The George Washington University
Department of Statistics

STAT 6197 – Spring 2020

Week 2 – January 24, 2020

Major Topic: DATA Step: Reading Data, and Creating Reports

Detailed Topics:

1. Reading Raw Data Files into SAS Data Sets Using
    a. Column Input
    b. Formatted Input
    c. List Input
    d. Named Input
2. Handling Missing Values
3. Reading Microsoft Excel Data into SAS Data Sets
4. Creating Raw Tabular Data Files or Reports Using DATA _NULL_
5. Compilation vs. Execution (Revisited)

Readings:

1. Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012.

2. Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015.

3. Reading Raw Data with the INPUT Statement (SAS Documentation)
4. Some Rules for Reading Numeric and Character Data (SAS Documentation)
5. Ways to represent missing values for numeric variables in SAS
6. Call Missing Routine (SAS(R) 9.4 Documentation)
7. Six ways to list variables in SAS (SAS Blogs)
8. PUT it there! Six tips for using PUT and %PUT Statements in SAS (SAS Blogs)

Raw Data Files

A raw data file is also known as a *flat file*.

- They are text files that contain one record per line.
- A record typically contains multiple fields.
- Flat files do not have internal metadata.
- External documentation, known as a *record layout*, should exist.
- A record layout describes the fields and locations within each record.

3

Reading Raw Data Files

These are the steps for creating a SAS data set from a raw data file.

Step 1	Name the output SAS data set.
Step 2	Locate and name the input raw data file.
Step 3	Examine the raw data file.
Step 4	Examine the file layout.
Step 5	Give SAS names to the fields.
Step 6	Determine the variable type.

8

Variable Types

The two types of SAS variables are listed below:

- character
- numeric

VIEWTABLE: Project.Enroll

	first_name	last	state2	_age_	enrolldate
1	Danny	Brown	CO	.	15684
2	William	Johnson		22	17318
3	Samantha	McCormick	CA	47	16674
4	Tina	Stewart	TX	53	14287

three character variables

two numeric variables

62

Variable Types: Character

Character variables are stored with a length of 1 to 32,767 bytes with 1 character equaling 1 byte.

VIEWTABLE: Project.Enroll

	first_name	last	state2	_age_	enrolldate
1	Danny	Brown	CO	.	15684
2	William	Johnson		22	17318
3	Samantha	McCormick	CA	47	16674
4	Tina	Stewart	TX	53	14287

8 bytes

9 bytes

2 bytes

Character variables can contain letters (A-Z), numeric digits (0-9), and other special characters (_, #, %, &,…).

63

Variable Types: Numeric

Numeric variables are stored as floating-point numbers with a default byte size of 8.

VIEWTABLE: Project.Enroll

	first_name	last	state2	_age_	enrolldate
1	Danny	Brown	CO	.	15684
2	William	Johnson		22	17318
3	Samantha	McCormick	CA	47	16674
4	Tina	Stewart	TX	53	14287

8 bytes 8 bytes

To be stored as a floating point number, the numeric value can contain numeric digits (0-9), plus or minus sign, decimal point, and E for scientific notation.

64

Standard and Nonstandard Data (Review)

Standard data is data that SAS can read without any additional instruction.

- Character data is always standard.
- Some numeric values are standard and some are not.

Standard Numeric Data

58
67.23 -23
5.67E5 00.99
1.2E-2

Nonstandard Numeric Data

(23) $67.23
5,823 01/12/2010
12May2009

8

The following are the only acceptable characters in a standard numeric field:

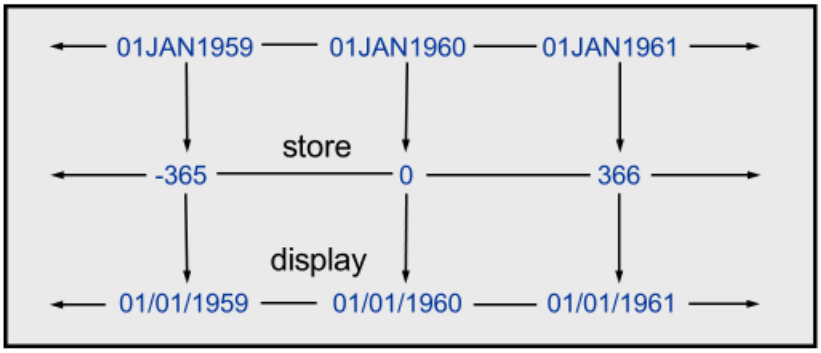
```
0 1 2 3 4 5 6 7 8 9 . E e D d - +
```

Leading or trailing blanks are also acceptable.

E, **e**, **D**, and **d** represent exponential notation in a standard numeric field. An alternate way of writing **300000**, for example, is **3E5**.

SAS Date Values

SAS stores calendar dates as numeric values.



A *SAS date value* is stored as the number of days between January 1, 1960, and a specific date.

11

SAS can read perform calculations on dates starting from A.D. 1582 through A. D. 19,900.

Raw Data Files

Fields in a raw data file can be delimited or arranged in fixed columns.

Delimited File

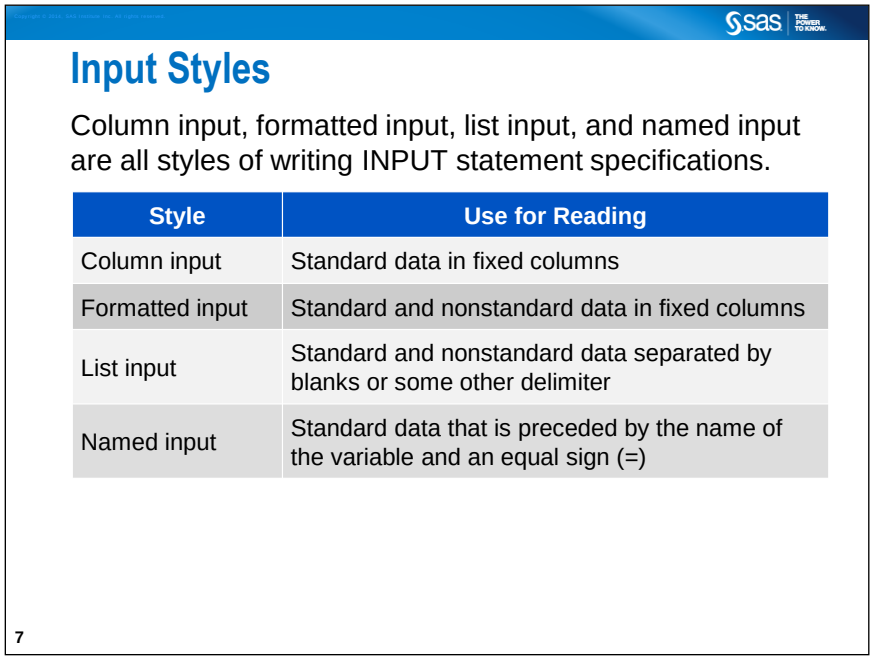
```
120102,Tom,Zhou,M,108255,Sales Manager,AU,11AUG1973,06/01/1993
120103,Wilson,Dawes,M,87975,Sales Manager,AU,22JAN1953,01/01/1978
120121,Irenie,Elvish,F,26600,Sales Rep. II,AU,02AUG1948,01/01/1978
120122,Christina,Ngan,F,27475,Sales Rep. II,AU,27JUL1958,07/01/1982
```

Fixed Column File

```
          1    1    2    2    3    3    4    4    5    5    6
1---5----0----5----0----5----0----5----0----5----0----5----0--
120102Tom         Zhou           Sales Manager        108255AU
120103Wilson      Dawes          Sales Manager         87975AU
120121Irenie      Elvish         Sales Rep. II         26600AU
120122Christina   Ngan           Sales Rep. II         27475AU
```

4

Reading Raw Data Files: Input Styles



The list input style that handles standard data and character variables whose values are of no more than 8 bytes long is called *simple list input*.

The list input style that additionally handles nonstandard data and character variables whose values are more than 8 bytes long is called *modified list input*.

Modified List Input = List input + some features of Formatted Input.

When to Use Column Input

Reading Fixed Column Files

Column input is used to read standard values from fixed column data files.

Partial **salesemp.dat**

```
          1    1    2    2    3    3    4    4    5    5    6
1---5----0----5----0----5----0----5----0----5----0----5----0--
120102 Tom        Zhou           Sales Manager       108255 AU
120103 Wilson     Dawes          Sales Manager       87975  AU
120121 Irenie     Elvish         Sales Rep. II       26600  AU
120122 Christina  Ngan           Sales Rep. II       27475  AU
120123 Kimiko     Hotstone       Sales Rep. I        26190  AU
```

- Formatted input is used to read standard and nonstandard values from fixed column data files.

16

Using Column Input

Column input requires a variable name, type, starting column, and ending column for each field to be read.

Partial **salesemp.dat**

```
          1    1    2    2    3    3    4    4    5    5    6
1---5----0----5----0----5----0----5----0----5----0----5----0--
120102 Tom        Zhou           Sales Manager       108255 AU
120103 Wilson     Dawes          Sales Manager       87975  AU
```

Name: EmpID
Type: Numeric
Columns: 1-7

Field	Type	Starting Column	Ending Column
Employee ID	Numeric	1	7
First Name	Character	8	17
Last Name	Numeric	19	32
Job Title	Character	34	52
Salary	Numeric	54	59
Country	Character	61	62

17

Using Column Input

Use INFILE and INPUT statements in a DATA step to read a raw data file.

```
data work.sales;
   infile "&path\salesemp.dat";
   input EmpID 1-7 First_Name $ 8-17
         Last_Name $ 19-32 Job_Title $ 34-52
         Salary 54-59 Country $ 61-62;
run;
```

```
DATA output-data-set;
    INFILE "raw-data-file";
    INPUT variable <$> start-end ...;
RUN;
```

18 p2aad02

INFILE Statement

The INFILE statement identifies the raw data file to be read.

```
infile "&path\salesemp.dat";
```

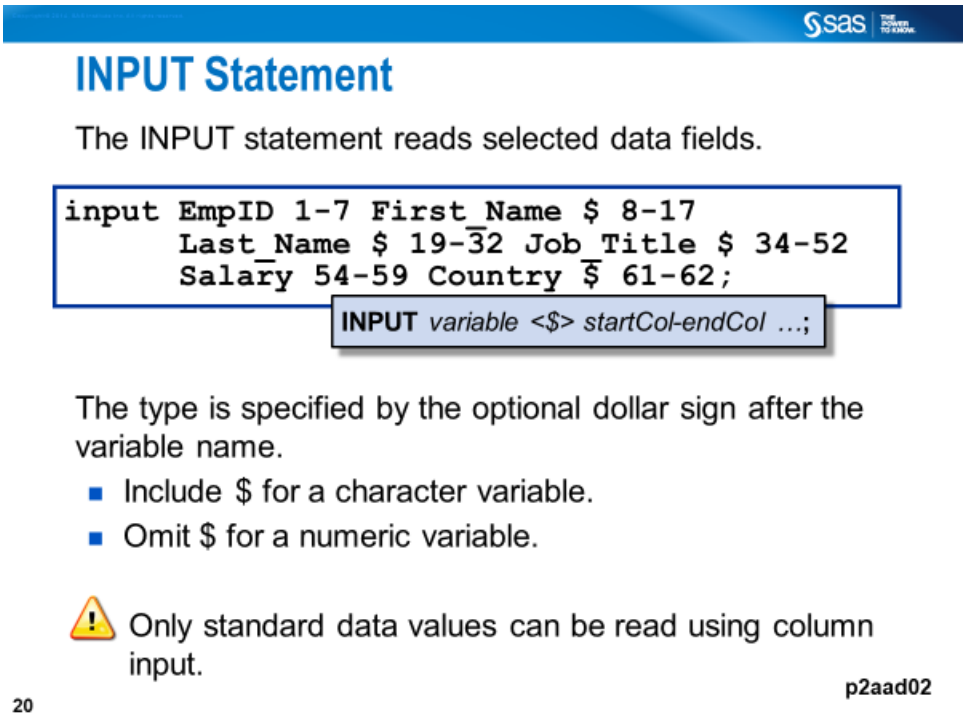
```
INFILE "raw-data-file";
```

- A full pathname is recommended.
- Using the **&path** macro variable reference makes the program more flexible.

⚠ Be sure to use double quotation marks when referencing a macro variable within a quoted string.

19

In the INFILE statement you can specify either the name the external file or point to the external file with a fileref defined with the FILENAME statement, unless you have instream data following the DATALINES statement. You may also need to specify one of more options to this statement.



Advantages of Column Input

- Data fields can be read in any order.
- The same field can be reread.
- Parts of a data field can be read
- Both leading and trailing blanks within the field are ignored.
- Character values can contain embedded blanks.
- Placeholders (e.g., period), are not required for missing data.

Disadvantage: The *Column Input* cannot read nonstandard data values.

