ASK THE EXPERT

The Lost Art of Reading and Loading Raw Data Into SAS®

Rich Papel
Principal Technical Training Consultant

Rich Papel

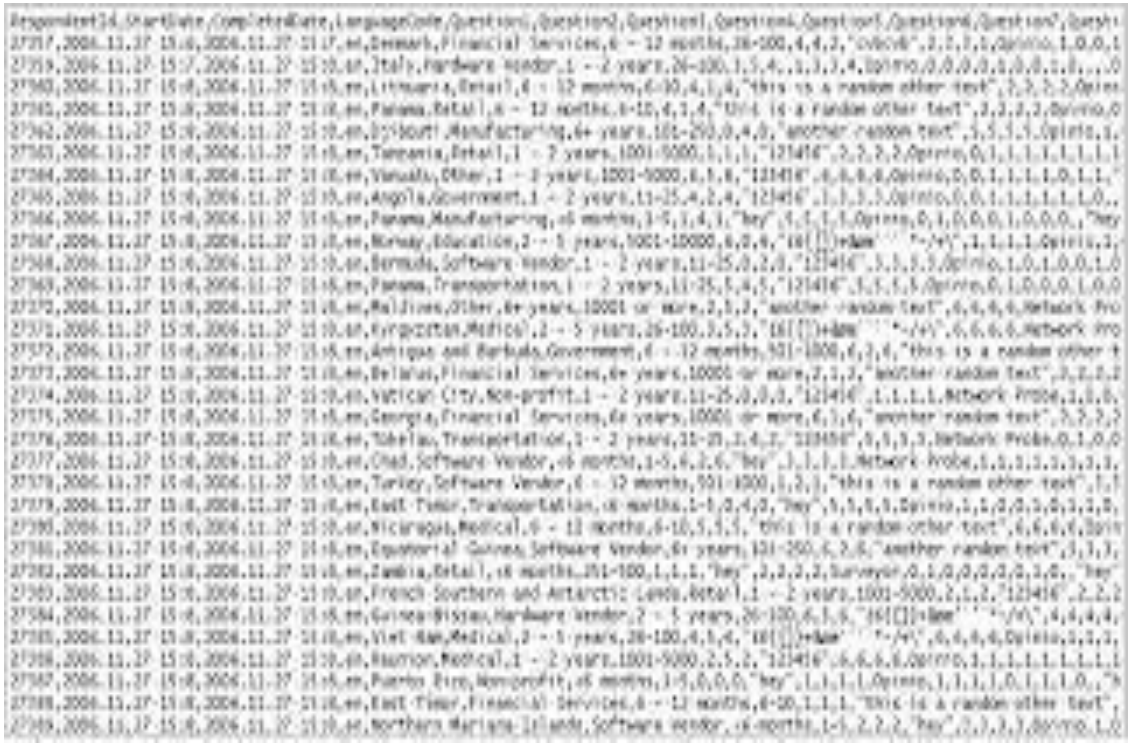
Principal Technical Training Consultant

Rich has more than 25 years in SAS Education with prior experience in public health, transportation and economics research. He holds graduate degrees in both biostatistics and urban planning.

The Lost Art of Reading and Loading Raw Data Into SAS

Presented August 24, 2023

Rich Papel
Principal Instructor, SAS/San Francisco



Introduction

Raw Data – What's That?

- Anything you can type on a keyboard
- If you can read it in Notepad – it's raw
- Does not contain metadata
- Often a collection of numbers and characters
- Maintained for subsequent data validation or auditing
- Used for peer-review studies
- Regulations often govern how long raw data retained

PROC IMPORT

```
%let path = c:\ask_the_expert_rawdata\data;

proc import datafile="&path\sales.csv"
        dbms=dlm out=sales_from_csv replace;
        delimiter=',';
        getnames=no;
        guessingrows=10;
run;
```

Obs	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7	VAR8	VAR9
1	120102	Tom	Zhou	M	108255	Sales Manager	AU	11AUG1973	06/01/1993
2	120103	Wilson	Dawes	M	87975	Sales Manager	AU	22JAN1953	01/01/1978
3	120121	Irenie	Elvish	F	26600	Sales Rep. II	AU	02AUG1948	01/01/1978
4	120122	Christina	Ngan	F	27475	Sales Rep. II	AU	27JUL1958	07/01/1982
5	120123	Kimiko	Hotstone	F	26190	Sales Rep. I	AU	28SEP1968	10/01/1989

PROC IMPORT Can Read These Files Only

- Microsoft Access databases
- Microsoft Exel workbooks
- Lotus 1-2-3 spreadsheets
- Paradox files
- SPSS files
- Stata files
- dBase
- JMP files
- Delimited files only

Why Not Use PROC IMPORT?

- Great for “idealized” situations
- Trouble with embedded quoting
- Titles/headers may be mishandled
- Incorrect variable lengths and type assignments
- Difficulty with hierarchical or multi-record file structures
- Incorrect starting row(s)
- Need structured rows and columns, e.g. SAS log, web logs, texts and narratives
- Likely still requires a Data Step for processing and analysis

Our Business Scenario

Management wants to analyze discount offer data. The data currently resides in a raw data file and needs to be converted to a SAS data set.

Partial **offers.dat**

```
          1    1    2    2
1---5----0----5----0----5
104012/02/11 Outdoors15%
202010/07/11 Golf     7%
103009/22/11 Shoes   10%
```

Cust_type	Offer_dt	Item_gp	Discount
1040	02DEC2011	Outdoors	0.15
2020	07OCT2011	Golf	0.07
1030	22SEP2011	Shoes	0.10

Only one “little” change in your SAS Code

Reading Raw Data

```
data new_sas_table;
   infile ;
   input ;
   remaining data
   step statements
   here
run;
```

Instead of

Reading SAS Data

```
data new_sas_table;
   set ;
   remaining data
   step statements
   here
run;
```

Input Styles

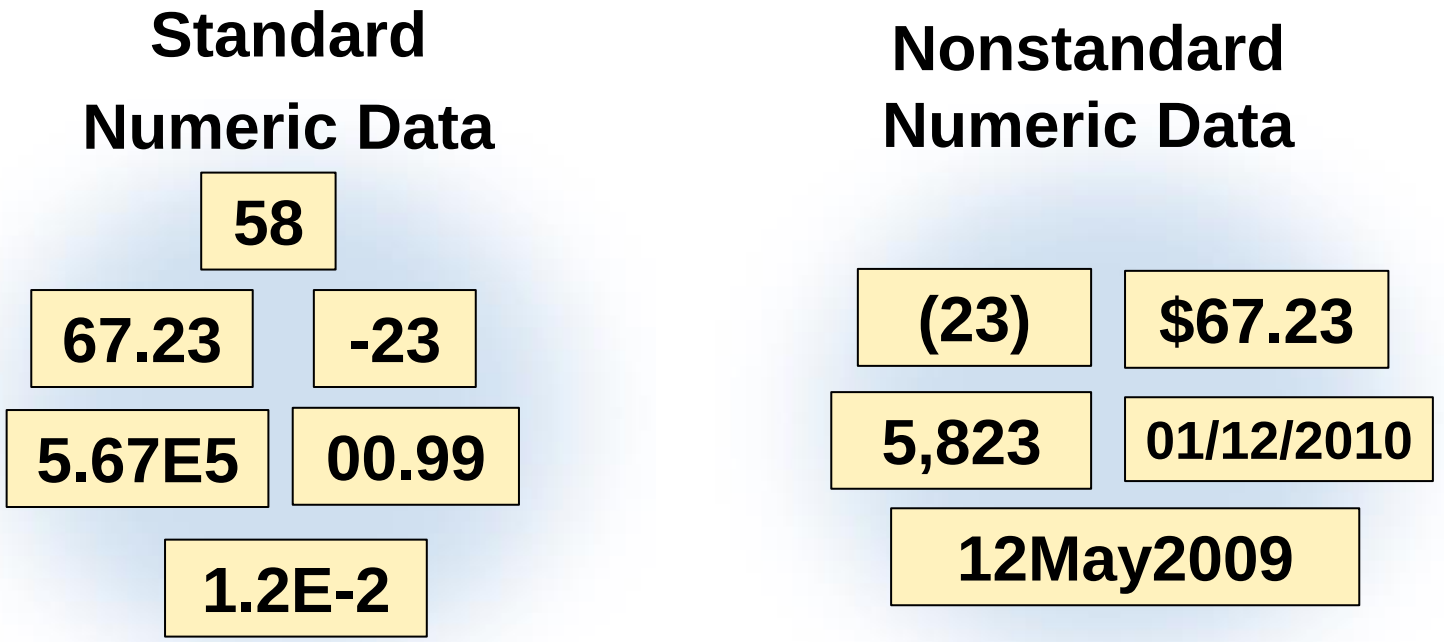
Column input, formatted input, list input, and named input are all styles of writing INPUT statement specifications.

