'Group 2'

256 OTHER = 'Invalid';

NOTE: Format AFT has been output.

257 RUN;

NOTE: PROCEDURE FORMAT used:

real time 0.27 seconds

cpu time 0.05 seconds

258

259 DATA one;

260 SET bios511.class(KEEP=name age);

261

262 newage = PUT(age,aft7.);

263

264 PUT \_ALL\_;

265 RUN;

NAME=CHRISTIANSEN AGE=37 newage=Group 1 \_ERROR\_=0 \_N\_=1

NAME=HOSKING J AGE=31 newage=Group 1 \_ERROR\_=0 \_N\_=2

NAME=HELMS R AGE=41 newage=Group 2 \_ERROR\_=0 \_N\_=3

NAME=PIGGY M AGE=. newage=Invalid \_ERROR\_=0 \_N\_=4

NAME=FROG K AGE=3 newage=Group 1 \_ERROR\_=0 \_N\_=5

NAME=GONZO AGE=14 newage=Group 1 \_ERROR\_=0 \_N\_=6

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: The data set WORK.ONE has 6 observations and 3 variables.

NOTE: DATA statement used:

real time 0.36 seconds

cpu time 0.05 seconds

###### PUT Function Example 2: Recoding

Recode AGE to a numeric variable with values 1, 2, 3, or missing ( . ).

266 PROC FORMAT;

267 VALUE aft 0-<30 ='1'

268 30-<40 ='2'

269 40-HIGH ='3'

270 OTHER ='.';

NOTE: Format AFT is already on the library.

NOTE: Format AFT has been output.

271 RUN;

NOTE: PROCEDURE FORMAT used:

real time 0.01 seconds

cpu time 0.01 seconds

272

273 DATA one;

274 SET bios511.class(KEEP=name age);

275

276 temp = PUT(age,aft1.);

277 newage = INPUT(temp,1.);

278

279 /\* or

280 newage = INPUT((PUT(age,aft1.)),1.);

281 \*/

282

283 PUT \_ALL\_;

284 RUN;

NAME=CHRISTIANSEN AGE=37 temp=2 newage=2 \_ERROR\_=0 \_N\_=1

NAME=HOSKING J AGE=31 temp=2 newage=2 \_ERROR\_=0 \_N\_=2

NAME=HELMS R AGE=41 temp=3 newage=3 \_ERROR\_=0 \_N\_=3

NAME=PIGGY M AGE=. temp=. newage=. \_ERROR\_=0 \_N\_=4

NAME=FROG K AGE=3 temp=1 newage=1 \_ERROR\_=0 \_N\_=5

NAME=GONZO AGE=14 temp=1 newage=1 \_ERROR\_=0 \_N\_=6

NOTE: There were 6 observations read from the data set BIOS511.CLASS.

NOTE: The data set WORK.ONE has 6 observations and 4 variables.

NOTE: DATA statement used:

real time 0.05 seconds

cpu time 0.05 seconds

###### 

#### Customized Report Writing with the DATA Step

* The DATA step can be used to write programs that generate customized reports.
* PUT statements can be used to write detailed reports
* The FILE statement defines the destination of the lines written with PUT statements and provides several useful options.
* The special fileref PRINT on the FILE statement directs lines written by the PUT statements to the standard SAS print file (the Output window in the windowing environment or the \*.lst file in batch mode).
* The NOTITLES option on the FILE statement suppresses any currently defined titles.
* The HEADER= option on the FILE statement can be used to write user-defined headings whenever SAS goes to a new page.
* The N=PS option on the FILE statement allows the user to access a whole page at a time.
* Since the FILE and PUT statements are used in a DATA step, the full power of SAS programming statements is available for writing reports.
* In general, DATA step reports are a lot more work than using a procedure. Therefore, this approach should be a last resort, used only after you have convinced yourself that there is no easier way.

**Write a simple report with a PUT statement**

DATA \_NULL\_;

SET sashelp.class;

FILE PRINT;

PUT @21 name $8. @33 sex $1. @39 age 2.