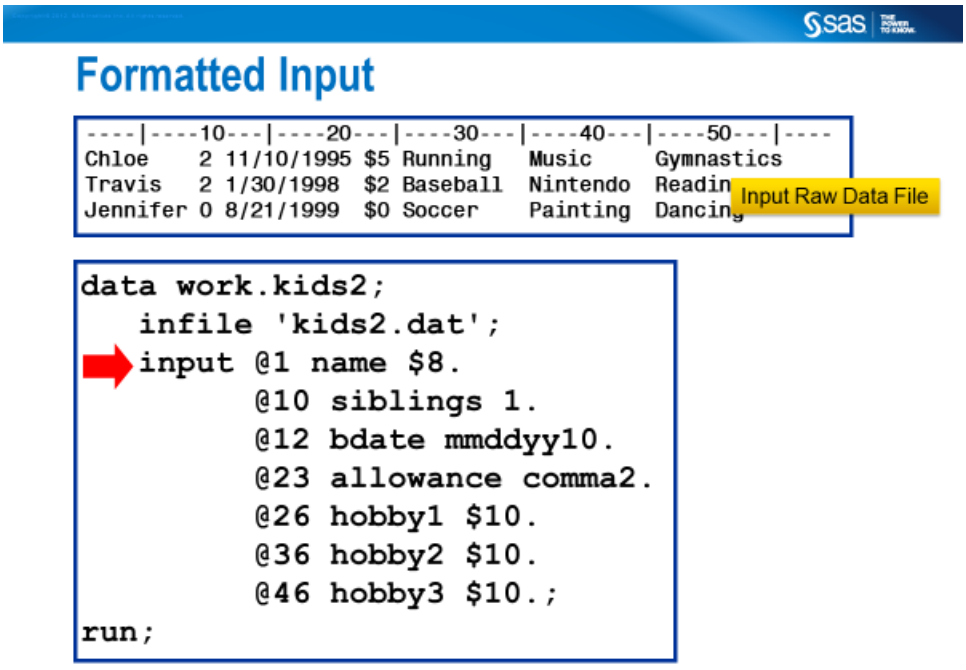
When to Use Formatted Input

Records with standard numeric, nonstandard numeric or character data fields aligned in columns

Features of the formatted input style

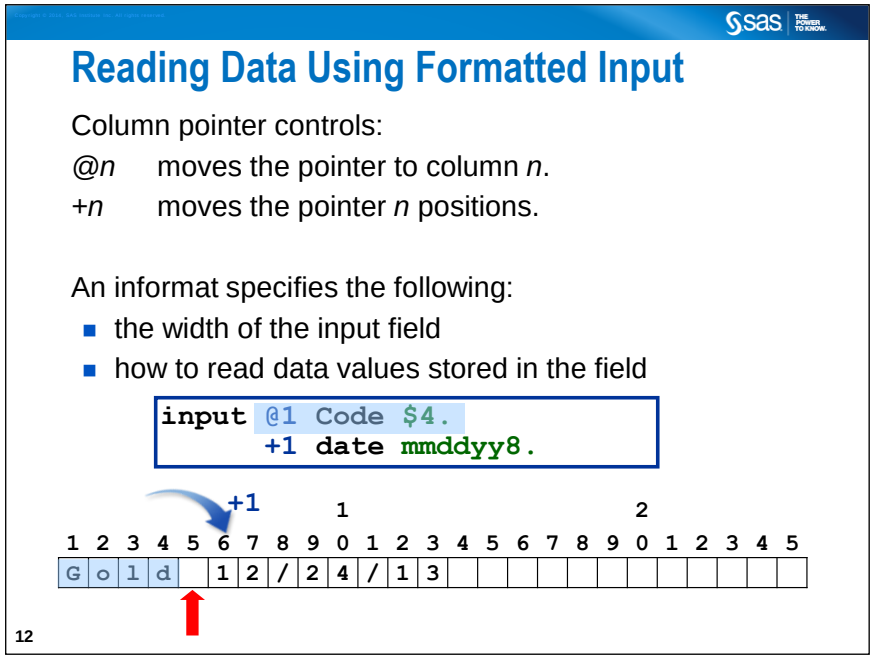
- Starting position
- Variable
- INFORMAT

The informat name must contain a period (.) at the end for SAS to distinguish it (i.e., informat name) from a variable name as can be seen from the code below.



"Formatted input causes the pointer to move like that of column input to read a variable value. The pointer moves the length that is specified in the informat and stops at the next column." – SAS Documentation.

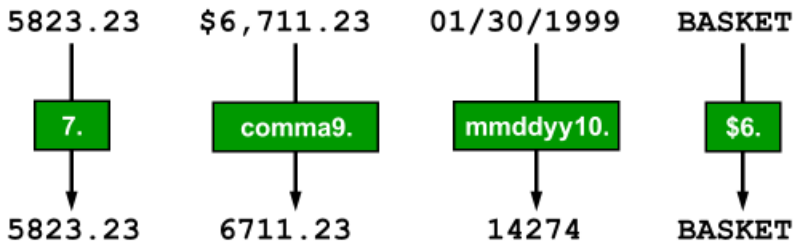
Formatted Input: Use of Absolute Pointer Control vs. Relative Pointer Control



Remember that when you use formatted input, the column pointer control moves to the first column following the field that was just read.

Informats

An *informat* is an instruction that SAS uses to read data values into a variable.



SAS uses the informat to determine the following:

- whether the variable is numeric or character
- the length of character variables

48

a

Informats

Raw Data Value	Informat	SAS Data Value
$12,345	COMMA7. DOLLAR7.	12345
$12.345	COMMAX7. DOLLARX7.	12345
€12.345	EUROX7.	12345
Australia	$11.	Australia
Australia	$CHAR11.	Australia
au	$UPCASE2.	AU
01/01/1960	MMDDYY10.	0
31/12/60	DDMMYY8.	365
31DEC1959	DATE9.	-1

53

The ANYDTDTE*w.* informat can be used to read data that has a variety of date forms.

Informats always contain a *w* value to indicate the width of the raw data field. A perod (.) ends the informat or separates the *w* value from the optional *d* value, which specifies the number of decimal places.

SAS Format Examples

Selected SAS formats:

Format	Stored Value	Displayed Value
$4.	Programming	Prog
12.	27134.5864	27135
12.2	27134.5864	27134.59
COMMA12.2	27134.5864	27,134.59
DOLLAR12.2	27134.5864	$27,134.59
COMMAX12.2	27134.5864	27.134,59
EUROX12.2	27134.5864	€27.134,59

10

SAS Date Format Examples

SAS date formats display SAS date values in standard date forms.

Format	Stored Value	Displayed Value
MMDDYY10.	0	01/01/1960
MMDDYY8.	0	01/01/60
MMDDYY6.	0	010160
DDMMYY10.	365	31/12/1960
DDMMYY8.	365	31/12/60
DDMMYY6.	365	311260

13

Column Input: LRECL= and PAD Options on the INFILE Statement (Handling Shorter Records)

Example data in fixed columns in an external file:

```
001 Tim Dyson    74 8724
002 Sam Larson   96 8224
003 Jane Miller  91
004 Bikas Das    90 8783
```

Shorter record (not padded with blanks)

```
Data HAVE;
 infile 'C:\SASCourse\Week2\short_records.txt'   Lrecl=25 PAD;
 input id 1-3 name $ 5-16  score 18-19 @21 some_value 5.2;
run;
```

With the PAD option on the INFILE statement, SAS pads the records that are in fixed columns and read from an external file with blanks to the length that is specified in the LRECL= option or implied by the column position.

TRUNCOVER Option on the INFILE Statement

Example data in an external file:

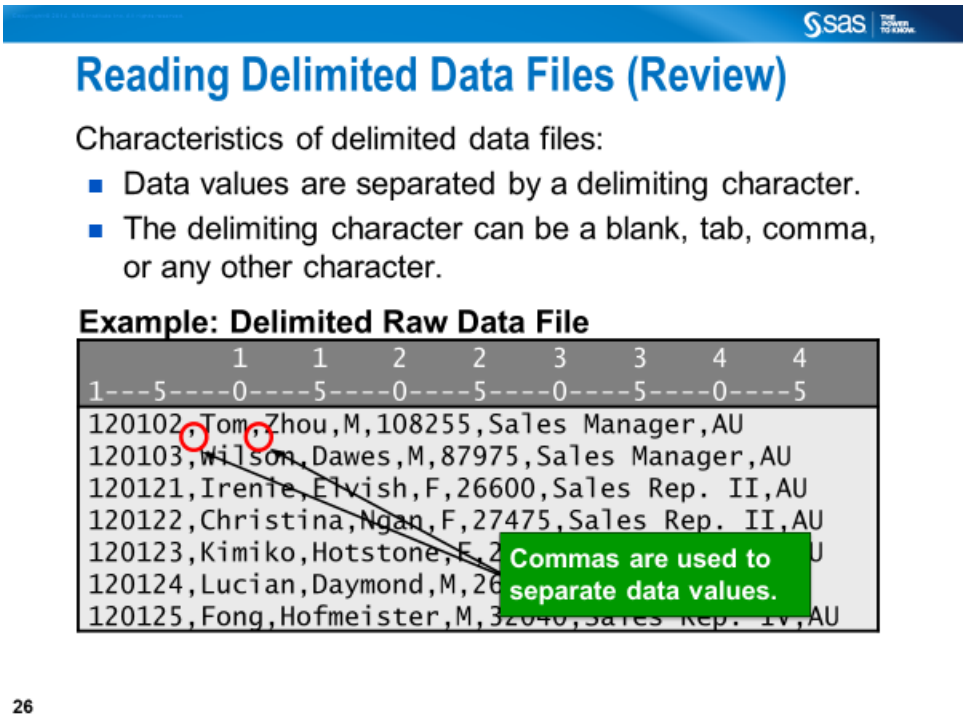
```
001 Tim Dyson    74 8724
002 Sam Larson   96  224
003 Jane Miller  91   24
004 Bikas Das    90    4
```

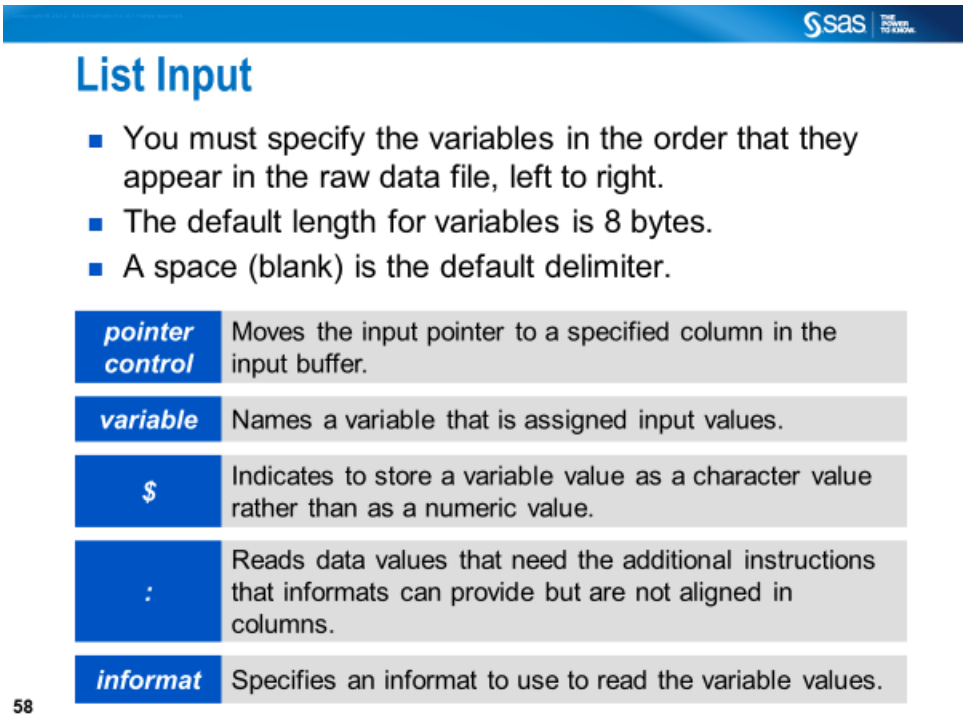
For the last three records (above), the very last field is shorter than expected. The TRUNCOVER option on the INFILE statement correctly assign the contents of the input buffer to the last field.

```
data HAVE;
infile 'C:\SASCourse\Week2\short_values.txt' TRUNCOVER;
 input id 1-3 name $ 5-16 score 18-19 @21 some_value 5.2;
run;
```

When to Use List Input

- Free-Format Data (Space or Character-Delimited Files).
- Data fields have character or numeric values.





Reading a Delimited Raw Data File

Use *INFILE* and *INPUT statements* in a DATA step to read a raw data file.

```
data work.subset;
   infile "&path\sales.csv" dlm=',';
   input Employee_ID First_Name $
         Last_Name $ Gender $ Salary
         Job_Title $ Country $;
run;
```

DATA *output-data-set*;
INFILE "*raw-data-file*" <**DLM**='*delimiter*'>;
INPUT *variable* <**$**> *variable* <**$**> … ;
RUN;

11 p108d01

INPUT Statement

The INPUT statement reads the data fields sequentially, left to right. Standard data fields require only a variable name and type.

Partial **sales.csv**

```
120102,Tom,Zhou,M,108255,Sales Manager,AU,11AUG1969,06/01/1989
120103,Wilson,Dawes,M,87975,Sales Manager,AU,22JAN1949,01/01/1974
120121,Irenie,Elvish,F,26600,Sales Rep. II,AU,02AUG1944,01/01/1974
```

```
input Employee_ID First_Name $ Last_Name $
      Gender $ Salary Job_Title $ Country $;
```

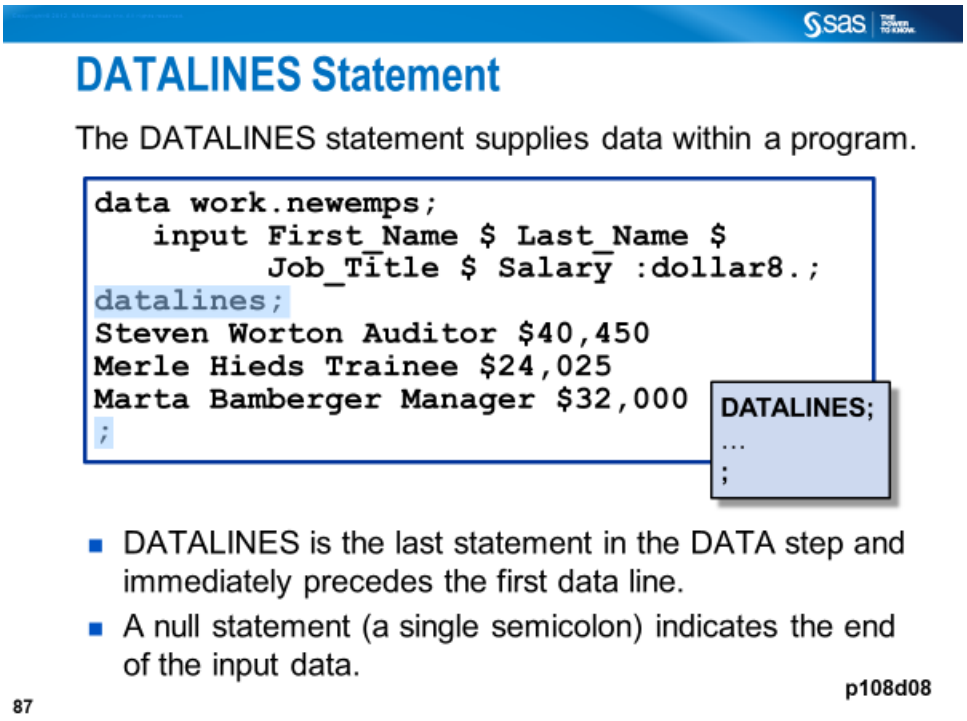
INPUT *variable* <$> *variable* <$> …;

- The optional dollar sign indicates a character variable.
- Default length for ***all*** variables is eight bytes, regardless of type.