Style	Use for Reading
Column input	Standard data in fixed columns
Formatted input	Standard and nonstandard data in fixed columns
List input	Standard and nonstandard data separated by blanks or some other delimiter
Named input	Standard data that is preceded by the name of the variable and an equal sign (=)

Standard and Nonstandard Data (Review)

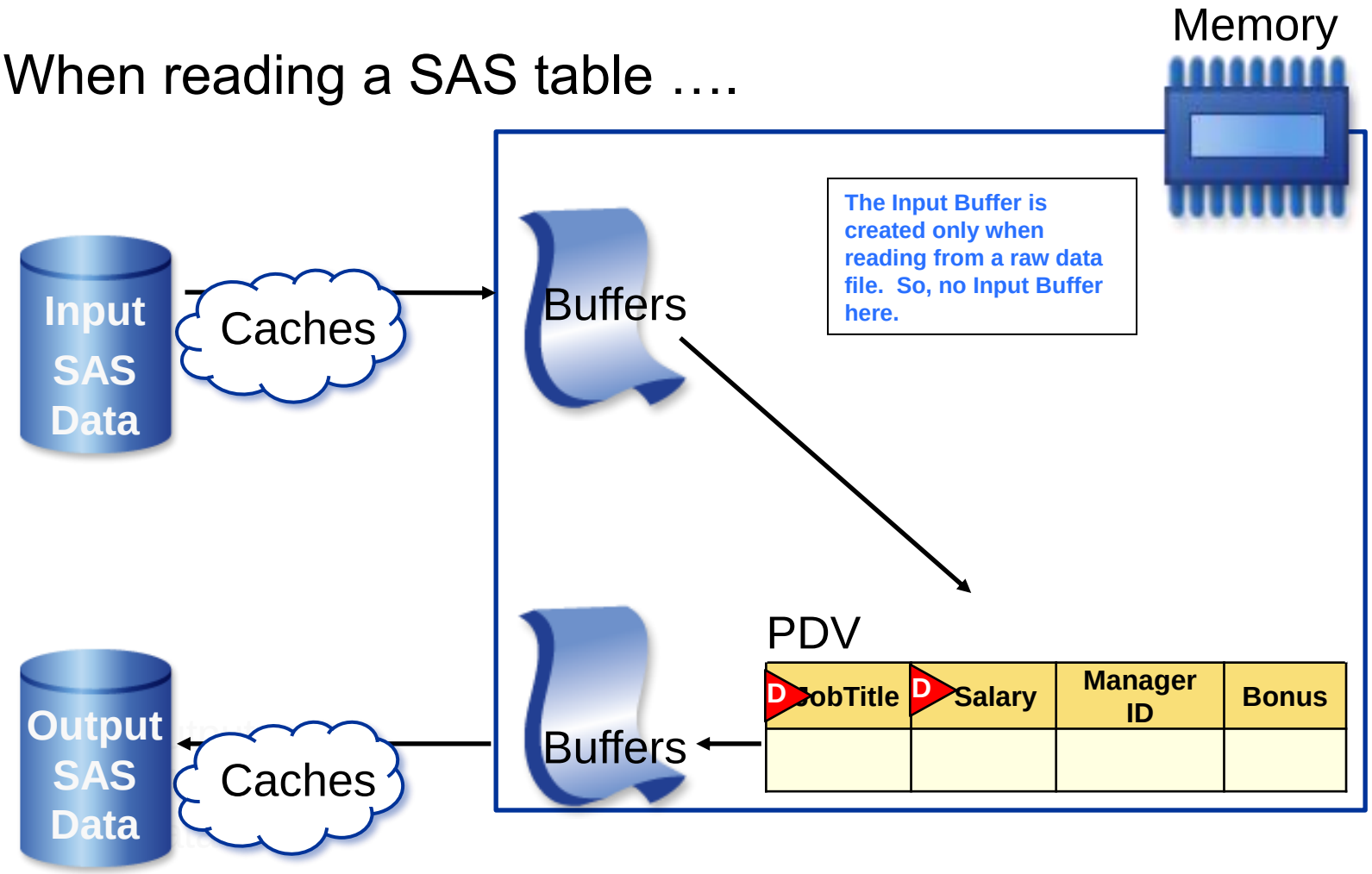
Standard data is data that SAS can read without any additional instruction.

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



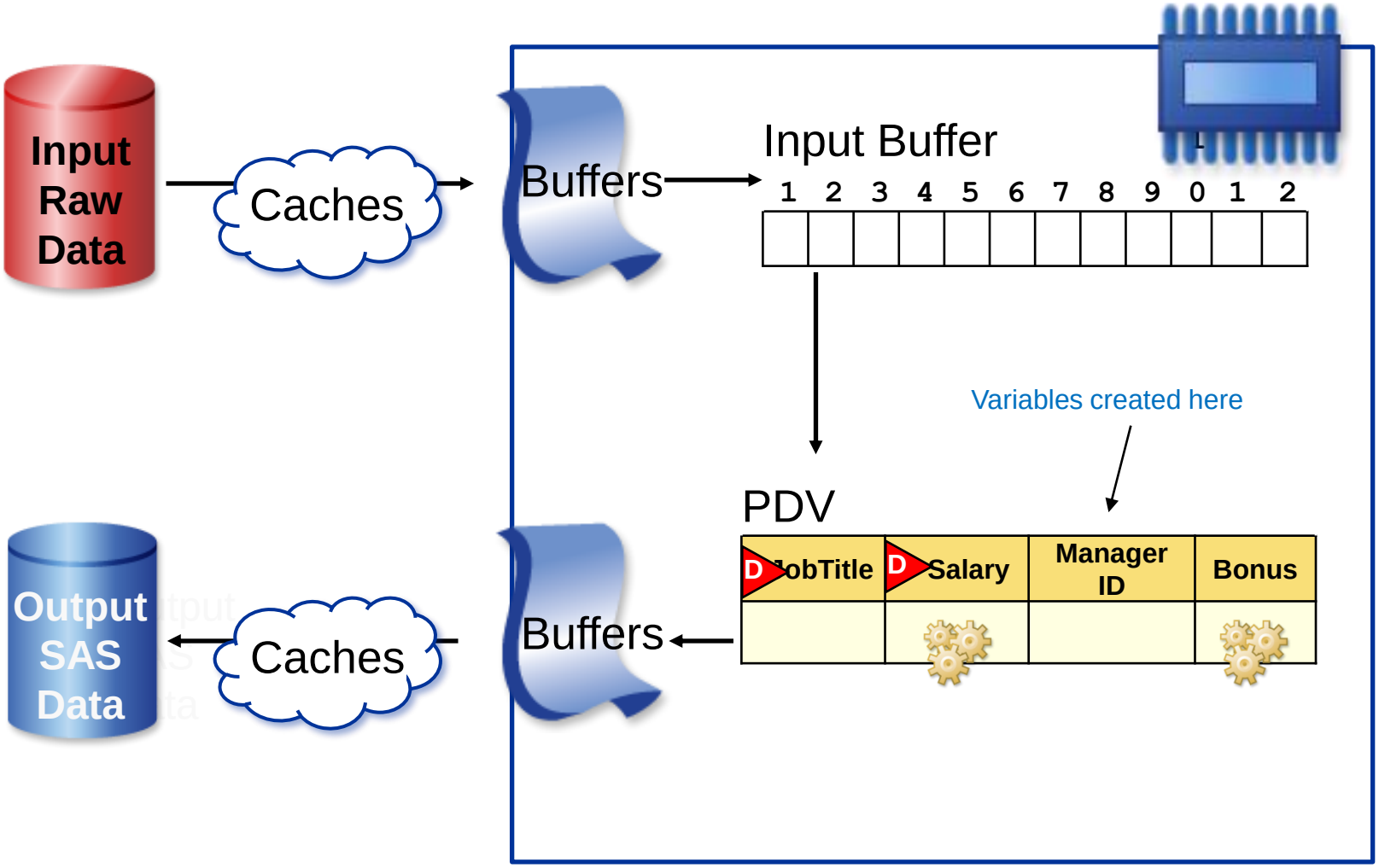
What's the difference between reading a SAS table vs Raw data?