@47 height 4.1 @57 weight 5.1;

RUN;

The SAS System

Alfred M 14 69.0 112.5

Alice F 13 56.5 84.0

Barbara F 13 65.3 98.0

Carol F 14 62.8 102.5

Henry M 14 63.5 102.5

James M 12 57.3 83.0

Jane F 12 59.8 84.5

Janet F 15 62.5 112.5

Jeffrey M 13 62.5 84.0

John M 12 59.0 99.5

Joyce F 11 51.3 50.5

Judy F 14 64.3 90.0

Louise F 12 56.3 77.0

Mary F 15 66.5 112.0

Philip M 16 72.0 150.0

Robert M 12 64.8 128.0

Ronald M 15 67.0 133.0

Thomas M 11 57.5 85.0

William M 15 66.5 112.0

**Enhance the report**

* Suppress the title (NOTITLES option).
* Add column headings (HEADER= option).

DATA \_NULL\_;

SET sashelp.class;

FILE PRINT NOTITLES HEADER=colhead;

PUT @21 name $8. @33 sex $1. @39 age 2.

@47 height 4.1 @57 weight 5.1;

RETURN;

colhead:

PUT @21 ‘Name’ @32 ‘Sex’ @39 ‘Age’

@46 ‘Height’ @56 ‘Weight’;

RETURN;

RUN;

Name Sex Age Height Weight

Alfred M 14 69.0 112.5

Alice F 13 56.5 84.0

Barbara F 13 65.3 98.0

Carol F 14 62.8 102.5

Henry M 14 63.5 102.5

James M 12 57.3 83.0

Jane F 12 59.8 84.5

Janet F 15 62.5 112.5

Jeffrey M 13 62.5 84.0

John M 12 59.0 99.5

Joyce F 11 51.3 50.5

Judy F 14 64.3 90.0

Louise F 12 56.3 77.0

Mary F 15 66.5 112.0

Philip M 16 72.0 150.0

Robert M 12 64.8 128.0

Ronald M 15 67.0 133.0

Thomas M 11 57.5 85.0

William M 15 66.5 112.0

**Final report**

* Add a column for the observation number.
* Skip a line between the heading and the first observation.

DATA \_NULL\_;

SET sashelp.class;

FILE PRINT NOTITLES HEADER=colhead;

RETAIN n 0;

n = n + 1;

PUT @15 n 2. @21 name $8. @33 sex $1. @39 age 2.

@47 height 4.1 @57 weight 5.1;

RETURN;

colhead:

PUT @14 ‘Obs’ @21 ‘Name’ @32 ‘Sex’ @39 ‘Age’

@46 ‘Height’ @56 ‘Weight’ /;

RETURN;

RUN;

Obs Name Sex Age Height Weight

1 Alfred M 14 69.0 112.5

2 Alice F 13 56.5 84.0

3 Barbara F 13 65.3 98.0

4 Carol F 14 62.8 102.5

5 Henry M 14 63.5 102.5

6 James M 12 57.3 83.0

7 Jane F 12 59.8 84.5

8 Janet F 15 62.5 112.5

9 Jeffrey M 13 62.5 84.0

10 John M 12 59.0 99.5

11 Joyce F 11 51.3 50.5

12 Judy F 14 64.3 90.0

13 Louise F 12 56.3 77.0

14 Mary F 15 66.5 112.0

15 Philip M 16 72.0 150.0

16 Robert M 12 64.8 128.0

17 Ronald M 15 67.0 133.0

18 Thomas M 11 57.5 85.0

19 William M 15 66.5 112.0

**Controlling an entire page**

First, design the report. You will probably want to use graph paper! (See the handout.)

* Put females on the left side of the page.
* Put males on the right side of the page.
* Use appropriate column headings.

In your SAS code, use the N=PS option to access a whole page.

DATA \_null\_;

RETAIN f 5 m 5;

SET sashelp.class;

FILE PRINT NOTITLES HEADER=h N=PS;

IF UPCASE(sex)=’F’ THEN DO;

f = f + 1;

PUT #f @2 name $8. +2 age 2.

+4 height 4.1 +4 weight 5.1;

END;

ELSE IF UPCASE(sex)=’M’ THEN DO;

m = m + 1;

PUT #m @45 name $8. +2 age 2.