13

Modified List Input: Using the Colon Modifier in the Input Statement

- Free-Format Data
- Data fields separated by 1+ blanks or other delimiters
- Data fields have standard numeric values or characters more than 8 bytes
- Data fields have non-standard numeric values



The colon (:) format modifier enables you to use list input but also to specify an informat after a variable name, whether character or numeric. SAS reads until it encounters a blank column, the defined length of the variable (character only), or the end of the data line, whichever comes first.

Here the INFORMAT specifies the length for each character variable, not the number of columns that are read.

How SAS Treats Variables When You Assign Informats with the INFORMAT Statement

The LENGTH Statement

A LENGTH statement contains the names of the variables followed by the number of bytes to be used for storage.

```
LENGTH  RECTYPE  3 HHX  $ 6 INTV_QRT 3 INTV_MON 3;
```

The length of a variable is the number of bytes SAS allocates for storing the variable. It is not necessarily the same as the number of characters in the variable. The LENGTH statement is used in the data step to reduce the size (in terms of disk space) of SAS data sets.

By default, SAS uses 8 bytes to store numeric variables. Variables containing integer values can be stored using less than 8 bytes.

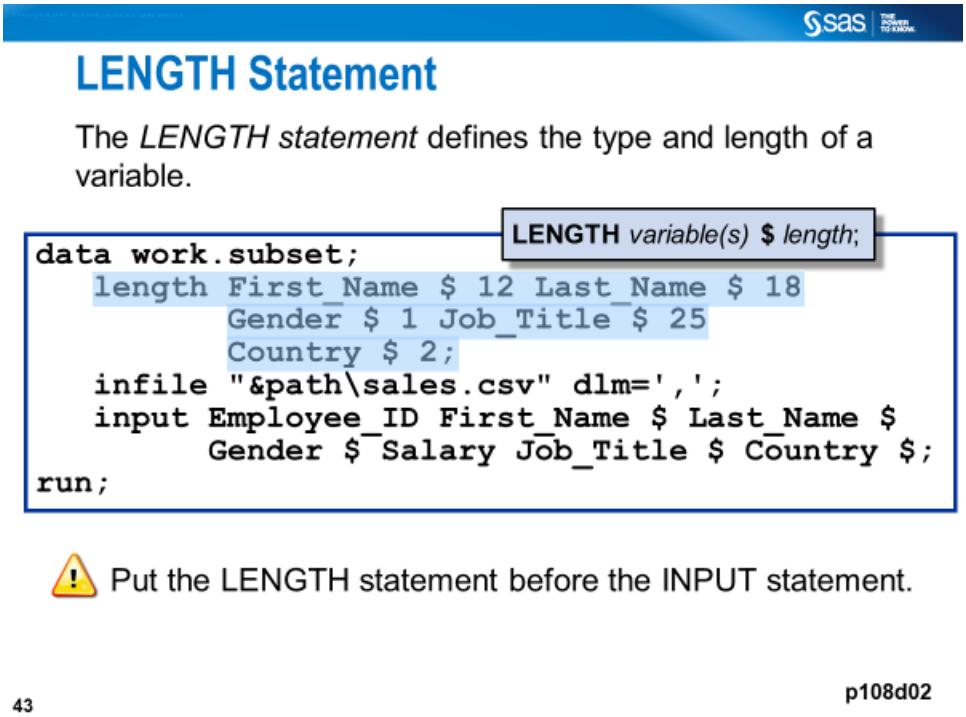
Largest integer represented exactly by length for SAS variables under Windows

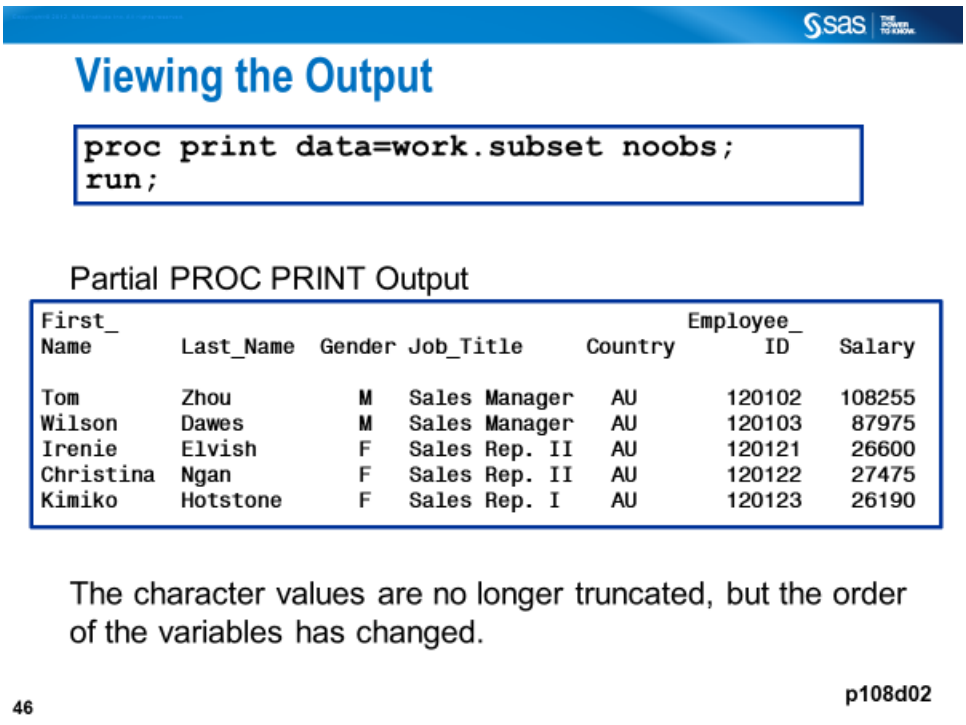
Length in bytes	Largest integer represented exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,990

Data sets containing many integer variables (such are common with data collected by questionnaire) or indicator variables can be reduced in size by more than 50%.

Variables containing real numbers should be left with the default length of 8. Note that specifying a length less than that required will result in a loss of precision without any warning being given.

LENGTH Statement for Character Fields





LENGTH Statement to Define the Length of Both Numeric and Character Variables

Using a LENGTH Statement

The LENGTH statement identifies the character variables, so dollar signs can be omitted from the INPUT statement.

```
data work.subset;
   length Employee_ID 8 First_Name $ 12
          Last_Name $ 18 Gender $ 1
          Salary 8 Job_Title $ 25
          Country $ 2;
   infile "&path\sales.csv" dlm=',';
   input Employee_ID First_Name Last_Name
         Gender Salary Job_Title Country;
run;
```

p108d03

49

Viewing the Output

Display the variables in creation order.

```
proc contents data=work.subset varnum;
run;
```

Partial PROC CONTENTS Output

Variables in Creation Order

#	Variable	Type	Len
1	Employee_ID	Num	8
2	First_Name	Char	12
3	Last_Name	Char	18
4	Gender	Char	1
5	Salary	Num	8
6	Job_Title	Char	25
7	Country	Char	2

50

Reading Non-Standard Data: Modifier with an Informat instead of the Informat Statement

Considerations

Use modified list input to read all the fields from **sales.csv**. Store the date fields as SAS dates.

Partial **sales.csv**

```
120102,Tom,Zhou,M,108255,Sales Manager,AU,11AUG1973,06/01/1993
120103,Wilson,Dawes,M,87975,Sales Manager,AU,22JAN1953,01/01/1978
120121,Irenie,Elvish,F,26600,Sales Rep. II,AU,02AUG1948,01/01/1978
120122,Christina,Ngan,F,27475,Sales Rep. II,AU,27JUL1958,07/01/1982
120123,Kimiko,Hotstone,F,26190,Sales Rep. I,AU,28SEP1968,10/01/1989
```

63

Modified List Input

This DATA step uses *modified list input.* Instead of a LENGTH statement, an informat specifies the length for each character variable.

input *variable* <:*informat.*> …;

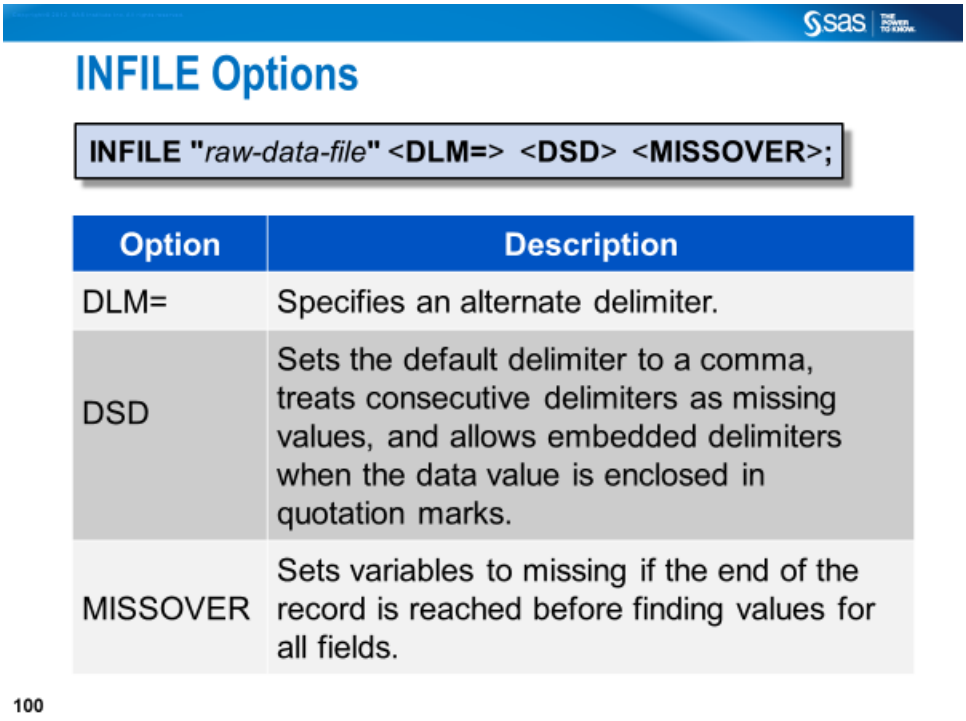
```
data work.subset;
   infile "&path\sales.csv" dlm=',';
   input Employee_ID First_Name :$12.
         Last_Name :$18. Gender :$1. Salary
         Job_Title :$25. Country :$2.;
run;
```

- The **$12.** informat defines a length of 12 for **First_Name** and allows up to 12 characters to be read.
- The **:** format modifier tells SAS to read until it encounters a delimiter.

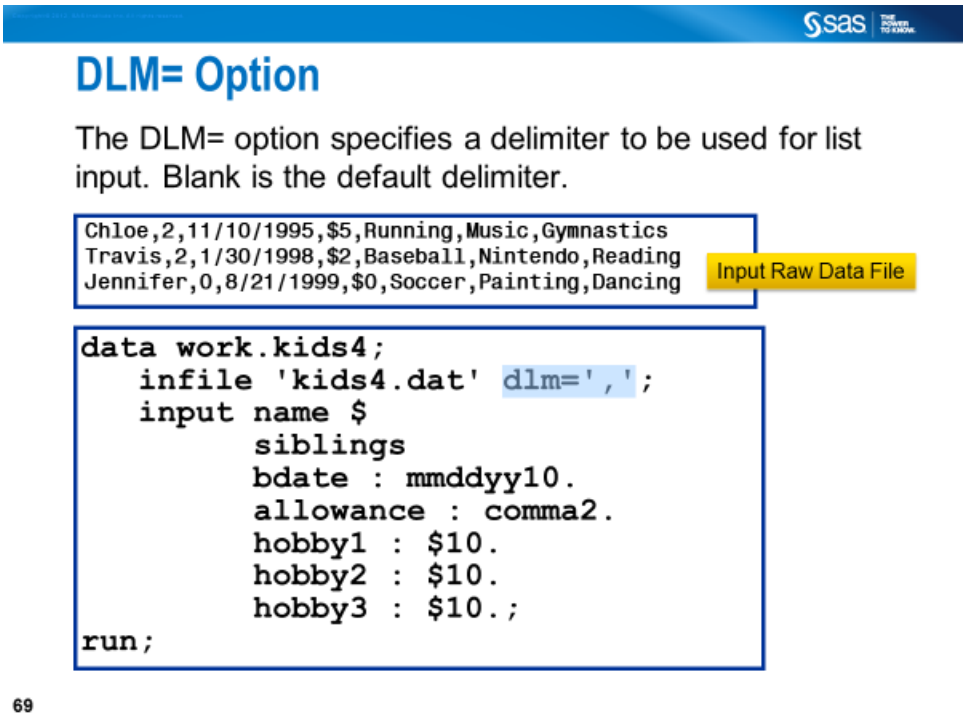
64 p108d05

The INFORMAT statement has the same impact of the LENGTH statement for character variables.