When reading a SAS table ….



What's the difference between reading a SAS table vs raw data?

When reading raw data ….



FORMATTED INPUT

Use Formatted Input When…

- You are reading “standard” or “non-standard” data from fixed column positions and informats are needed. For example:

Partial **offers.dat**

```
                1    1    2    2
1---5----0----5----0----5
1040 12/02/11 Outdoors 15%
2020 10/07/11 Golf      7%
1030 09/22/11 Shoes    10%
1030 09/22/11 Clothes  10%
2020 07/08/11 Clothes  15%
2030 07/08/11 Clothes  25%
```

Reading Data Using Formatted Input

This SAS program uses formatted input to read the raw data file in **offers.dat**.

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

Absolute pointer control

INFILE - identifies the location and file name you're reading

INPUT – assigns the variables' names, types (char vs num) and arrangement, loading one record at a time into the Input Buffer

Program 1.sas

Reading Data Using Formatted Input

This SAS program uses formatted input to read the raw data file in **offers.dat**.

```
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 *absolute pointer-control variable informat* . . . ;

- starting position – specify pointer position
- Variable name – give your variables their names here
- Informat – an instruction for reading the specified positions

Program 1

Reading Data Using Formatted Input

Column pointer controls:

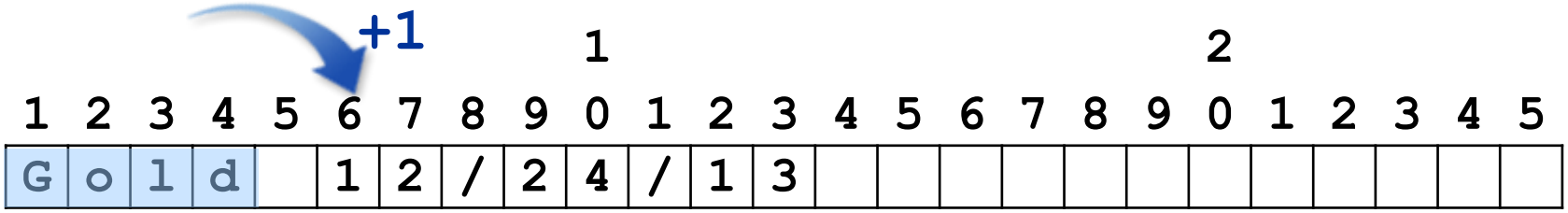
@n moves the pointer to column *n*. (absolute pointer)

+n moves the pointer *n* positions. (relative pointer cntl)

An informat specifies the following:

- the width of the input field
- how to read data values stored in the field

```
input @1 Code $4.
      +1 date mmddyy8.
```



Reading Data Using Formatted Input

Column pointer controls:

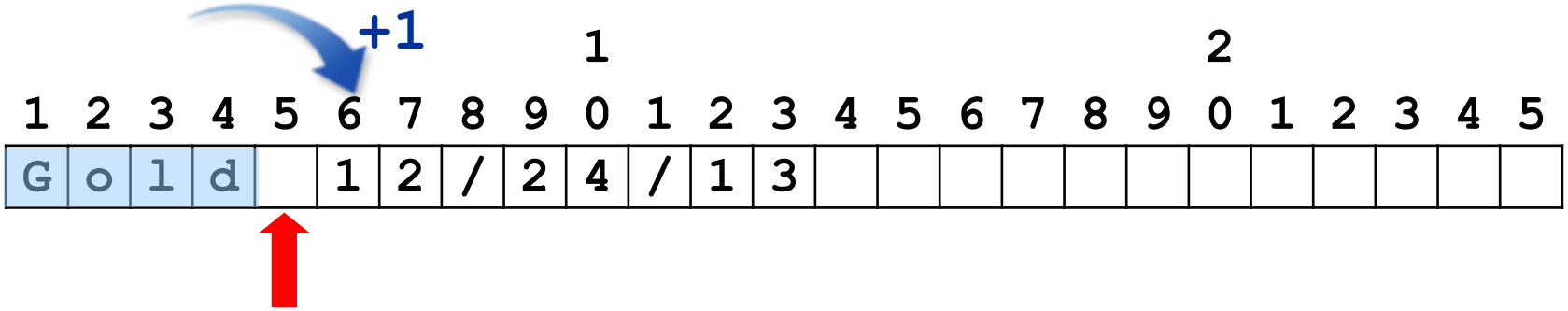
@n moves the pointer to column *n*. (absolute pointer)

+n moves the pointer *n* positions. (relative pointer cntl)

An informat specifies the following:

- the width of the input field
- how to read data values stored in the field

```
input @1 Code $4.
      +1 date mmddyy8.
```



Reading Data Using Formatted Input

Column pointer controls:

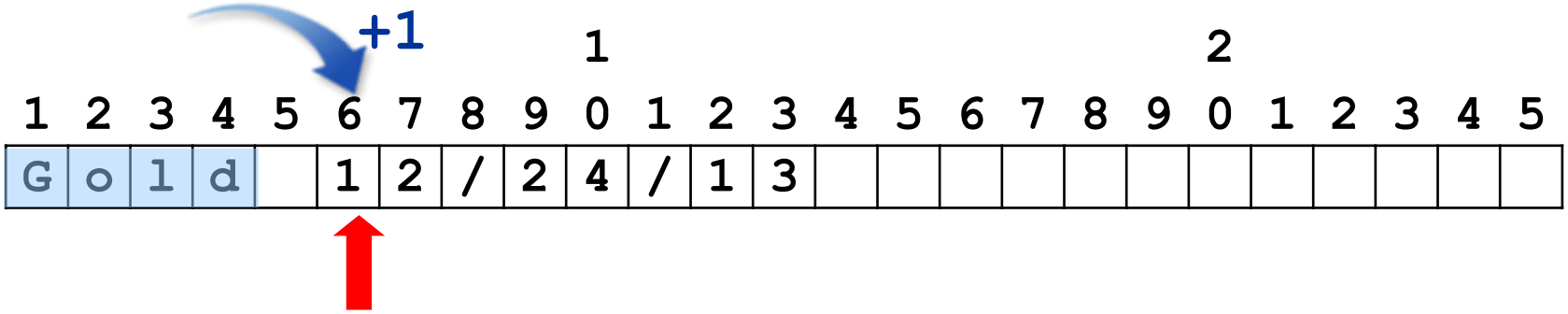
@n moves the pointer to column *n*. (absolute pointer)

+n moves the pointer *n* positions. (relative pointer cntl)

An informat specifies the following:

- the width of the input field
- how to read data values stored in the field

```
input @1 Code $4.
      +1 date mmddyy8.;
```



SAS Informat Examples

Examples of informats showing the raw data values and the converted SAS values:

Informat	Raw Data Value	SAS Data Value
$8.	`Outdoors`	*Outdoors*
5.	`12345`	*12345*
COMMA7. DOLLAR7.	`$12,345`	*12345*
COMMAX7. DOLLARX7.	`$12.345`	*12345*
EUROX7.	`€12.345`	*12345*
PERCENT3.	`15%`	*.15*

SAS Date Informat Examples

Examples of date informats showing the nonstandard raw data values and the converted SAS numeric values:

Informat	Raw Data Value	SAS Date Value
MMDDYY6.	010160	*0*
MMDDYY8.	01/01/60	*0*
MMDDYY10.	01/01/1960	*0*
DDMMYY6.	311260	*365*
DDMMYY8.	31/12/60	*365*
DDMMYY10.	31/12/1960	*365*
DATE7.	31DEC59	*-1*
DATE9.	31DEC1959	*-1*

Reading Data Using Formatted Input

Use formatted input to create a SAS data set named **discounts** from the raw data in **offers.dat**.

Layout: **offers.dat**

Description	Column
Customer Type	1-4
Offer Date	5-12
Item Group	14-21
Discount	22-24

Partial **offers.dat**

```
                1    1    2    2
1---5----0----5----0----5
1040 12/02/11 Outdoors 15%
2020 10/07/11 Golf      7%
1030 09/22/11 Shoes    10%
1030 09/22/11 Clothes  10%
2020 07/08/11 Clothes  15%
2030 07/08/11 Clothes  25%
```

Writing INPUT Specifications

Identify the starting position, variable name, and informat for each input field.