+4 height 4.1 +4 weight 5.1;

END;

RETURN;

h:

PUT @7 ‘Listing of Females’

@52 ‘Listing of Males’ //

@2 ‘Name’ @11 ‘Age’ @17 ‘Height’ @25 ‘Weight’

@45 ‘Name’ @54 ‘Age’ @60 ‘Height’ @68 ‘Weight’;

RETURN;

RUN;

**Generated report**

Listing of Females Listing of Males

Name Age Height Weight Name Age Height Weight

Alice 13 56.5 84.0 Alfred 14 69.0 112.5

Barbara 13 65.3 98.0 Henry 14 63.5 102.5

Carol 14 62.8 102.5 James 12 57.3 83.0

Jane 12 59.8 84.5 Jeffrey 13 62.5 84.0

Janet 15 62.5 112.5 John 12 59.0 99.5

Joyce 11 51.3 50.5 Philip 16 72.0 150.0

Judy 14 64.3 90.0 Robert 12 64.8 128.0

Louise 12 56.3 77.0 Ronald 15 67.0 133.0

Mary 15 66.5 112.0 Thomas 11 57.5 85.0

William 15 66.5 112.0

**Print the PAYROLL data set**

PROC PRINT DATA=bios511.payroll; RUN;

The SAS System

Gross

Obs Name Dept Sex NetPay Pay Phone

1 Hafer 911 F 96.64 121.95 2561

2 Young 911 M 229.69 313.60 6263

3 Reynolds 911 F 134.03 174.15 1727

4 Stride 911 M 272.53 383.40 7281

5 Isaac 911 M 219.91 313.60 2718

6 Green 911 M 238.04 365.60 2182

7 Smothers 911 M 202.43 315.20 8272

8 Krause 911 F 182.09 242.40 7222

9 Post 911 M 206.60 292.00 9293

10 Chung 911 M 167.53 243.95 2187

11 Larson 914 F 215.47 283.92 5262

12 Arnold 914 M 356.87 445.50 6473

13 Manhart 914 M 250.34 344.78 8273

14 Vetter 914 M 189.28 279.36 5327

15 Curci 914 M 215.31 376.00 2638

16 Greco 914 M 685.23 1004.00 2723

17 Zhao 914 M 291.56 399.20 8389

18 Johnson 914 M 135.23 180.72 2765

19 Thompson 914 M 221.09 297.56 2343

20 Corning 915 F 103.43 146.16 1273

21 Wood 915 F 238.17 372.24 1734

22 Green 915 M 725.16 1124.50 2156

23 Hayden 915 F 103.54 146.31 8827

24 Taylor 915 F 287.22 401.72 8374

25 Chapman 915 M 264.31 385.47 2154

26 Roche 915 M 287.37 402.12 1133

27 Moore 915 M 557.18 728.12 2358

28 Helms 915 M 479.26 701.26 6789

29 Farlow 916 M 527.19 842.03 3455

30 Shires 916 F 187.12 279.22 4353

31 Richards 916 M 219.27 352.84 7787

32 Smith 916 F 147.09 201.88 2232

33 Murphy 916 M 272.66 385.11 2099

34 Fairley 916 M 521.66 834.80 6009

35 Herzog 917 M 692.57 1045.20 7440

36 Tall 917 M 355.19 492.26 3830

37 Higgins 917 M 777.50 1235.40 5667

38 McKay 917 M 821.44 1392.20 3748

39 Sanford 917 F 284.06 405.37 3321

40 Clinton 917 F 169.06 272.29 9987

41 Brandon 918 M 554.31 804.64 4043

42 Hebert 918 M 224.36 310.40 3900

43 Clancy 918 M 245.37 347.12 2604

44 Powell 918 F 189.39 271.54 3017

45 Hicks 940 M 553.87 807.31 2806

46 Vance 940 M 487.03 789.00 3000

47 Carter 940 M 712.89 1158.20 2077

**Generating a payroll report**

See the handout for the report design.

PROC SORT DATA=bios511.payroll OUT=payroll;

BY dept;

RUN;

DATA \_NULL\_;

SET payroll END=eof;

BY dept;

FILE PRINT HEADER=h NOTITLES LINESLEFT=ll;

RETAIN nettot grosstot grandnet grandgro 0;

IF FIRST.dept THEN DO;

nettot = 0; grosstot = 0;

IF ll<14 THEN PUT \_PAGE\_;

END;

PUT @9 dept 3. @22 name $8. @36 phone 5.