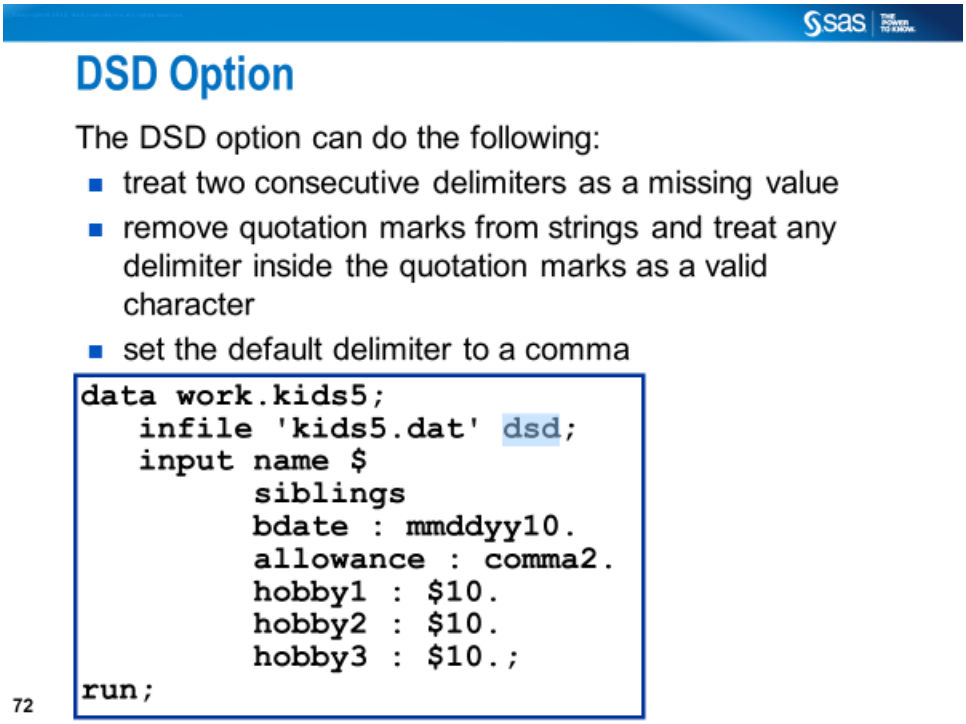
```
DATA Work.Have3;
  INFORMAT st_name $ 10.;
   INPUT st_name $ pop percent_pop18p ;
   DATALINES;
   Alabama 4833722  77
   California 38332521 76.1
 ;
PROC PRINT data=work.Have3 noobs; run;
proc contents data=Have3 varnum;
ods select position;
run;
```

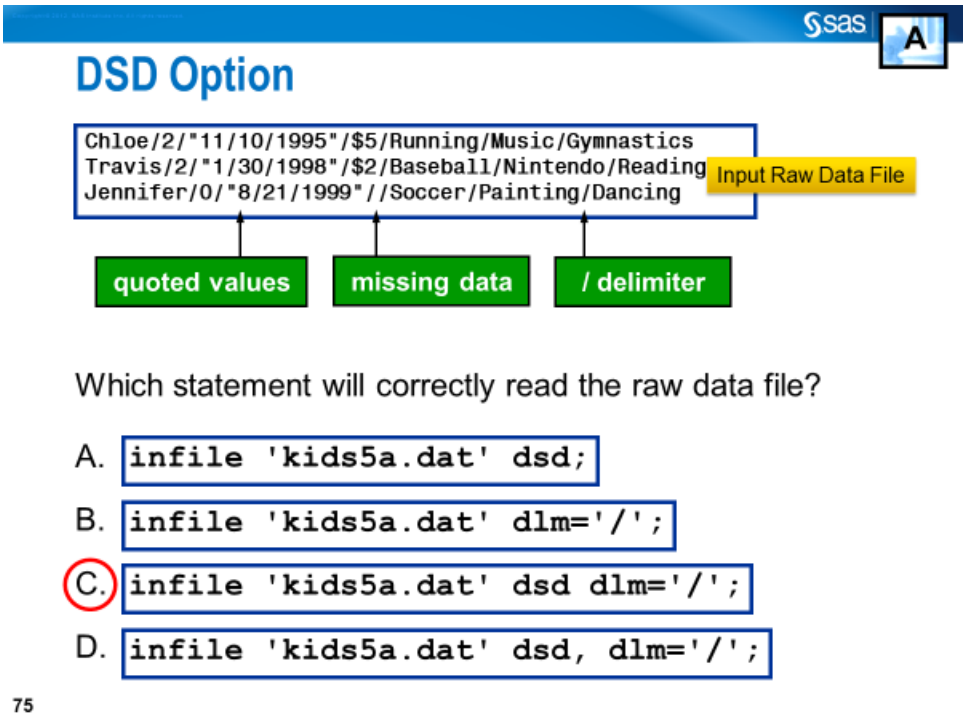


Modified List Input and DLM= Option on the INFILE Statement



Modified List Input and DSD Option on the Infile Statement





Missing Values at the End of a Record

The raw data file **phone.csv** contains missing values at the end of some records.

phone.csv

```
          1    1    2                    4    4
1---5----0----5----0----5----0----5----0----5
James Kvarniq,(704) 293-8126,(701) 281-8923
Sandrina Stephano,(919) 871-7830
Cornelia Krahl,(212) 891-3241,(212) 233-5413
Karen Ballinger,(714) 344-4321
Elke Wallstab,(910) 763-5561,(910) 545-3421
```

missing values

The DSD option is not appropriate because the missing data is not marked by consecutive delimiters.

97

MISSOVER Option

The *MISSOVER option* prevents SAS from loading a new record when the end of the current record is reached.

```
data contacts;
   length Name $ 20 Phone Mobile $ 14;
   infile "&path\phone.csv" dlm=',' missover;
   input Name $ Phone $ Mobile $;
run;

proc print data=contacts noobs;
run;
```

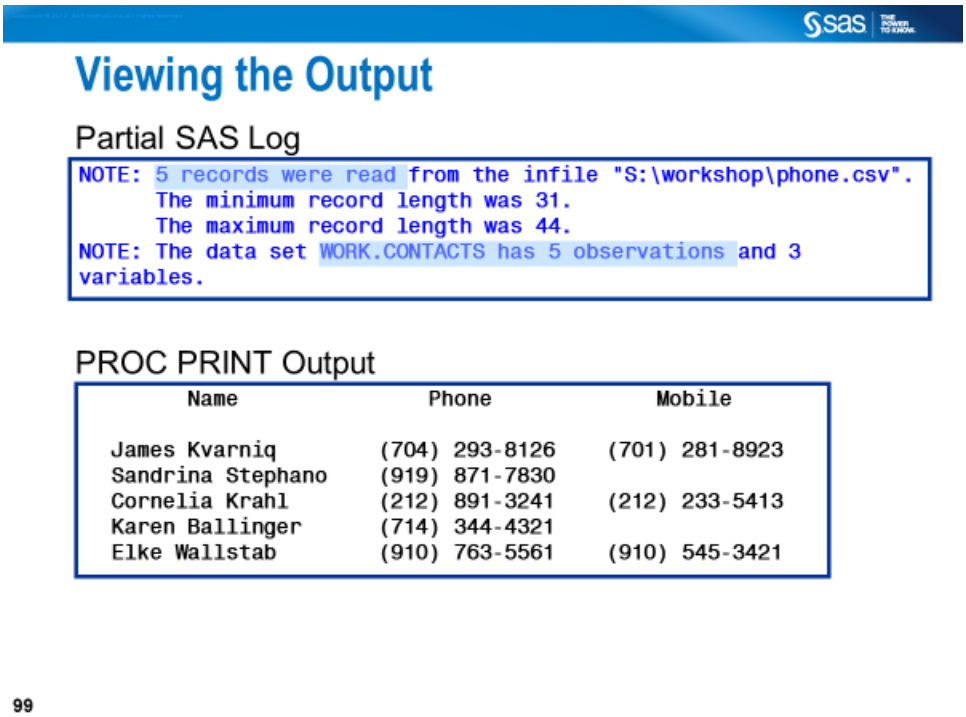
INFILE "*raw-data-file*" <DLM=> MISSOVER;

If SAS reaches the end of a record without finding values for all fields, variables without values are set to missing.

p108d10

98

Use MISSOVER if the last field or fields might be missing and you want SAS to assign missing values to the corresponding variable.



The Ampersand (&) Modifier

The ampersand (&) format modifier enables you to read character values that contains one or more embedded blanks with list input and to specify a character informat. SAS reads until it encounters **two consecutive blanks**, the defined length of the variable, or the end of the input line, whichever comes first.

```
*Ex6_Formated_Input_Dates.sas (Part 7);
options yearcutoff=1720;
data yc;
   INPUT state_name  & $22. date_entry :mmddyy.;
   FORMAT date_entry :mmddyy10.;
DATALINES;
Delaware  12/07/87
Pennsylvania  12/12/87
New Jersey  12/18/87
South Carolina  05/23/88
;
proc print data=yc noobs;
run;
```

Multiple INPUT Statements

By default, SAS advances the pointer to column 1 of the next input record when SAS encounters an INPUT statement.

Input Raw Data File

```
Chloe    2
11/10/1995 $5
Running Music Gymnastics
Travis   2
1/30/1998  $2
Baseball Nintendo Reading
Jennifer 0
8/21/1999  $0
Soccer Painting Dancing
```

```
data work.kids8;
   infile 'kids8.dat';
   input name $ 1-8
         siblings 10;
   input @1 bdate mmddyy10.
         @12 allowance comma2.;
   input hobby1:$10.
         hobby2:$10.
         hobby3:$10.;
run;
```

82

Line-Pointer Controls

The / line-pointer control advances the pointer to column 1 of the next input record.

```
data work.kids8;
   infile 'kids8.dat';
   input name $ 1-8
         siblings 10
       / @1 bdate mmddyy10.
         @12 allowance comma2.
       / hobby1:$10.
         hobby2:$10.
         hobby3:$10.;
run;
```

86

Line-Pointer Controls

The #*n* line-pointer control advances the pointer to column 1 of record *n*.

```
data work.kids8;
   infile 'kids8.dat';
   input #1 name $ 1-8
            siblings 10
         #2 @1 bdate mmddyy10.
            @12 allowance comma2.
         #3 hobby1:$10.
            hobby2:$10.
            hobby3:$10.;
run;
```

87

Line-Pointer Controls

The second record is skipped in each iteration of the DATA step.

```
input name $ 1-8 siblings 10 //
      hobby1:$10. hobby2:$10. hobby3:$10.;
```

```
input #1 name $ 1-8 siblings 10
      #3 hobby1:$10. hobby2:$10. hobby3:$10.;
```

```
input name $ 1-8 siblings 10;
input;
input hobby1:$10. hobby2:$10. hobby3:$10.;
```

Output Data Set

VIEWTABLE: Work.Kids8

	name	siblings	hobby1	hobby2	hobby3
1	Chloe	2	Running	Music	Gymnastics
2	Travis	2	Baseball	Nintendo	Reading
3	Jennifer	0	Soccer	Painting	Dancing

90

6. Which one of the following INPUT statements correctly reads the values for **Fname** (in the second field), **Lname**, **Department**, and **Salary** (in that order)?

   a. input @14 Fname $10. @1 Lname $10./
      Department $7./
      Salary comma10.;
   b. input @14 Fname $10.
      @1 Lname $10.
      #2 Department $7.
      #3 Salary comma10.;
   c. both a and b

```
          1    1    2    2
1---5----0----5----0----5
ABRAMS         THOMAS
SALES
$45,209.03
BARCLAY        ROBERT
ACCTING
$49,180.36
COURTNEY        MARK
EDUC
$44,006.16
```

You can use the @*n* column pointer control to read the values for Lname and Fname. You can use either the / or #*n* line pointer control to advance the input pointer to the second and third lines of the raw data file to read the values for Department and Salary.

94

When to Use the Forward Slash (/) Line Pointer Control

You have data elements that need to be read sequentially from multiple records to create a single observation.

Example: Mix of Column Input, Formatted Input, and List Input Styles

INPUT Statement

```
----|----10---|----20---|----30---|----40---|----50
Chloe    2 11/10/1995 $5Running Music Gymnastics
Travis   2 1/30/1998  $2Baseball Nintendo Reading
Jennifer 0 8/21/1999  $0Soccer Painting Dancing
```

Input Raw Data File

The following three ways can describe a record's values in the INPUT statement:

- column input
- formatted input
- list input

```
input name $ 1-8 siblings 10
      @12 bdate mmddyy10.
      @23 allowance comma2.
      hobby1 $ hobby2 $ hobby3 $;
```

column

formatted

list

11

Column Input

With column input, the column numbers that contain the value follow a variable name in the INPUT statement.

```
input name $ 1-8 siblings 10
      @12 bdate mmddyy10.
      @23 allowance comma2.
      hobby1 $ hobby2 $ hobby3 $;
```

To read with column input, data values

- must be in the same columns in all the input data records
- must be in standard form.

12

Formatted Input

With formatted input, an informat follows a variable name and defines how SAS reads the values of this variable. An informat gives the data type and the field width of an input value.

```
input name $ 1-8 siblings 10
      @12 bdate mmddyy10.
      @23 allowance comma2.
      hobby1 $ hobby2 $ hobby3 $;
```

To read with formatted input, data values

- must be in the same columns in all the input data records
- can be in standard or nonstandard form.

13

List Input

With list input, variable names in the INPUT statement are specified in the same order that the fields appear in the input data records.

```
input name $ 1-8 siblings 10
      @12 bdate mmddyy10.
      @23 allowance comma2.
      hobby1 $ hobby2 $ hobby3 $;
```

To read with list input, data values

- must be separated with a delimiter
- can be in standard or nonstandard form.

14

The Use of Multiple Input Statements

```
 1 *Ex29_Multiple_Input_Statements.sas;
 2 data work.HAVE(drop=i);
 3  input date: Anydtdte9. @;
 4  do i = 1 to 4;
 5  input name $ study_hours @;
 6  output;
 7  end;
 8 datalines;
 9 27Aug2018 Doris 5.5 Alice 4.0 Mike 2.0 James 1.0
10 28Jun2018 Doris 3.0 Alice 3.0 Mike 3.0 James 1.0
11 ;
12 proc print data=work.HAVE;
13 Format date worddate.;
14 run;
```

Obs	date	name	study_hours
1	August 27, 2018	Doris	5.5
2	August 27, 2018	Alice	4.0
3	August 27, 2018	Mike	2.0
4	August 27, 2018	James	1.0
5	June 28, 2018	Doris	3.0
6	June 28, 2018	Alice	3.0
7	June 28, 2018	Mike	3.0
8	June 28, 2018	James	1.0

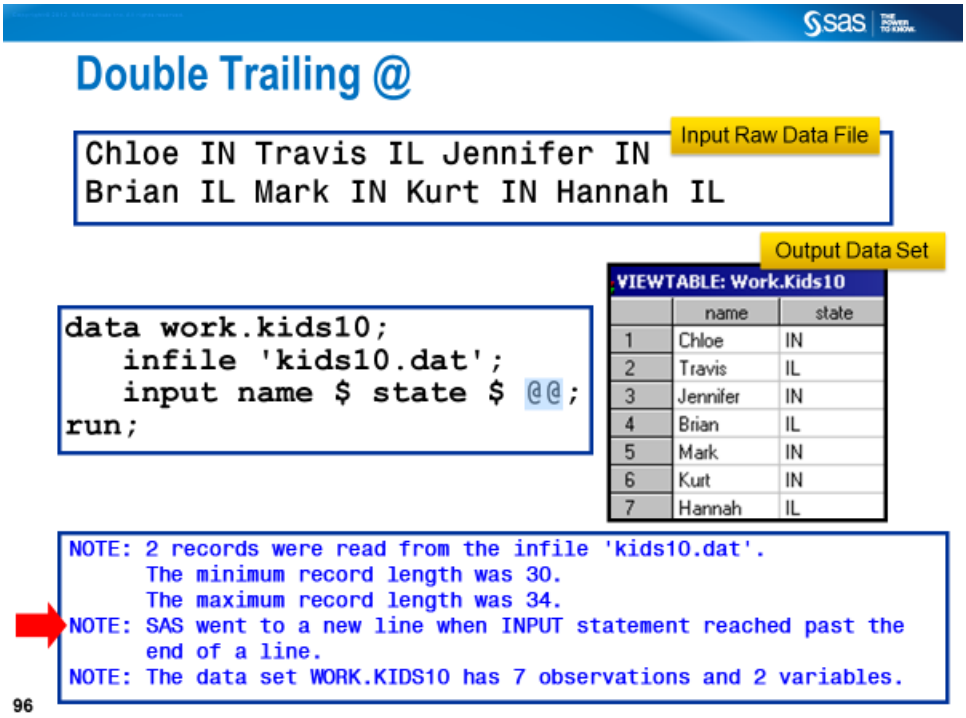
Normally, each INPUT statement in a DATA step reads a new data record into the input buffer. However, when you use a trailing @, the following things occur:

- The pointer position does not change.
- No new record is read into the input buffer.
- The next INPUT statement for the same iteration of the DATA step continues to read the same record rather than a new one.