```
input @1 Cust_type 4.
```

Layout: **offers.dat**

Description	Column
Customer Type	1-4
Offer Date	5-12
Item Group	14-21
Discount	22-24

Partial **offers.dat**

```
                1    1    2    2
1---5----0----5----0----5
1040 12/02/11 Outdoors 15%
202010/07/11 Golf      7%
103009/22/11 Shoes    10%
103009/22/11 Clothes  10%
202007/08/11 Clothes  15%
203007/08/11 Clothes  25%
```

Writing INPUT Specifications

Continue writing the INPUT statement to read Offer Date. (Hint: Use the MMDDYY8. informat.)

```
input @1 Cust_type 4.
      @5 Offer_dt mmddyy8.
```

Layout: **offers.dat**

Description	Column
Customer Type	1-4
Offer Date	5-12
Item Group	14-21
Discount	22-24

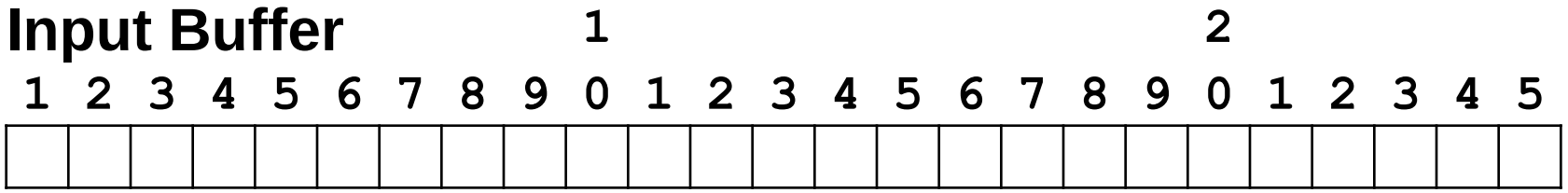
Partial **offers.dat**

```
                1    1    2    2
1---5----0----5----0----5
1040 12/02/11 Outdoors 15%
202010/07/11 Golf      7%
103009/22/11 Shoes    10%
103009/22/11 Clothes  10%
202007/08/11 Clothes  15%
203007/08/11 Clothes  25%
```

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



PDV

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

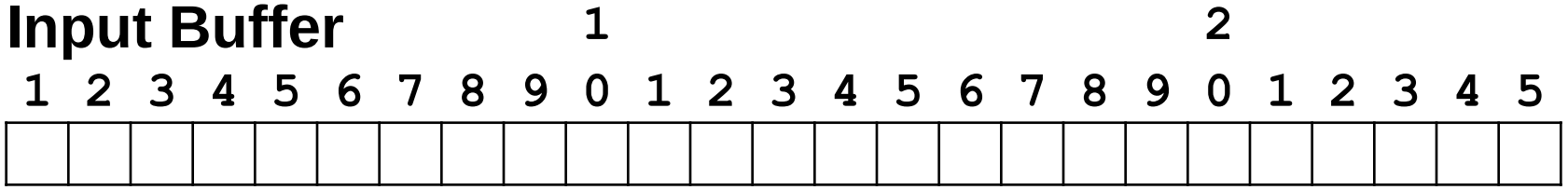
Program 1 ...

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



PDV

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

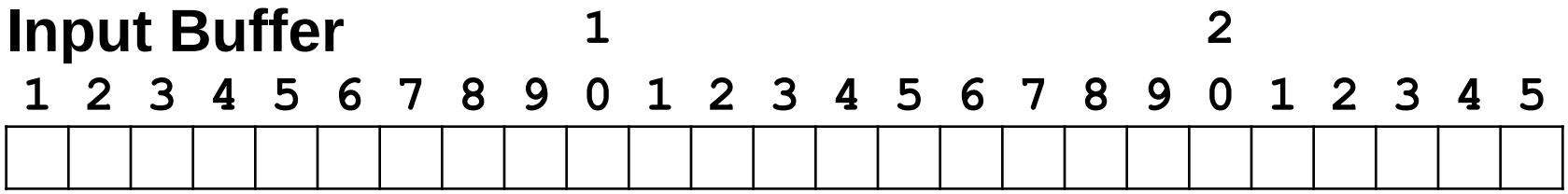
...

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



PDV

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

...

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	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
.	.		.

...

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	.		.

...

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		.

...

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	.

...

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

...

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

```
                                                                  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

...

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													
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	/	2	3		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
.	.		.

Read Discount Offers File: Output

```
proc print data=work.discounts noobs;
run;
```

Partial PROC PRINT Output

SAS date values

Cust_type	Offer_dt	Item_gp	Discount
1040	23346	Outdoors	0.15
2020	23290	Golf	0.07
1030	23275	Shoes	0.10
1030	23275	Clothes	0.10

Read Discount Offers File: Output

```
proc print data=work.discounts noobs;
 format Offer_dt date9. discount percent5.1;
run;
```

Partial PROC PRINT Output

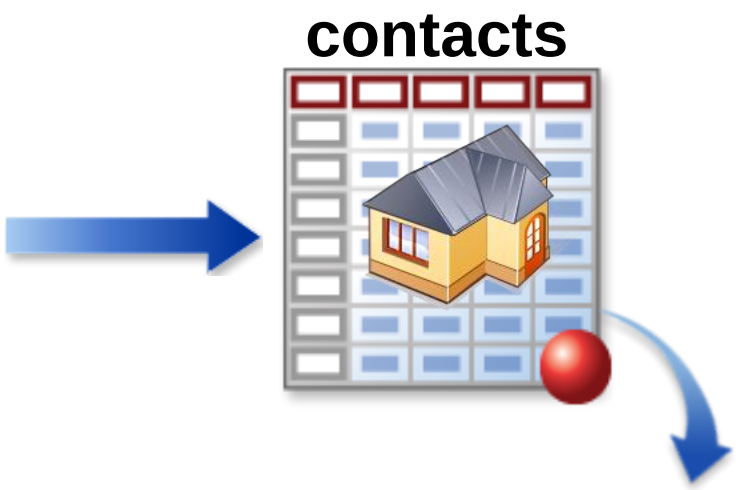
Cust_type	Offer_dt	Item_gp	Discount
1040	02DEC2023	Outdoors	15%
2020	07OCT2023	Golf	7%
1030	22SEP2023	Shoes	10%
1030	22SEP2023	Clothes	10%

Business Scenario

Create a SAS data set that contains the necessary information for the company's contact directory.

Partial **Address.dat**

```
          1    1    2    2
1---5----0----5----0----5
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```



FullName	Address2	Phone
Ms. Sue Farr	Macon, GA 31298	869-7008
Dr. Kay B. Cox	Kern, CA 93280	483-3321
Mr. Ron Mason	Miami, FL 33054	589-9030
Ms. G. H. Ruth	Munger, MI 48747	754-3582

Multiple INPUT Statements

By default, SAS loads a new record into the input buffer when it encounters an INPUT statement.

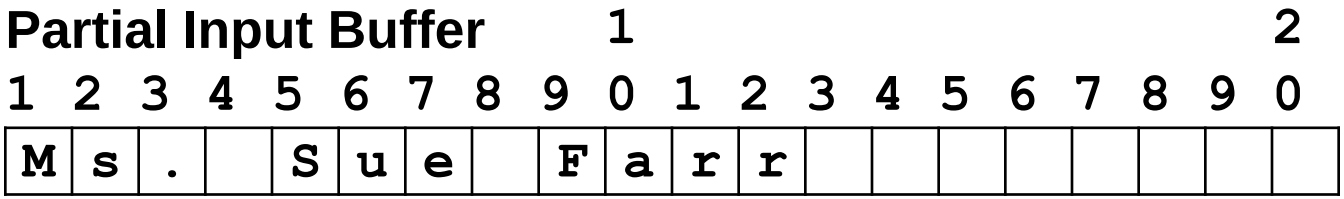
You can have multiple INPUT statements in one DATA step.

```
data contacts;
   infile "&path\address.dat";
   input FullName $30.;
   input;
   input Address2 $25.;
   input Phone $8.;
run;
```

Ms. Sue Farr
15 Harvey Rd.
Macon, GA 31298
869-7008

Load first line of raw data

Partial Input Buffer



Program 2 ...

Multiple INPUT Statements

```
data contacts;
   infile "&path\address.dat";
   input FullName $30.;
   input;
   input Address2 $25.;
   input Phone $8.;
run;
```

```
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```

Load second line of raw data

Partial 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 | 5 |   | H | a | r | v | e | y |   | R | d | . |   |   |   |   |   |   |   |
```

- Even though no variables are listed, the INPUT statement still loads the raw data line into the input buffer.

...

Multiple INPUT Statements

```
data contacts;
   infile "&path\address.dat";
   input FullName $30.;
   input;
   input Address2 $25.;
   input Phone $8.;
run;
```

```
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```

Load third line of raw data

Partial Input Buffer

```
          1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
M a c o n , _ G A _ 3 1 2 9 8 _ _ _ _ _
```


...