@45 sex $1. @51 netpay 6.2 @62 grosspay 7.2;

nettot = nettot + netpay;

grosstot = grosstot + grosspay;

IF LAST.dept THEN DO;

PUT @50 ‘-------‘ @62 ‘-------‘ / ‘Department Total’

@47 nettot DOLLAR10.2

@59 grosstot DOLLAR10.2 //;

grandnet = grandnet + nettot;

grandgro = grandgro + grosstot;

END;

IF eof THEN PUT ‘Overall Total’

@47 grandnet DOLLAR10.2

@59 grandgro DOLLAR10.2;

RETURN;

h:

PUT // @25 ‘ABC Manufacturing Company’ /

@31 ‘Payroll Report’ /// @22 ‘Employee’

@34 ‘Telephone’ @52 ‘Net’ @63 ‘Gross’ /

@5 ‘Department’ @24 ‘Name’ @35 ‘Number’

@44 ‘Sex’ @52 ‘Pay’ @64 ‘Pay’ //;

RETURN;

RUN;

**The generated report**

ABC Manufacturing Company

Payroll Report

Employee Telephone Net Gross

Department Name Number Sex Pay Pay

911 Hafer 2561 F 96.64 121.95

911 Young 6263 M 229.69 313.60

911 Reynolds 1727 F 134.03 174.15

911 Stride 7281 M 272.53 383.40

911 Isaac 2718 M 219.91 313.60

911 Green 2182 M 238.04 365.60

911 Smothers 8272 M 202.43 315.20

911 Krause 7222 F 182.09 242.40

911 Post 9293 M 206.60 292.00

911 Chung 2187 M 167.53 243.95

------- -------

Department Total $1,949.49 $2,765.85

914 Larson 5262 F 215.47 283.92

914 Arnold 6473 M 356.87 445.50

914 Manhart 8273 M 250.34 344.78

914 Vetter 5327 M 189.28 279.36

914 Curci 2638 M 215.31 376.00

914 Greco 2723 M 685.23 1004.00

914 Zhao 8389 M 291.56 399.20

914 Johnson 2765 M 135.23 180.72

914 Thompson 2343 M 221.09 297.56

------- -------

Department Total $2,560.38 $3,611.04

915 Corning 1273 F 103.43 146.16

915 Wood 1734 F 238.17 372.24

915 Green 2156 M 725.16 1124.50

915 Hayden 8827 F 103.54 146.31

915 Taylor 8374 F 287.22 401.72

915 Chapman 2154 M 264.31 385.47

915 Roche 1133 M 287.37 402.12

915 Moore 2358 M 557.18 728.12

915 Helms 6789 M 479.26 701.26

------- -------

Department Total $3,045.64 $4,407.90

ABC Manufacturing Company

Payroll Report

Employee Telephone Net Gross

Department Name Number Sex Pay Pay

916 Farlow 3455 M 527.19 842.03

916 Shires 4353 F 187.12 279.22

916 Richards 7787 M 219.27 352.84

916 Smith 2232 F 147.09 201.88

916 Murphy 2099 M 272.66 385.11

916 Fairley 6009 M 521.66 834.80

------- -------

Department Total $1,874.99 $2,895.88

917 Herzog 7440 M 692.57 1045.20

917 Tall 3830 M 355.19 492.26

917 Higgins 5667 M 777.50 1235.40

917 McKay 3748 M 821.44 1392.20

917 Sanford 3321 F 284.06 405.37

917 Clinton 9987 F 169.06 272.29

------- -------

Department Total $3,099.82 $4,842.72

918 Brandon 4043 M 554.31 804.64

918 Hebert 3900 M 224.36 310.40

918 Clancy 2604 M 245.37 347.12

918 Powell 3017 F 189.39 271.54

------- -------

Department Total $1,213.43 $1,733.70

940 Hicks 2806 M 553.87 807.31

940 Vance 3000 M 487.03 789.00

940 Carter 2077 M 712.89 1158.20

------- -------

Department Total $1,753.79 $2,754.51

Overall Total $15,497.54 $23,011.60

**An unusual report: Draw a circle**

DATA \_NULL\_;

radius = 20;

FILE PRINT N=PS;

DO angle=0 TO 2\*3.1416 BY .02;

x = radius\*cos(angle) + 37;

y = 26 - radius\*sin(angle)\*.6;

PUT #y @x '\*';

END;

RUN;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* \*\*\*

\*\*\* \*\*\*

\*\* \*\*

\*\* \*\*

\*\* \*\*

\*\* \*\*

\* \*

\*\* \*\*

\* \*

\* \*

\* \*

\* \*\*

\* \*

\* \*

\*\* \*\*

\* \*

\*\* \*\*

\*\* \*\*

\*\* \*\*\*

\*\* \*\*

\*\*\* \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*