SAS releases a record held by a trailing @ when

- a null INPUT statement executes (**input;**)
- an INPUT statement without a trailing @ executes
- the next iteration of the DATA step begins.

List Input with a Double Trailing @ at the End of the Input Statement



Modified List Input with a double @ at the End of the Input Statement

```
 1 *Ex30_@@.sas;
 2 data work.HAVE;
 3  input date: Anydtdte. name $ study_hours @@;
 4 datalines;
 5 27Aug2018 Doris 5.5 28Aug2018 Alice 4.0
 6 29Aug2018 Mike 2.0 29Aug2018 James 1.0
 7 30Jun2018 Doris 3.0 31Aug2018 Alice 3.0
 8 01Sep2018 Mike 3.0
 9 02Sep2018 James 1.0
10 ;
11 proc print data=work.HAVE;
12 Format date mmddyy10.;
13 run;
```

In the above program, we are reading 5 lines of data with the **Datalines** statement. Lines 5, 6, and 7 each have have two records; and lines 8 and 9 each have one record. Here, our goal is to create a SAS data file that will have one observation from each record – a total of 8 observations.

Notice that the Informat for the variable **date** does not specify a *w* value because we are reading data using List Input with a colon modifier; here, SAS reads data fields until it encounters a blank.

Obs	date	name	study_ hours
1	08/27/2018	Doris	5.5
2	08/28/2018	Alice	4.0
3	08/29/2018	Mike	2.0
4	08/29/2018	James	1.0
5	06/30/2018	Doris	3.0
6	08/31/2018	Alice	3.0
7	09/01/2018	Mike	3.0
8	09/02/2018	James	1.0

Placement of the Subsetting IF Statement

Generally, the most efficient place to put the subsetting IF statement is as soon as all the variables that are needed to evaluate the condition are assigned values.

```
data EuropeQ1;
   infile "&path\sales.dat";
   input @6 Location $3. @;
   if Location = 'EUR';
   input  @1 SaleID $4.
          @10 SaleDate date9.
          @20 Amount commax7.;
run;
```

78 p204d07

SAS releases a record held by a trailing @ when the next iteration of the DATA step begins.

Subsetting Mixed Record Types: Output

```
proc print data=EuropeQ1 noobs;
   var SaleID Location SaleDate Amount;
run;
```

PROC PRINT Output

Sale ID	Location	Sale Date	Amount
3034	EUR	18657	1876.3
1345	EUR	18664	3145.6

79

When to Use Named Input

- Data records contain both variable names and values

What to Specify on the Input Statement (Named Input)

- Each variable name must be followed by an equal sign on the INPUT statement
- Each variable name must be followed by an equal sign on the data as well
- Character variables must be indicated by a "$" following the equals sign on the INPUT statement
- Appropriate format modifiers and informats must be specified
- The"/" at the end of the line must be used to read the next data line in order to complete the observation

```
 1  *Ex16_Named_Input.sas;
 2  options nocenter nodate nonumber ls=132;
 3  DATA TEST;
 4  input name = & $ 30. address = & $ 30.
 5        city_zip  = & $ 30. phone= $ 14.
 6        Num_employees = ;
 7        FORMAT Num_employees comma7.;
 8  DATALINES;
 9  name=Air Force Personnel Center /
10  address=550 C Street West /
11  city_zip=Randolph AFB, TX 78150 /
12  phone=1-800-525-0102 /
13  Num_employees=5876
14  name= Navy Personnel Command /
15  address= 5720 Integrity Drive /
16  city_zip= Millington, TN 38055 /
17  phone= 901-874-4885 /
18  Num_employees=3987
19  ;
20  proc print noobs; run;
```

name	address	city_zip	phone	Num_ employees
Air Force Personnel Center	550 C Street West	Randolph AFB, TX 78150	1-800-525-0102	5,876
Navy Personnel Command	5720 Integrity Drive	Millington, TN 38055	901-874-4885	3,987

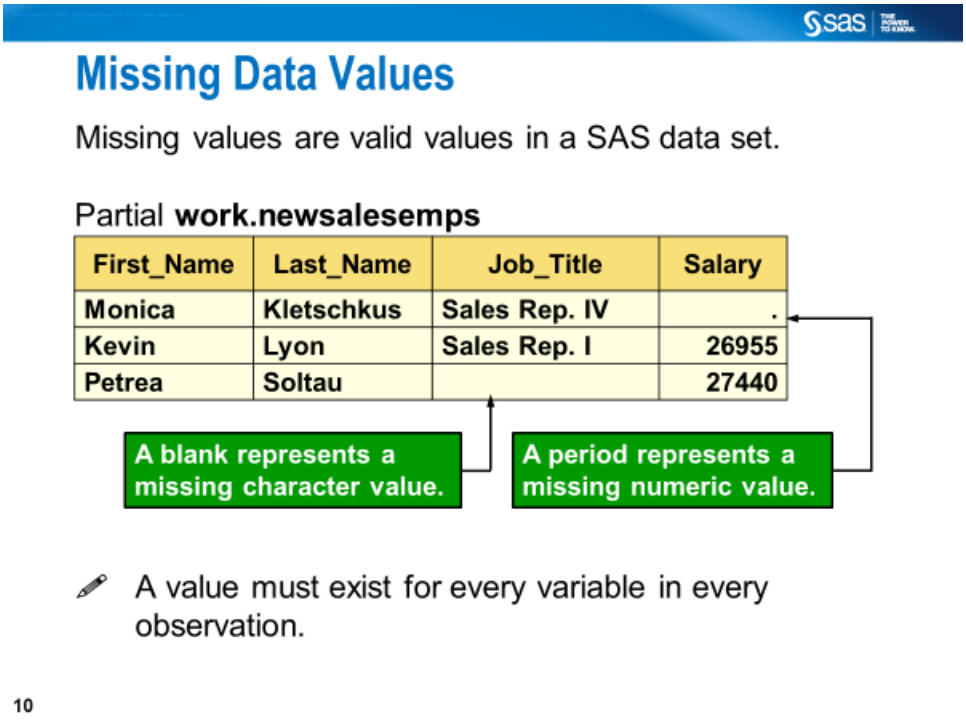
How to represent missing values

When you have a space delimited file and you have a missing value, you have to represent the missing value as a period when reading data using list or modified list input.

When you have a comma delimited file or any type of delimited file, you do not have to have a period as a missing value because you normally have multiple delimiters in a row to indicate a missing value.

When you have space delimited file that is column aligned, then you do not have to have a period to represent a missing value because you are using column input to read the data.

Handling Missing Values



In addition to the above two ordinary missing values, there are 26 special missing values that only apply to the numeric variable.

MISSING= System Option

The MISSING= system option allows to specify a character to be printed when numeric variables contain ordinary missing values (.).

MISSING Statement

If your data contain characters that represent special missing values, such as a or z, do not use the MISSING= system option to define them; simply define these values in a MISSING statement.

Reading Microsoft Excel Data into SAS Data Sets

SAS/ACCESS LIBNAME Statement

The default Excel engine can be specified when the bit count of SAS and Microsoft Office are the same (both are 32-bit or both are 64-bit).

```
libname myxls excel 'customers.xls';
```

The PC Files Server engine must be specified when the bit counts differ.

```
libname myxls pcfiles path='customers.xls';
```

110

XLS Files

Prior to Office 2007, XLS was the default format for Excel to store data in worksheets, charts, and macros.

The XLS format is

- a proprietary binary file format called *Binary Interchange File Format* (BIFF)
- readable by all Microsoft Excel versions.

35

XLSX Files

With the release of Microsoft Office 2007, Microsoft changed the default file format. To facilitate this new format, Microsoft added an additional X to their document extensions. For Excel, this is XLSX.

The XLSX file format

- is readable only by versions 2007 and later
- does not support Excel macros.

36

Excel File Structure Limitation

File Type	Row Limit	Column Limit
XLS	65,536	255
XLSX	1,048,576	16,384

38

Creating Raw Tabular Data Files or Reports Using DATA _NULL_

PUT and PUTLOG Statements

The PUT statement writes to the LOG or to an External File with a FILE statement, but the PUTLOG statement always writes to the LOG.

```
1  *Ex21_put.sas;
2  * List all DATA step variables and their values;
3  data _null_;
4     set sashelp.class(obs=2);
5     put _all_;
6  run;
```

Line 3: The keyword _NULL_ on the DATA statement is used to execute the data step without creating a data set.

Line 5: The PUT statement is used to create output records to the LOG window. The special SAS name list _ALL_ refers to all variables on the data step and program data vector including _N_ and _ERROR_.

```
Name=Alfred Sex=M Age=14 Height=69 Weight=112.5 _ERROR_=0 _N_=1
Name=Alice Sex=F Age=13 Height=56.5 Weight=84 _ERROR_=0 _N_=2
NOTE: There were 2 observations read from the data set SASHELP.CLASS.
```

```
 7  * Exclude automatic variables;
 8  data _null_;
 9     set sashelp.class(obs=2);
10     put (_all_)(=);
11  run;
```

Line 21: In the PUT statement, the variable list argument is _ALL_ [i.e., all variables on the data step and program data vector excluding _N_ and _ERROR_]; the format argument is the equal sign so that the variable name precedes its value.

```
Name=Alfred Sex=M Age=14 Height=69 Weight=112.5
Name=Alice Sex=F Age=13 Height=56.5 Weight=84
NOTE: There were 2 observations read from the data set SASHELP.CLASS.
```

```
17 /* Put each value on a new line and apply
18 a common format to all numeric variables*/
19 data _null_;
20    set sashelp.class(obs=2);
21    put (_all_)(=/12.2);
22 run;
```

Line 21: The PUT statement is used to tell SAS to write each of the variables in the data step and program data vector (except _N_ and _ERROR_) that must precede its value. An additional format argument / is used so that each variable and its value separated by an equal sign is output to a separate line. Note the **FORMAT***w.d* (i.e., 12.2) that applies to all numeric variables.

```
Name=Alfred
Sex=M
Age=14.00
Height=69.00
Weight=112.50

Name=Alice
Sex=F
Age=13.00
Height=56.50
Weight=84.00
NOTE: There were 2 observations read from the data set SASHELP.CLASS.
```

```
23 /* List values as a table and apply formats
24 to groups of variables*/
25 data _null_;
26   set sashelp.class(obs=2);
27   if _n_=1 then put @1 'NAME' @19 'SEX' @23 'AGE'
28                     @30 'HEIGHT' @38 'WEIGHT';
29   put (_all_)(1*$20.,1*$2.,1*3.,2*8.2);
30 run;
```

Line 27: During the first iteration, the variable names are printed each starting with the column position specified, to the LOG window by default.

Line 29: This is an example of the FORMATTED PUT statement. There is no equal sign or slash as a format-list argument. Formats specified as format-list arguments. For example, a character format **$20.** is applied to the variable NAME. Another character format **$2.** is applied to the variable SEX. A numeric format **3.** is applied to the variable AGE. Another numeric format 8.2 is applied to the variables HEIGHT and WEIGHT.

```
NAME                 SEX AGE    HEIGHT  WEIGHT
Alfred                 M  14    69.00  112.50
Alice                  F  13    56.50   84.00
NOTE: There were 2 observations read from the data set SASHELP.CLASS.
```

```
33 /* List values as a table and apply formats
34 to groups of variables. Route output to the
35 standard SAS output window. */
36 options nodate nonumber;
37 title;
38 data _null_;
39   set sashelp.class(obs=2);
40   file print;
41   if _n_=1 then put @1 'NAME' @19 'SEX' @23 'AGE'
42                     @30 'HEIGHT' @38 'WEIGHT';
43   put (_all_)(1*$20.,1*$2.,1*3.,2*8.2);
44 run;
```

Line 40: This statement creates the tabular output to the OUTPUT window, not the LOG window.