Multiple INPUT Statements

```
data contacts;
   infile "&path\address.dat";
   input FullName $30.;
   input;
   input Address2 $25.;
   input Phone $8.;
run; /*implied output*/
```

```
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```

Load fourth line of raw data

Partial Input Buffer

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

PDV

FullName $ 30	Address2 $ 25	Phone $ 8
Ms. Sue Farr	Macon, GA 31298	869-7008

Multiple INPUT Statements

Partial SAS Log

```
NOTE: 48 records were read from the infile 'address.dat'.
      The minimum record length was 18.
      The maximum record length was 30.
NOTE: The data set WORK.CONTACTS has 12 observations
      and 3 variables.
```

Multiple INPUT Statements

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

Partial PROC PRINT Output

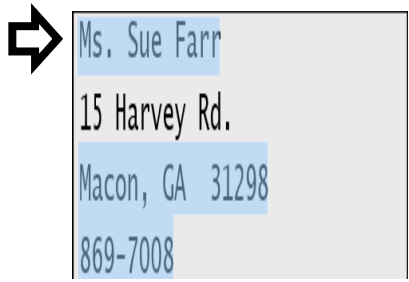
FullName	Address2	Phone
Ms. Sue Farr	Macon, GA 31298	869-7008
Dr. Kay B. Cox	Kern, CA 93280	483-3321
Mr. Ron Mason	Miami, FL 33054	589-9030
Ms. G. H. Ruth	Munger, MI 48747	754-3582

Line Pointer Controls – Relative Line Pointer

```
data contacts;
   infile "&path\address.dat";
   input FullName $30. / /
         Address2 $25. /
         Phone $8. ;
run;
```

Relative Line Pointer Control

Each '/' loads another record into the Input Buffer



Input Statement Loads first line of raw data

Partial Input Buffer

	1									2									
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
M	s	.		S	u	e		F	a	r	r								

SAS loads the next record when it encounters a forward slash.

Program 3 ...

Line Pointer Controls

```
data contacts;
   infile "&path\address.dat";
   input FullName $30. / /
         Address2 $25. /
         Phone $8. ;
run;
```

Load second line of raw data

```
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```

Partial Input Buffer

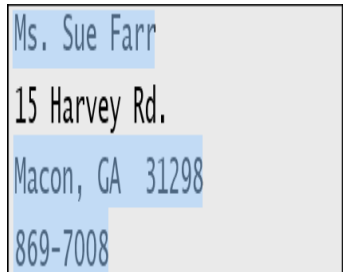
									1									2
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
1	5		H	a	r	v	e	y		R	d	.							

...

Line Pointer Controls

```
data contacts;
   infile "&path\address.dat";
   input FullName $30. / /
         Address2 $25. /
         Phone $8. ;
run;
```

Load third line of raw data



Partial Input Buffer

	1									2									
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
M	a	c	o	n	,		G	A		3	1	2	9	8					

...

Line Pointer Controls

```
data contacts;
   infile "&path\address.dat";
   input FullName $30. / /
         Address2 $25. /
         Phone $8. ;
run;
```

Load fourth line of raw data

```
Ms. Sue Farr
15 Harvey Rd.
Macon, GA  31298
869-7008
```

Partial Input Buffer

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

Line Pointer Controls

Partial SAS Log

```
NOTE: 48 records were read from the infile 'address.dat'.
      The minimum record length was 18.
      The maximum record length was 30.
NOTE: The data set WORK.CONTACTS has 12 observations
      and 3 variables.
```

Line Pointer Controls

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

Partial PROC PRINT Output

FullName	Address2	Phone
Ms. Sue Farr	Macon, GA 31298	869-7008
Dr. Kay B. Cox	Kern, CA 93280	483-3321
Mr. Ron Mason	Miami, FL 33054	589-9030
Ms. G. H. Ruth	Munger, MI 48747	754-3582

Absolute Line Pointer Control

```
data contacts;
   infile "&path\address.dat";      #1
   input FullName $30.
         #3 Address2 $25.           #3
         #4 Phone $8. ;             #4
run;
```

Absolute line pointer controls

Load first line of raw data

Partial Input Buffer

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
⇨	M	s	.		S	u	e		F	a	r	r								
	1	5		H	a	r	v	e	y		R	d	.							
⇨	M	a	c	o	n	,		G	A		3	1	2	9	8					
⇨	8	6	9	-	7	0	0	8												

Program 4 ...

Absolute Line Pointer Control

```
data contacts;
   infile "&path\address.dat";      #1
   input FullName $30.
         #3 Address2 $25.           #3
         #4 Phone $8. ;             #4
run;
```

Absolute line pointer controls

Load first line of raw data

Partial Input Buffer

									1										2
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
M	s	.		S	u	e		F	a	r	r								

Program 4 ...

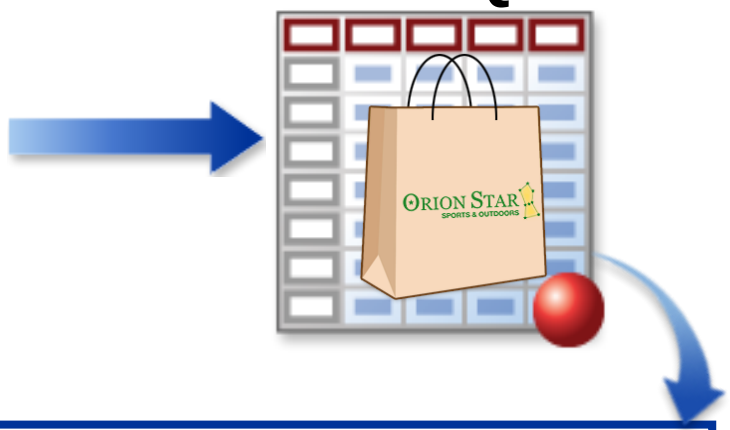
Mixed Record Types

Create a SAS data set from the raw data that contains records that do not have the same format. We will read with Formatted Input.

sales.dat

```
                1    1    2    2    3
1---5----0----5----0----5----0
101  USA 1-20-2011 3295.50
3034 EUR 30JAN2011 1876,30
101  USA 1-30-2011 2938.00
```

salesQ1



Sale ID	Location	Sale Date	Amount
101	USA	18647	3295.50
3034	EUR	18657	1876.30
101	USA	18657	2938.00
128	USA	18663	2908.74

First Attempt: Unexpected Output

```
proc print data=salesQ1 noobs;
run;
```

PROC PRINT Output - Incorrect

SaleID	Location	SaleDate	Amount
101	USA	.	.
101	USA	18663	2908.74
1345	EUR	.	278910.00

To get the correct results, SAS needs some way to keep the second INPUT statement from moving to the next line of raw data.

Desired Output

work.salesQ1

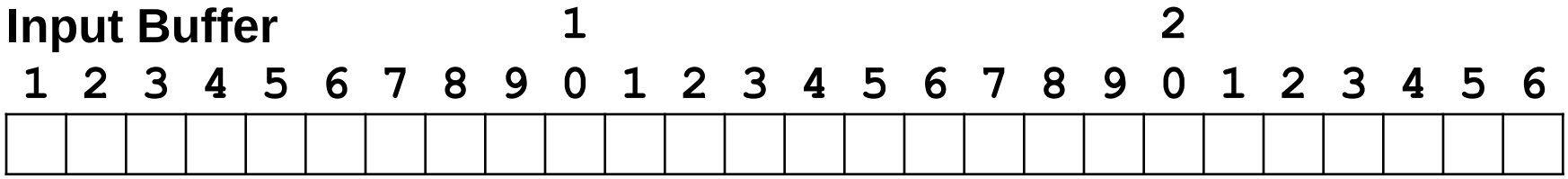
SaleID	Location	SaleDate	Amount
101	USA	18647	3295.50
3034	EUR	18657	1876.30
101	USA	18657	2938.00
128	USA	18663	2908.74
1345	EUR	18664	3145.60
109	USA	18703	2789.10

Execution: First Attempt

Initialize PDV

```
data salesQ1;
   infile "s:\&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Input Buffer



PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
		.	.

...

Mixed Record Types: First Attempt

A good start, but yields unexpected results.

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

sales.dat

```
                1    1    2    2    3
1---5----0----5----0----5----0----5----0
101  USA 1-20-2011 3295.50
3034 EUR 30JAN2011 1876,30
101  USA 1-30-2011 2938.00
```

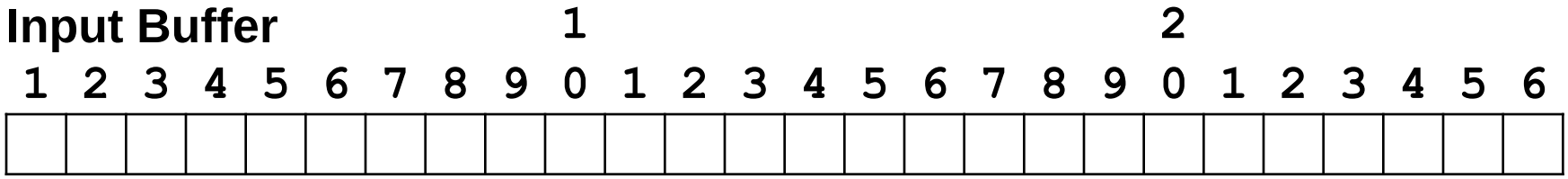
Program 5_error

Execution: First Attempt

Specify input data file

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Input Buffer



PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
		.	.

...

Execution: First Attempt

Load first record to input buffer

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
                @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
                @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
		.	.

...

Execution: First Attempt

Load values into PDV

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Input Buffer

	1									2										3						
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
101	USA	.	.

...

Execution: First Attempt

Load values into PDV

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
                @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
                @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
101	USA	.	.

...

Execution: First Attempt

Load values into PDV

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
                @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
                @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: First Attempt

True

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: First Attempt

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.
run;
```

Oh No!! Loads next record to input buffer

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	6
3	0	3	4		E	U	R		3	0	J	A	N	2	0	1	1		1	8	7	6	,	3	0

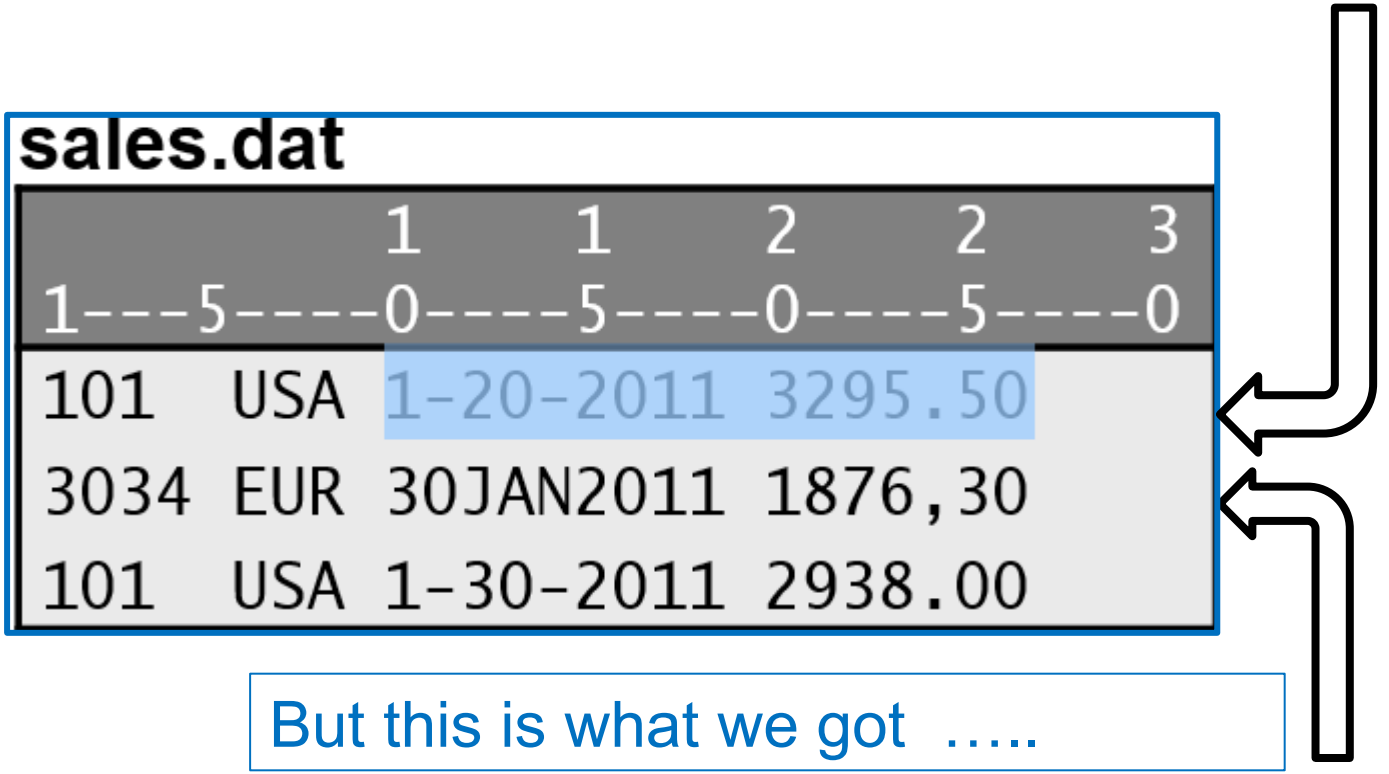
PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
101	USA	.	.

...

Here's the Raw Data….

This is what we want to read…..



But this is what we got …..

Execution: First Attempt

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Invalid data message written to SAS log

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 6
3 0 3 4   E U R   3 0 J A N 2 0 1 1   1 8 7 6 , 3 0
```

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: First Attempt

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
                @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
                @20 Amount commax7.;
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	6
3	0	3	4		E	U	R		3	0	J	A	N	2	0	1	1		1	8	7	6	,	3	0

PDV

SaleID	Location	SaleDate	Amount
$ 4	$ 3	N 8	N 8
101	USA	.	.

...

Execution: First Attempt

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3.;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
                @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
                @20 Amount commax7.;
run;
```

Continue until EOF

Input Buffer

	1									2										3						
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	6
3	0	3	4		E	U	R		3	0	J	A	N	2	0	1	1		1	8	7	6	,	3	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

First Attempt: Unexpected Output

Partial SAS Log

```
NOTE: Invalid data for SaleDate in line 2 10-19.
NOTE: Invalid data for Amount in line 2 20-26.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+
          3034 EUR 30JAN2011 1876,30
SaleID=101 Location=USA SaleDate=. Amount=. _ERROR_=1 _N_=1
          .
          .
NOTE: 6 records were read from the infile 'sales.dat'.
      The minimum record length was 26.
      The maximum record length was 27.
NOTE: The data set WORK.SALESQ1 has 3 observations and 4 variables.
```

Mixed Record Types: Correct Program

The single trailing @ prevents the subsequent INPUT statement from loading another record until SAS does one of the following:

- executes an INPUT statement with no trailing @
- begins the next iteration of the DATA step

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Program 5_good

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Load input buffer

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	6	
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0	

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
		.	.

...

Execution: Correct Program

Load values into PDV

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Do not read new record at next INPUT statement

Hold

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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

True

Hold

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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

Do ***not*** load another record to the input buffer

Hold

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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	.	.

...

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
run;
```

The @ "hold" is now released here.

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 6
1 0 1       U S A   1 - 2 0 - 2 0 1 1   3 2 9 5 . 5 0
```

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	18647	3295.50

...