(Partial SAS Log)

```
NOTE: 3 lines were written to file PRINT.
```

Snapshot from the SAS output window

```
NAME                SEX AGE    HEIGHT  WEIGHT
Alfred                M  14   69.00  112.50
Alice                 F  13   56.50   84.00
```

The PUT statement in a DATA step always writes to the SAS log, unless it is preceded by a FILE statement that specifies otherwise. [SAS® Documentation]

```
46 /* List values as a table and apply formats
47 to groups of variables. Route output to a file */
48 options nodate nonumber;
49 title;
50 data _null_;
51    set sashelp.class(obs=2);
52    file "E:\class_data.txt";
53    if _n_=1 then put @1 'NAME' @19 'SEX' @23 'AGE'
54                      @30 'HEIGHT' @38 'WEIGHT';
55    put (_all_)(1*$20.,1*$2.,1*3.,2*8.2);
56 run;
```

Line 52: This statement creates a tabular raw data file named CLASS_DATA.TXT in the path specified, all enclosed in quotation mark.

```
NOTE: The file "E:\class_data.txt" is:
      Filename=E:\class_data.txt,
      RECFM=V,LRECL=256,File Size (bytes)=0,
      Last Modified=20Aug2016:10:44:00,
      Create Time=20Aug2016:10:42:55

NOTE: 3 records were written to the file "E:\class_data.txt".
      The minimum record length was 41.
      The maximum record length was 43.
NOTE: There were 2 observations read from the data set SASHELP.CLASS.
```

Read/watch this webinar: Top 10 Ways to Optimize Your SAS Code by Jeff Simpson

Automatic variable

INFILE specifies a character variable that references the contents of the current input buffer for this INFILE statement. You can use the variable in the same way as any other variable, even as the target of an assignment. The variable is automatically retained and initialized to blanks. As with automatic variables, the _INFILE_= variable is not written to the data set.

```
1 Data _Null_;
2  input name $ software $;
3  put _infile_;
4  datalines;
5  Mike R
6  John SAS
7  Rachel Python
8  ;
9  run;
```

```
Mike R
John SAS
Rachel Python
```

_N_ and _ERROR_

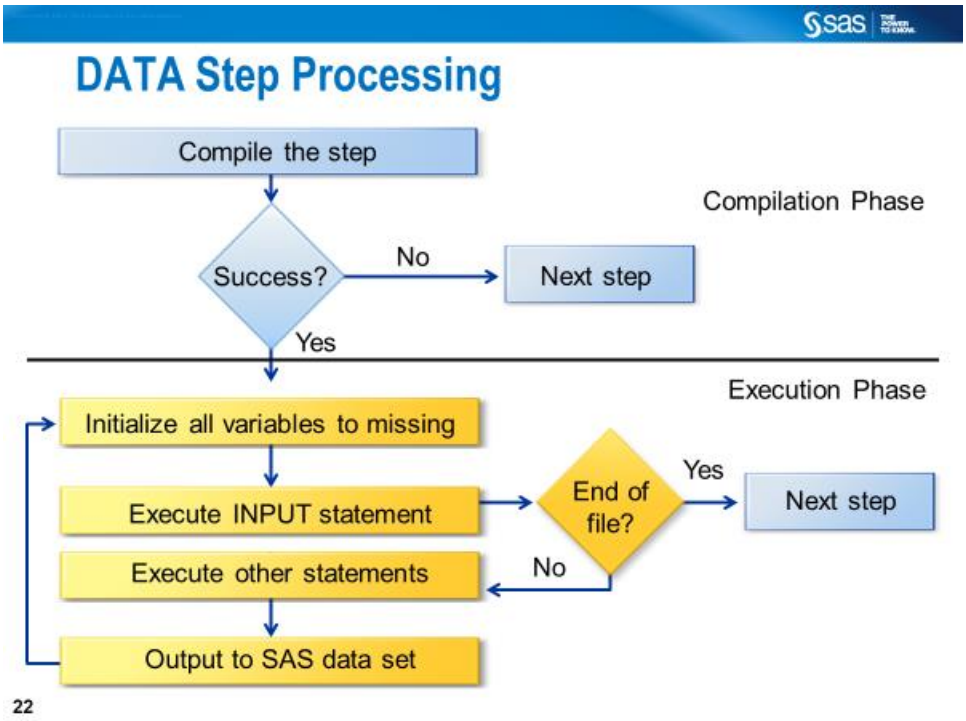
These two variables are created by SAS engine during compilation. These variables are used for processing but never written into the SAS data set created.

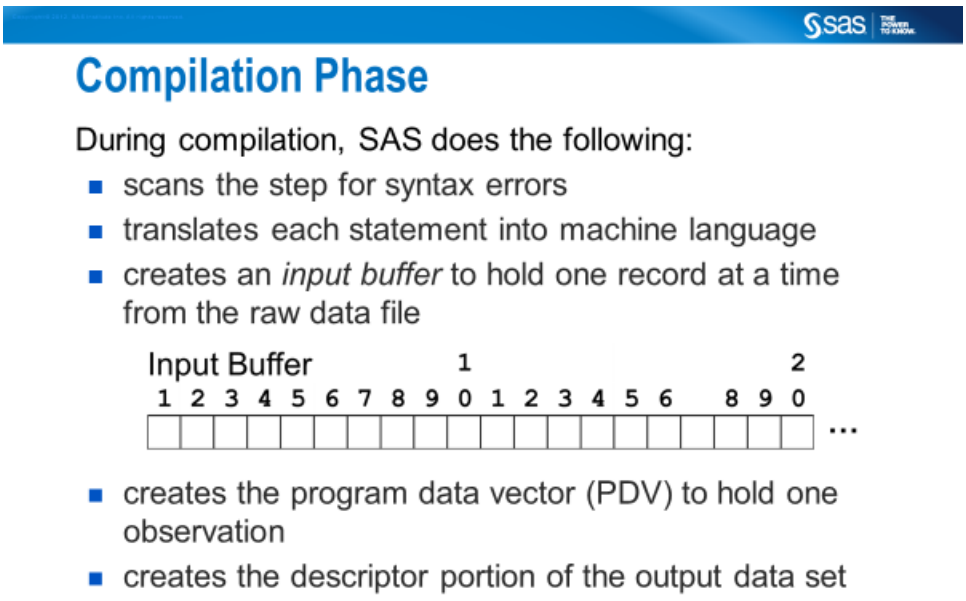
_N_ is initially set to 1. Each time the DATA step loops past the DATA statement, the variable _N_ increments by 1. _N_ =1 means that the program left the DATA statement for the first time. If the data set has set 5 observations, it will leave the DATA statement six times.

ERROR is 0 by default but is set to 1 whenever an error is encountered, such as an input data error, a conversion error, or a math error, as in division by 0 or a floating point overflow. You can use the value of this variable to help locate errors in data records and to print an error message to the SAS log.

Source: SAS Documentation

Compilation vs. Execution





Compilation: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Input Buffer

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8

19 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Initialize PDV

Input Buffer

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
.	.		.

20 ...

Execution: Formatted Input

Specify input data file

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
.	.		.

21 ...

Execution: Formatted Input

Load input buffer

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 0 4 0 1 2 / 0 2 / 1 1   O u t d o o r s 1 5 %
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
.	.		.

22 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Load first value into PDV

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 0 4 0 1 2 / 0 2 / 1 1   O u t d o o r s 1 5 %
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
1040	.		.

23 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Load second value into PDV

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 0 4 0 1 2 / 0 2 / 1 1   O u t d o o r s 1 5 %
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
1040	18963		.

24 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Load third value into PDV

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 0 4 0 1 2 / 0 2 / 1 1   O u t d o o r s 1 5 %
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
1040	18963	Outdoors	.

25 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Load fourth value into PDV

Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 0 4 0 1 2 / 0 2 / 1 1   O u t d o o r s 1 5 %
```

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
1040	18963	Outdoors	.15

26 ...

Execution: Formatted Input

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Implicit OUTPUT;
Implicit RETURN;

Input Buffer

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
1	0	4	0	1	2	/	0	2	/	1	1		O	u	t	d	o	o	r	s	1	5	%	

PDV

Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
1040	18963	Outdoors	.15

27 ...

Execution: Formatted Input

Reinitialize PDV