Execution: Correct Program

```
data salesQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
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	6
1	0	1			U	S	A		1	-	2	0	-	2	0	1	1		3	2	9	5	.	5	0

PDV

SaleID $ 4	Location $ 3	SaleDate N 8	Amount N 8
101	USA	18647	3295.50

Correct Program: Output

Partial SAS Log

```
NOTE: 6 records were read from the infile 'sales.dat'.
      The minimum record length was 26.
      The maximum record length was 27.
NOTE: The data set WORK.SALESQ1 has 6 observations and 4 variables.
```

PROC PRINT Output

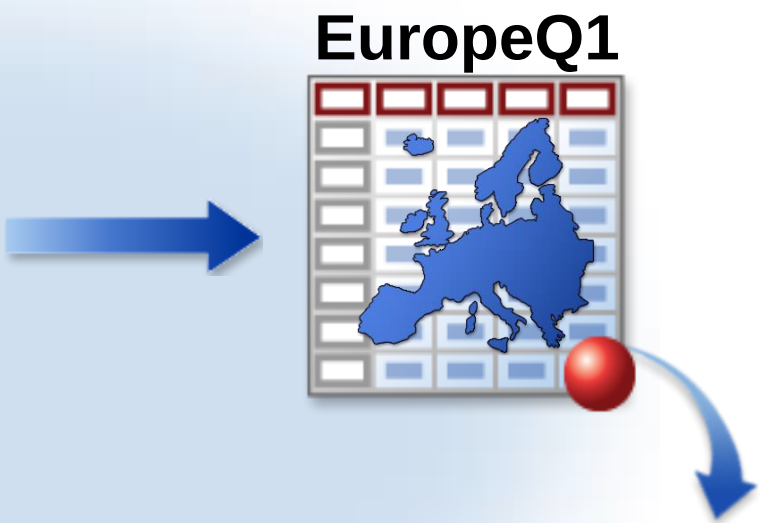
SaleID	Location	SaleDate	Amount
101	USA	18647	3295.50
3034	EUR	18657	1876.30
101	USA	18657	2938.00
128	USA	18663	2908.74
1345	EUR	18664	3145.60
109	USA	18703	2789.10

Business Scenario

Create a SAS data set that contains only the European observations.

sales.dat

```
                1    1    2    2    3
1---5----0----5----0----5----0
101   USA 1-20-2011 3295.50
3034  EUR 30JAN2011 1876,30
101   USA 1-30-2011 2938.00
128   USA 2-5-2011   2908.74
1345  EUR 6FEB2011   3145,60
```



SaleID	Location	SaleDate	Amount
3034	EUR	18657	1876.3
1345	EUR	18664	3145.6

Is this the best placement for the subsetting IF statement?

```
data EuropeQ1;
   infile "&path\sales.dat";
   input SaleID $4. @6 Location $3. @;
   if Location='USA' then
      input @10 SaleDate mmddyy10.
            @20 Amount 7.;
   else if Location='EUR' then
      input @10 SaleDate date9.
            @20 Amount commax7.;
   if Location = 'EUR';
run;
```

Program 6_less efficient

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;
```

Subsetting Mixed Record Types: Output

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

PROC PRINT Output

SaleID	Location	SaleDate	Amount
3034	EUR	18657	1876.3
1345	EUR	18664	3145.6

COLUMN INPUT

Embedding Your Data in a Data Step

```
data sales_employees;
   infile datalines;
   input @1 id $6.
         @8 fname $10.
         @21 lname $19.
         @40 gender $1.
         @43 job_title $20.;
datalines;
120102 Tom            Zhou                M  Sales Manager
120103 Wilson         Dawes               M  Sales Manager
120121 Irenie         Elvish              F  Sales Rep. II
120122 Christina      Ngan                F  Sales Rep. II
120123 Kimiko         Hotstone            F  Sales Rep. I
;
```

Still using formatted Input, but not necessary – All data fields are "standard" character or numeric

id	fname	lname	gender	job_title
120102	Tom	Zhou	M	Sales Manager
120103	Wilson	Dawes	M	Sales Manager
120121	Irenie	Elvish	F	Sales Rep. II
120122	Christina	Ngan	F	Sales Rep. II
120123	Kimiko	Hotstone	F	Sales Rep. I

Program 6A

Changing the Input Style to Column Input

Requirements:

- fixed position data still required
- all fields must contain standard character and/or numeric data
- perfectly fine to mix Formatted and Column Input styles

Column Input works here because no informat(s) are needed.

```
data sales_employees;
*infile "&path/sales1.dat";
infile datalines;
input id $1-6 fname $ 8-17 lname $ 21-39
      gender $ 40 job_title $ 43-55 salary 56-61;
datalines;
120102 Tom          Zhou              M  Sales Manager108255
120103 Wilson       Dawes             M  Sales Manager87975
120121 Irenie       Elvish            F  Sales Rep. II26600
120122 Christina    Ngan              F  Sales Rep. II27475
120123 Kimiko       Hotstone          F  Sales Rep. I26190
;
```

id	fname	lname	gender	job_title	salary
120102	Tom	Zhou	M	Sales Manager	108255
120103	Wilson	Dawes	M	Sales Manager	87975

Program 6B

Changing the Input Style to Column Input

```
data sales_employees;
infile datalines;
input id $1-6 fname $ 8-17 lname $ 21-39
      gender $ 40 job_title $ 43-55 salary 56-61;
datalines;
120102 Tom          Zhou               M  Sales Manager108,255
120103 Wilson       Dawes              M  Sales Manager87,975
120121 Irenie       Elvish             F  Sales Rep. II26,600
120122 Christina    Ngan               F  Sales Rep. II27,475
120123 Kimiko       Hotstone           F  Sales Rep. I26,190
;
```

id	fname	lname	gender	job_title	salary
120102	Tom	Zhou	M	Sales Manager	.
120103	Wilson	Dawes	M	Sales Manager	.
120121	Irenie	Elvish	F	Sales Rep. II	.
120122	Christina	Ngan	F	Sales Rep. II	.
120123	Kimiko	Hotstone	F	Sales Rep. I2	.

But does not work here because an informat is required to read Salary.

Program 7.sas

Mixed Input Style:Column and Formatted Input

```
data sales_employees;
*infile "&path/sales1.dat";
infile datalines;
input id $1-6 fname $ 8-17 lname $ 21-39
      gender $ 40 job_title $ 43-55
      @56 salary comma7.;
datalines;
120102 Tom          Zhou              M  Sales Manager108,255
120103 Wilson       Dawes             M  Sales Manager87,975
120121 Irenie       Elvish            F  Sales Rep. II26,600
120122 Christina    Ngan              F  Sales Rep. II27,475
120123 Kimiko       Hotstone          F  Sales Rep. I26,190
;
```

This statement not needed since data is in-stream.

Obs	id	fname	lname	gender	job_title	salary
1	120102	Tom	Zhou	M	Sales Manager	108255
2	120103	Wilson	Dawes	M	Sales Manager	87975
3	120121	Irenie	Elvish	F	Sales Rep. II	26600
4	120122	Christina	Ngan	F	Sales Rep. II	27475
5	120123	Kimiko	Hotstone	F	Sales Rep. I2	6190

LIST INPUT

Reading Delimited Data using List Input

Use list input to read delimited raw data files.

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
```

- SAS considers a space (blank) to be the default delimiter.
- Both standard and nonstandard data can be read.
- Fields must be read sequentially, left to right.
- Default length of all variables is 8 bytes

List Input: 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 $;
1 run;                                  2
```

Input Buffer

PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
.				.		

Program 8

...

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 $;
```

SAS reads a record into the input buffer.

Input Buffer

```
                1
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
.				.		

...

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 $;
```

SAS scans until it reaches a delimiter.

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
.				.		

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 $
```

The value is converted from text to a floating-point numeric value and copied to the PDV.

Input Buffer

```
1 run
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				.		

...

Execution

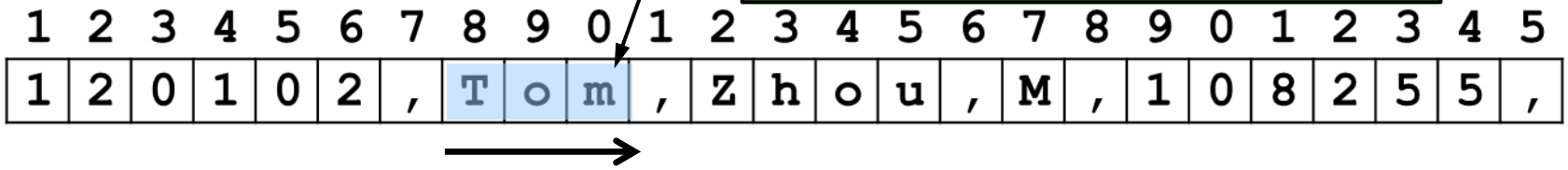
Partial **sales.csv**

```
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 $
1 run;
```

SAS skips the delimiter and scans to the next delimiter.

Input Buffer



PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
120102				.		

...

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 $
1 run;
```

The text value is copied to the PDV without conversion.

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			.		

...