```
data work.discounts;
   infile "&path\offers.dat";
   input @1 Cust_type 4.
         @5 Offer_dt mmddyy8.
         @14 Item_gp $8.
         @22 Discount percent3.;
run;
```

Continue until EOF

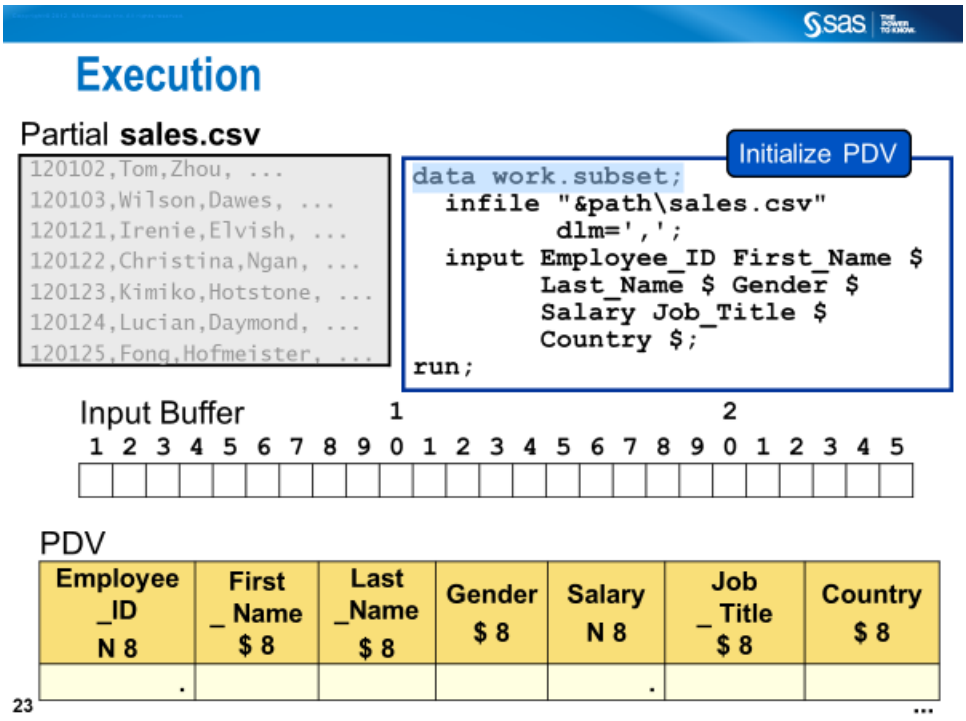
Input Buffer

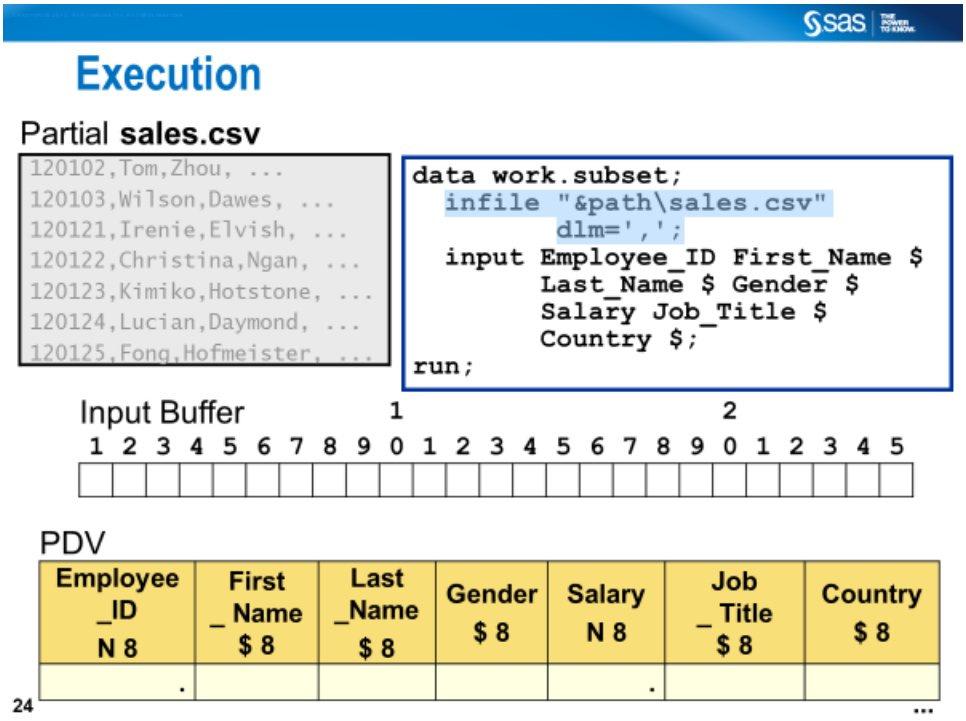
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
1	0	4	0	1	2	/	0	2	/	1	1		O	u	t	d	o	o	r	s	1	5	%	

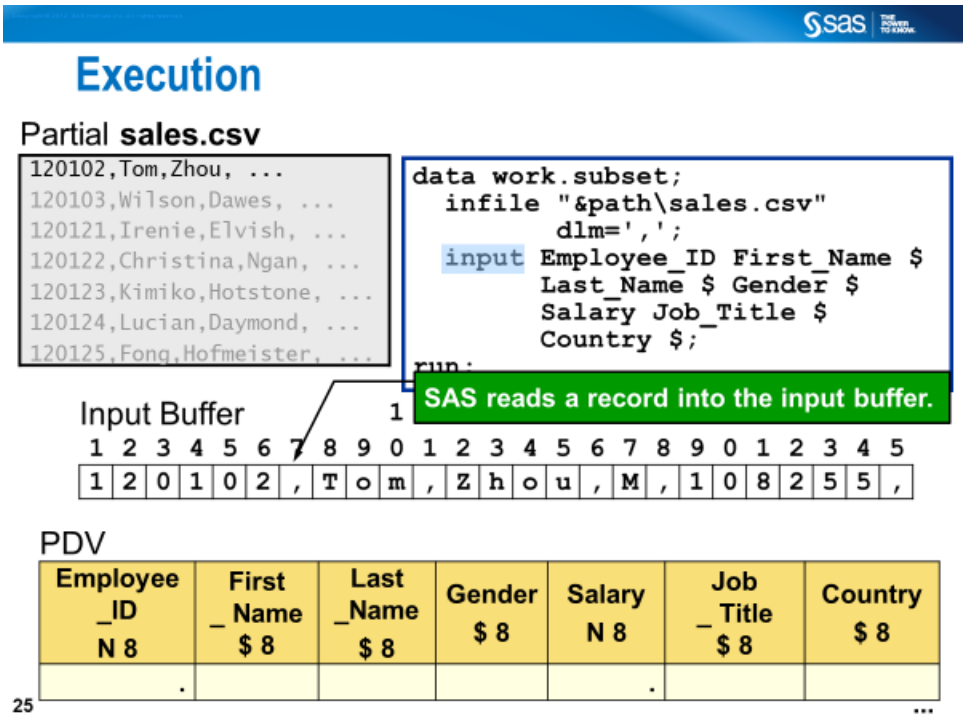
PDV

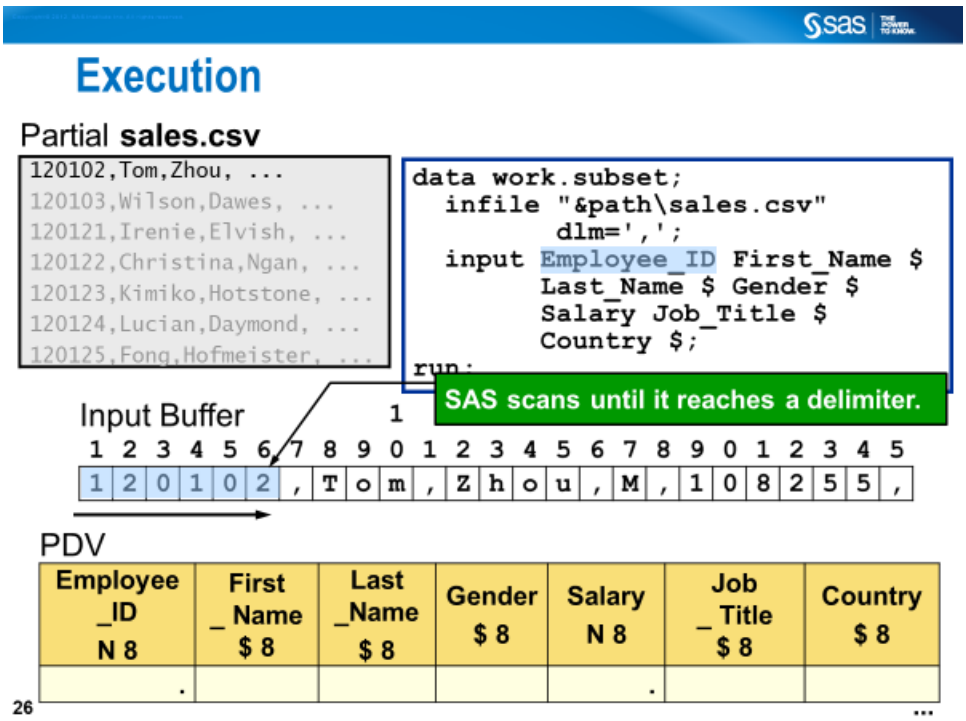
Cust_type N 8	Offer_dt N 8	Item_gp $ 8	Discount N 8
.	.		.

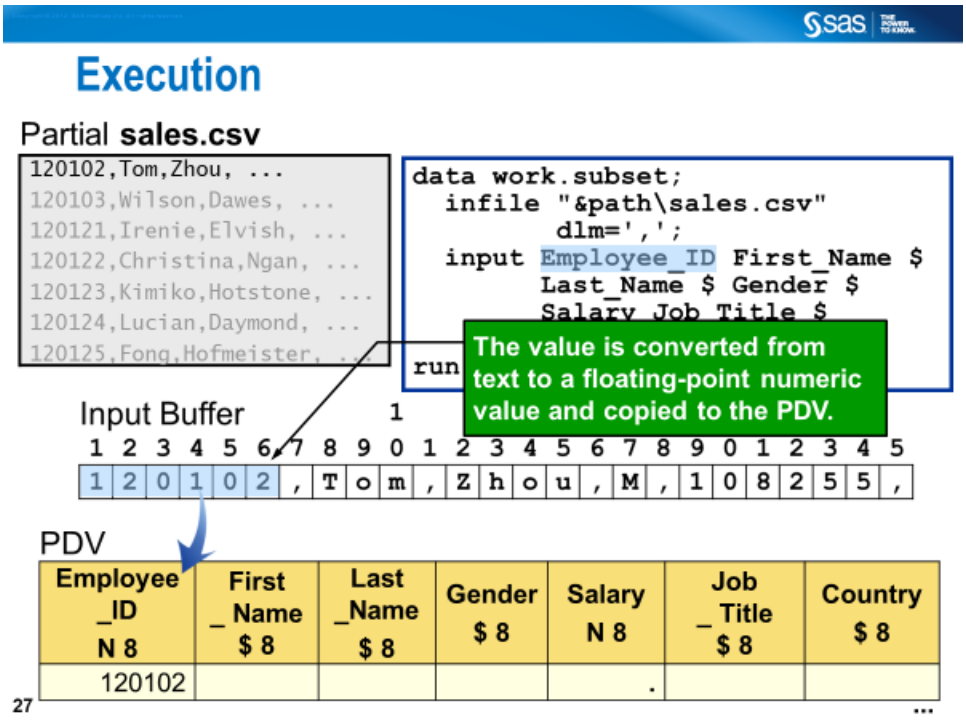
28

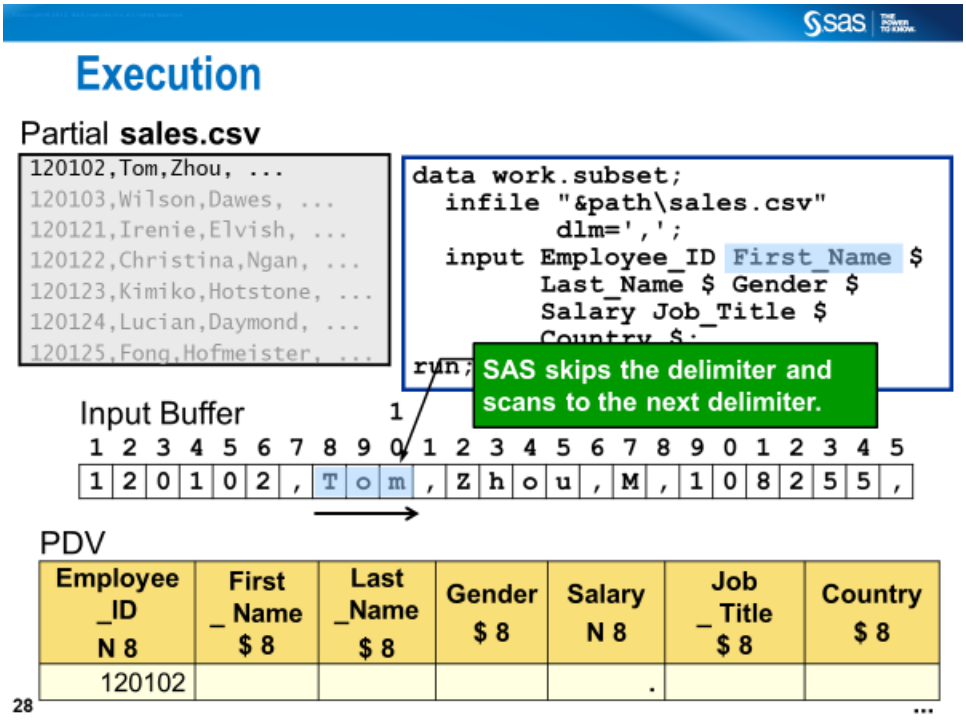


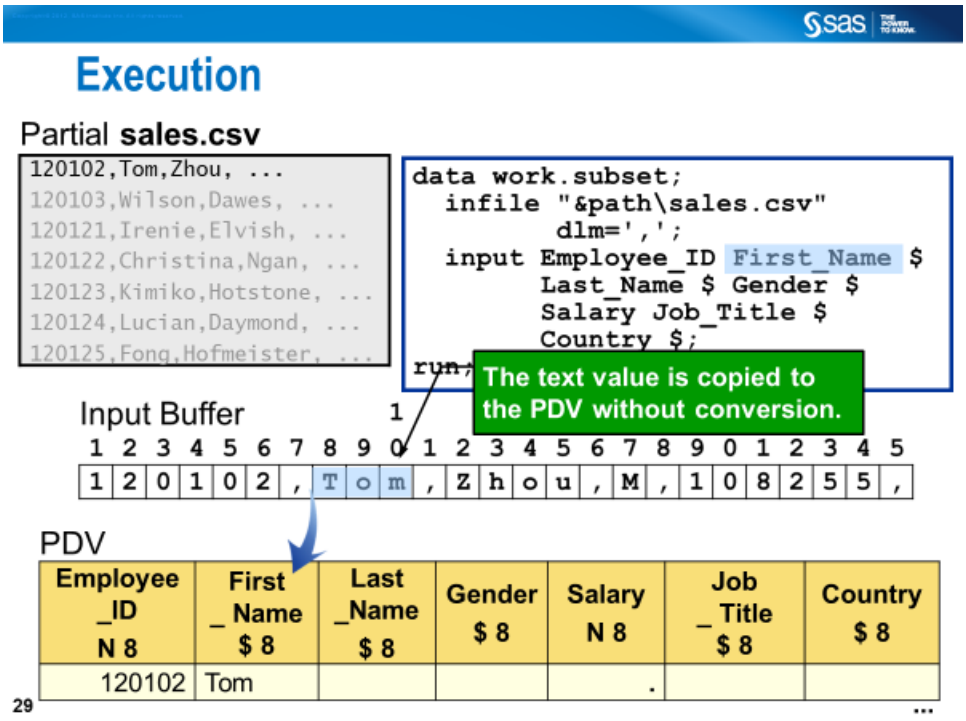


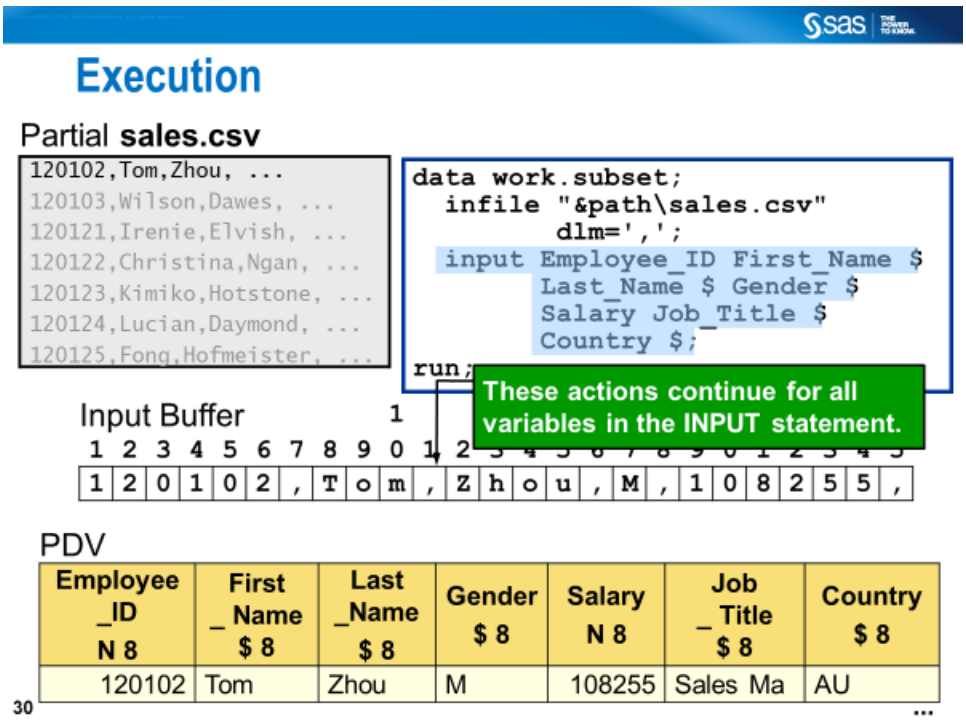












Execution

Partial **sales.csv**

```
120102,Tom,Zhou, ...
120103,Wilson,Dawes, ...
120121,Irenie,Elvish, ...
120122,Christina,Ngan, ...
120123,Kimiko,Hotstone, ...
120124,Lucian,Daymond, ...
120125,Fong,Hofmeister, ...
```

```
data work.subset;
  infile "&path\sales.csv"
         dlm=',';
  input Employee_ID First_Name $
        Last_Name $ Gender $
        Salary Job_Title $
        Country $;
run;
```

Implicit OUTPUT;
Implicit RETURN;

Input Buffer

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1 2 0 1 0 2 , T o m , Z h o u , M , 1 0 8 2 5 5 ,
```

PDV

Employee _ID N 8	First _ Name $ 8	Last _Name $ 8	Gender $ 8	Salary N 8	Job _ Title $ 8	Country $ 8
120102	Tom	Zhou	M	108255	Sales Ma	AU

31 ...

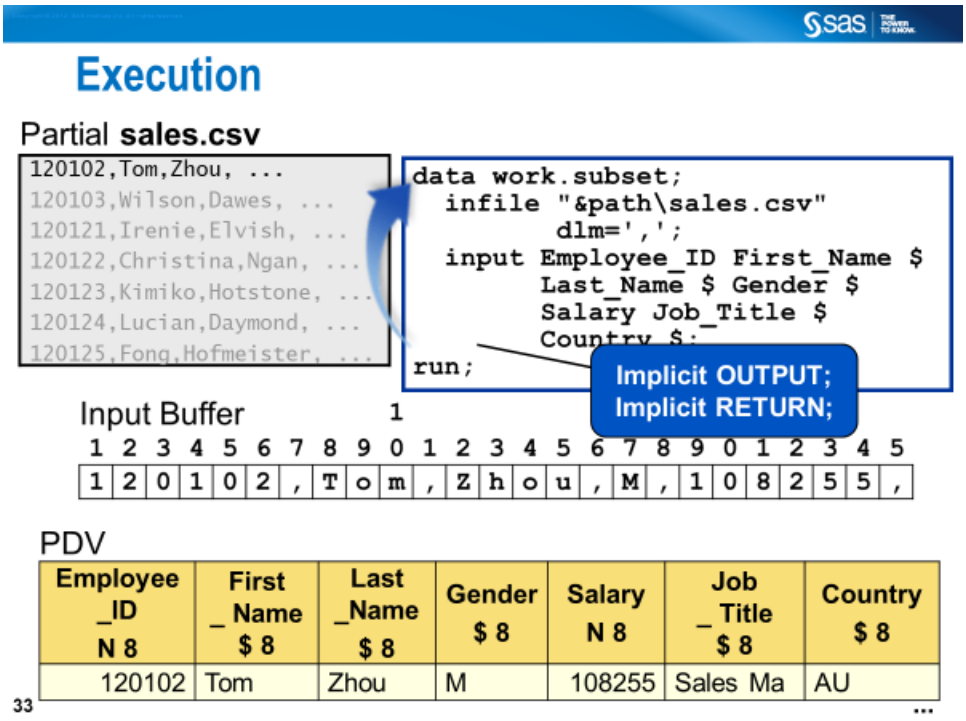
Execution

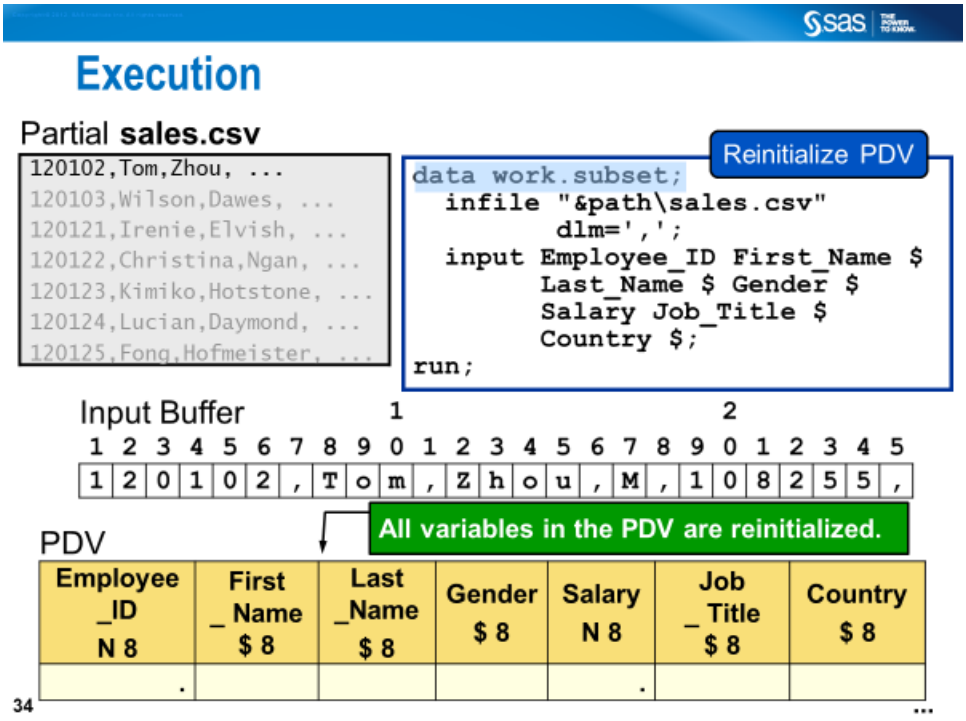
Here is the output data set after the first iteration of the DATA step.

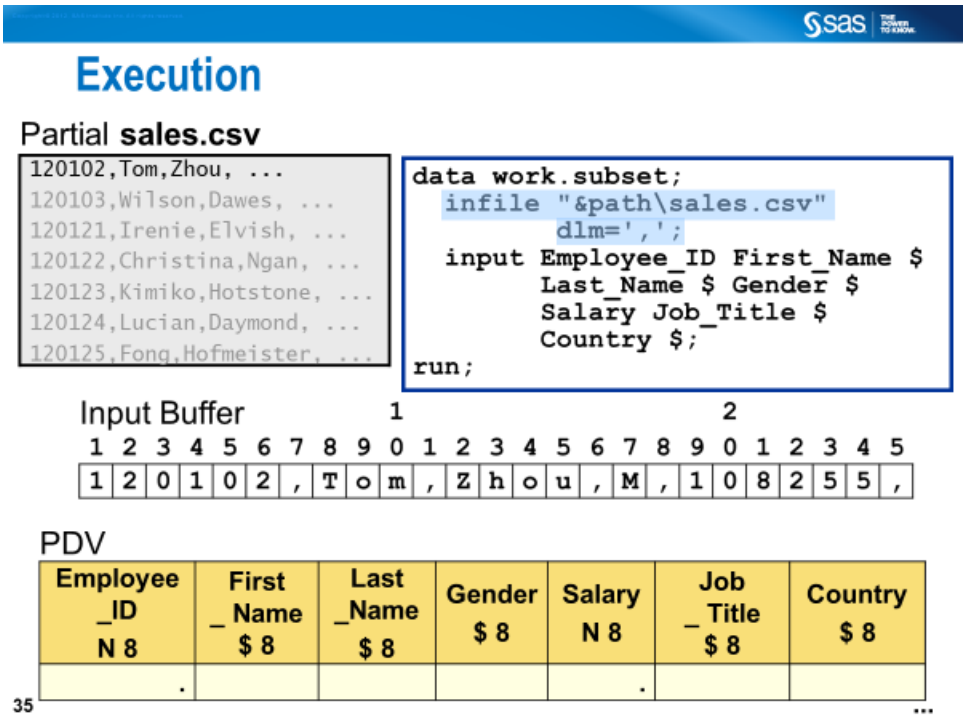
work.subset

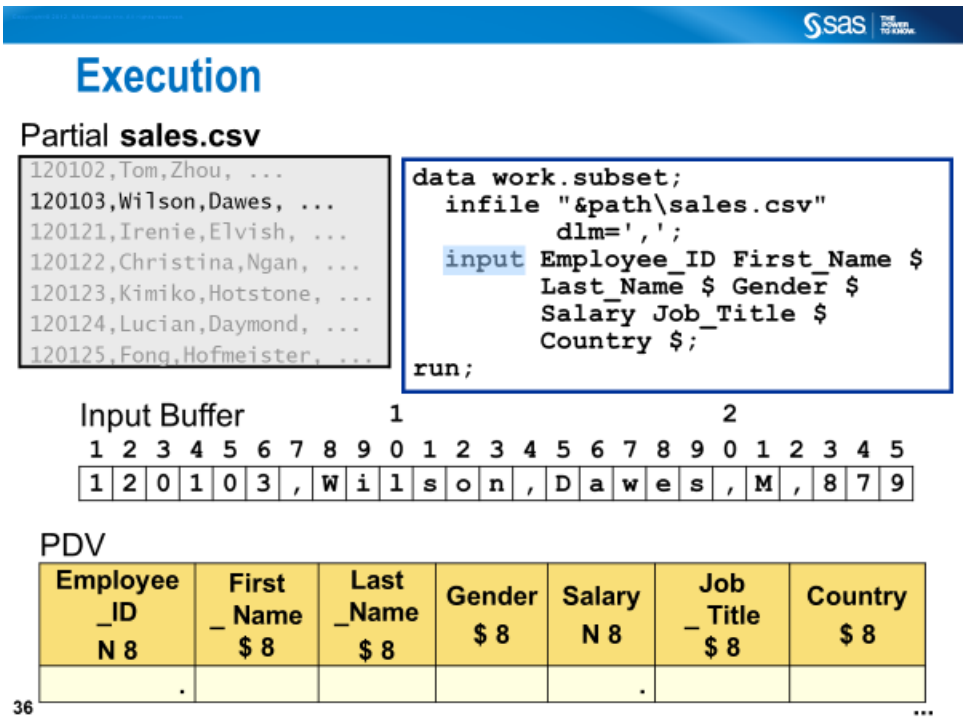
Employee _ID	First _Name	Last _Name	Gender	Salary	Job _Title	Country
120102	Tom	Zhou	M	108255	Sales Ma	AU

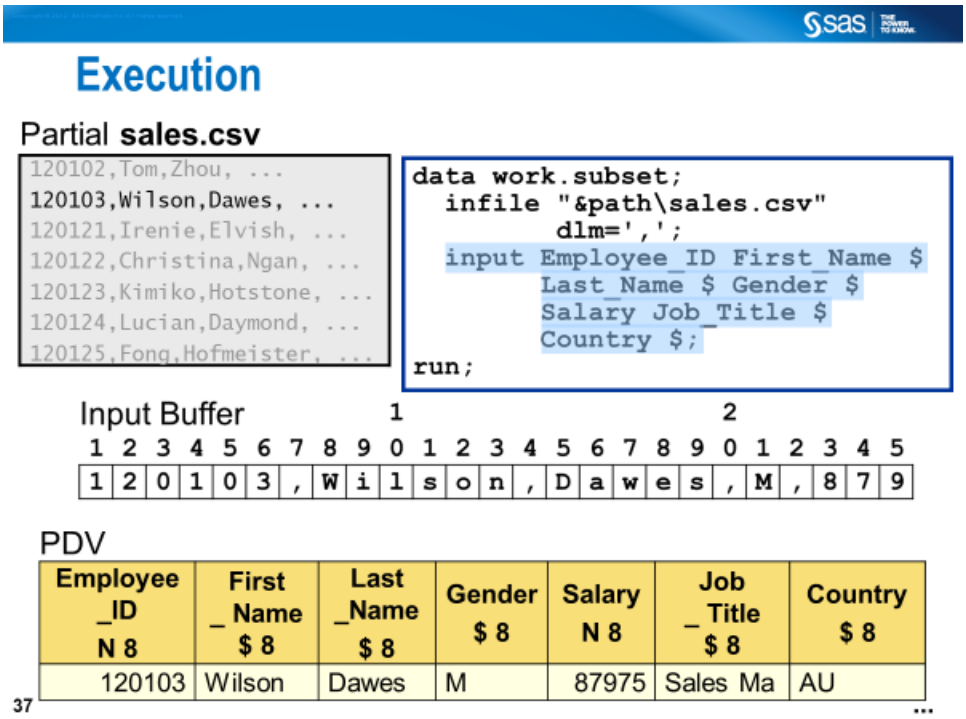
32 ...

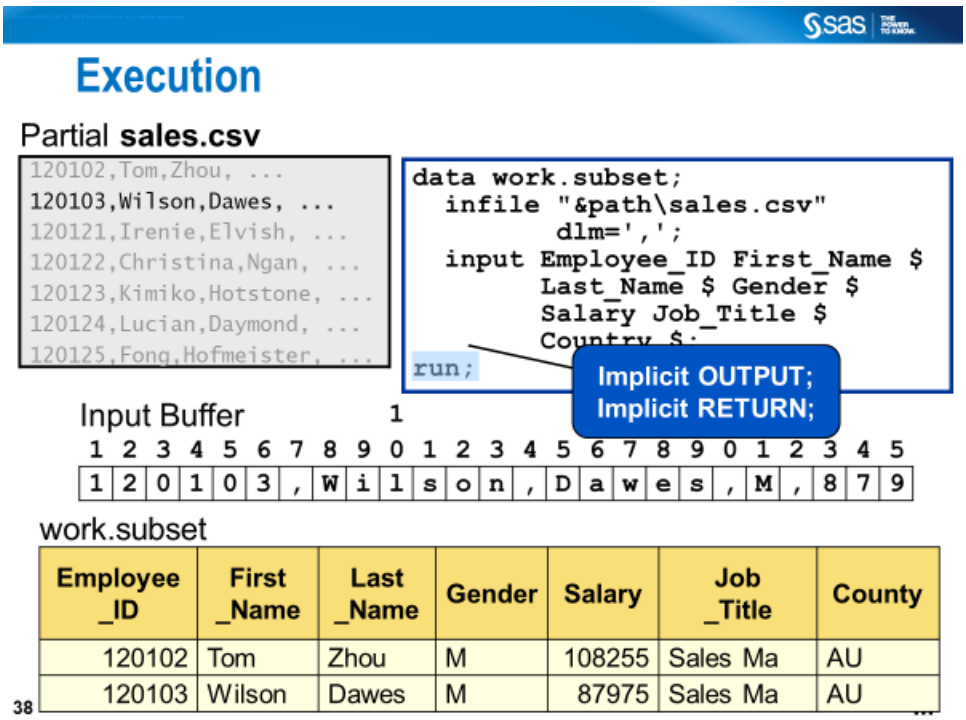


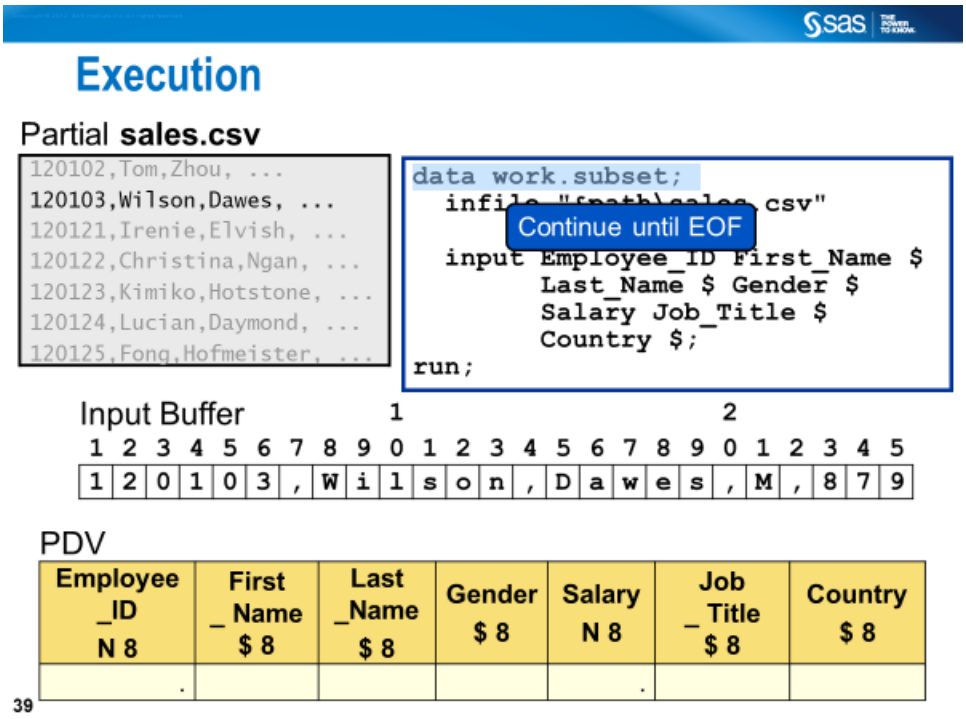












Review Questions

1. You are given a fixed-column raw data file containing numeric values with commas or dollar signs. Which numeric informat would you use in the INPUT statement to read those data values?

2. Explain when you are required to use the colon (:) format modifier with List Input.

3. Explain when you are required to use the ampersand (&) format modifier with List Input.

4. When reading free-format data, the INFORMAT statement has the same impact of the LENGTH statement for what kind of variables - numeric or character?

5. You have only standard data in fixed columns (multiple fields), and Column Input is an appropriate style of writing INPUT specifications. Alternatively, can you use Formatted Input to read these data?

6. You have only standard data in a delimited raw data file (multiple fields), and List Input is an appropriate style of writing INPUT specifications. Alternatively, can you use List Input with the colon format modifier to read these data?

7. In List Input, when you use the colon (:) format modifier to specify an informat after a variable name, what does the INFORMAT specify for the character variables, the length or the number of columns that are read?

8. Imagine that you have a space delimited file and you have a missing value. How would you represent the missing value when reading data using list or modified list input?

9. Describe the significance of each of the following options on the INFILE statement.
    - PAD
    - MISSOVER
    - TRUNCOVER
    - DLM=
    - DSD
    - FIRSTOBS=

10. Explain when you can use all three input styles (i.e., Column Input, Formatted Input, and List Input) to describe a record's values.

11. Is it allowed to skip certain fields when reading data using List Input in a DATA step?

12. Is it allowed to skip certain fields when reading data using Column Input in a DATA step?

13. When do you use the PUT statement preceded by a FILE statement in a DATA step?

14. When do you use the PUT statement with no FILE statement in a DATA step?

15. What is the difference between the PUT and PUTLOG statements in a DATA step?

16. What is the difference between INFILE and FILE statements?

17. How many ways can you add observations to an existing SAS data set? DATA Step Multiple Output Statement Method vs. PROC SQL INSERT INTO Method

18. What is the difference between INPUT and PUT statements?

19. What does the _INFILE_ statement do in the DATA step?