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;
```

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

...

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

...

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

...

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, ...
```

Reinitialize PDV

```
data work.subset;
  infile "&path\sales.csv"
         dlm=',';
  input Employee_ID First_Name $
        Last_Name $ Gender $
        Salary Job_Title $
        Country $;
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
1	2	0	1	0	2	,	T	o	m	,	Z	h	o	u	,	M	,	1	0	8	2	5	5	,

All variables in the PDV are reinitialized.

PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
.				.		

...

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;
```

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
.				.		

...

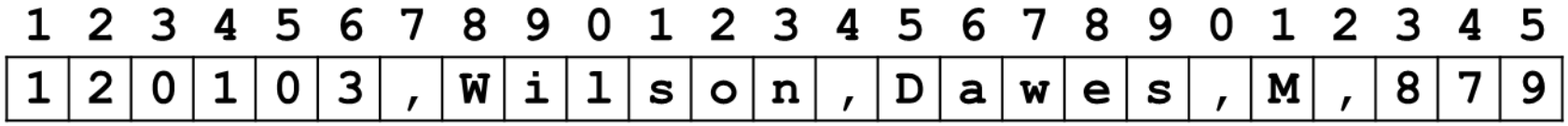
Execution

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;
```

Input Buffer



PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
.				.		

...

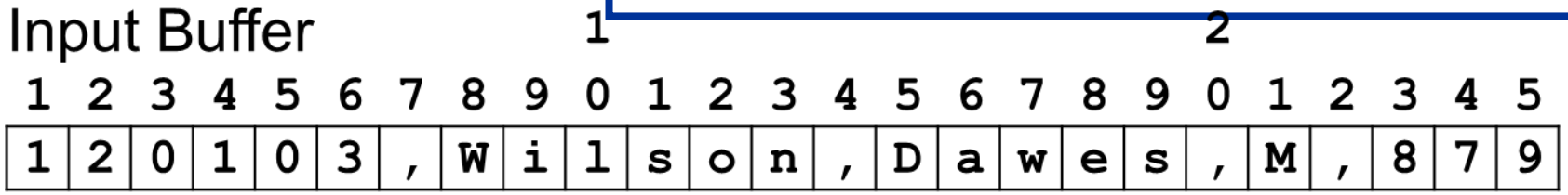
Execution

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;
```

Input Buffer



PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
120103	Wilson	Dawes	M	87975	Sales Ma	AU

...

Execution

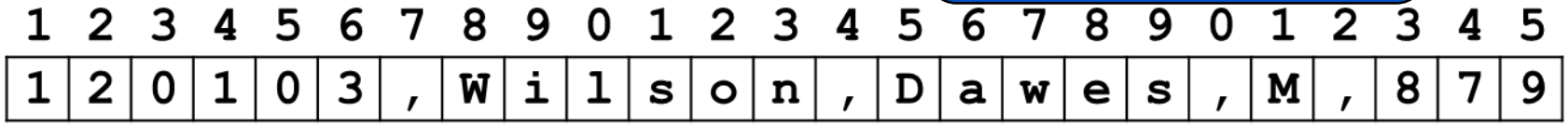
Partial **sales.csv**

```
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



work.subset

Employee_ID	First_Name	Last_Name	Gender	Salary	Job_Title	County
120102	Tom	Zhou	M	108255	Sales Ma	AU
120103	Wilson	Dawes	M	87975	Sales Ma	AU

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;
   infi                    .csv"
                Continue until EOF
   input Employee_ID First_Name $
         Last_Name $ Gender $
         Salary Job_Title $
         Country $;
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
1 2 0 1 0 3 , W i l s o n , D a w e s , M , 8 7 9
```

PDV

Employee_ID N 8	First_Name $ 8	Last_Name $ 8	Gender $ 8	Salary N 8	Job_Title $ 8	Country $ 8
.				.		

Viewing the Output

```
proc print data=work.subset noobs;
run;
```

employee_ID	first_name	last_name	gender	salary	job_title	country
120102	Tom	Zhou	M	108255	Sales Ma	AU
120103	Wilson	Dawes	M	87975	Sales Ma	AU
120121	Irenie	Elvish	F	26600	Sales Re	AU
120122	Christin	Ngan	F	27475	Sales Re	AU
120123	Kimiko	Hotstone	F	26190	Sales Re	AU
120124	Lucian	Daymond	M	26480	Sales Re	AU
120125	Fong	Hofmeist	M	32040	Sales Re	AU

Some character values are truncated.

Program 8

List Input: LENGTH Statement

The *LENGTH statement* defines the type and length of a variable. Default length is 8 bytes

LENGTH *variable(s)* <$> *length …*;

```
data work.subset;
   length first_name $ 12 last_name $18
          gender $ 1 job_title $ 25
          country $ 2;
   infile "&path/sales.csv" dlm=',';
   input employee_ID first_name $
         last_name $ gender $
         salary job_title $
         country $;
run;
```

Put the LENGTH statement before the INPUT statement.

Program 9.sas

List Input: Compilation

```
data work.subset;
   length first_name $ 12 last_name $18
          gender $ 1 job_title $ 25
          country $ 2;
   infile "&path/sales.csv" dlm=',';
   input employee_ID first_name $
         last_name $ gender $
         salary job_title $
         country $;
run;
```

PDV

First_ Name $ 12	Last _Name $ 18	Gender $ 1	Job_Title $ 25	Country $ 2	Employee_ ID N 8	Salary N 8

SAS scans the input statement looking for additional variables.

Viewing the Output

```
proc print data=work.subset noobs;
run;
```

Partial PROC PRINT Output

first_name	last_name	gender	job_title	country	employee_ID	salary
Tom	Zhou	M	Sales Manager	AU	120102	108255
Wilson	Dawes	M	Sales Manager	AU	120103	87975
Irenie	Elvish	F	Sales Rep. II	AU	120121	26600
Christina	Ngan	F	Sales Rep. II	AU	120122	27475
Kimiko	Hotstone	F	Sales Rep. I	AU	120123	26190

The character values are no longer truncated, but the order of the variables changed.

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;
```

Program 10

Viewing the Output

Partial PROC PRINT Output

employee_id	first_name	last_name	gender	salary	job_title	country
120102	Tom	Zhou	M	108255	Sales Manager	AU
120103	Wilson	Dawes	M	87975	Sales Manager	AU
120121	Irenie	Elvish	F	26600	Sales Rep. II	AU
120122	Christina	Ngan	F	27475	Sales Rep. II	AU
120123	Kimiko	Hotstone	F	26190	Sales Rep. I	AU
120124	Lucian	Daymond	M	26480	Sales Rep. I	AU
120125	Fong	Hofmeister	M	32040	Sales Rep. IV	AU

Business Scenario

Create a temporary SAS data set by reading both standard and nonstandard values from a comma-delimited raw data file.

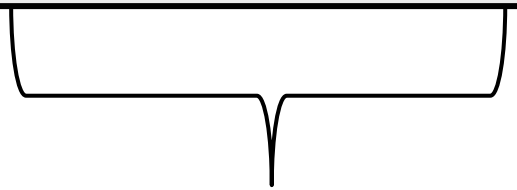
The new data set should contain a subset of the input data and include permanent attributes.

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
```



Tip: We'll need informats to read the last two variables.

Modified List Input – Informats optional here

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 enables up to 12 characters to be read.
- The **:** format modifier tells SAS to read until it encounters a delimiter or to the format length, whichever comes first..

Program 11_good

Modified List Input – What Can Go Wrong?

Omitting the colon modifier causes unexpected results.

Partial **sales.csv**

reads 12 characters

```
120102,Tom,Zhou,M,108255,Sales Manager,AU,11AUG1973,06/01/1993
```

```
input Employee_ID First_Name $12.
      Last_Name :$18. Gender :$1. Salary
      Job_Title :$25. Country :$2.;
```

PDV

Employee_ID N 8	First_Name $ 12	Last_Name $ 18	Gender $ 1
120102	Tom,Zhou,M,1	08255	S

Salary N 8	Job_Title $ 25	Country $ 2
.	11AUG1973	06

Program 11_error_1

Modified List Input

Omitting the colon modifier causes unexpected results.

Partial **sales.csv**

reads 12 characters

```
120102,Tom,Zhou,M,108255,Sales Manager,AU,11AUG1973,06/01/1993
```

```
input Employee_ID First_Name :$12.
      Last_Name :$18. Gender :$1. Salary
      Job_Title :$25. Country :$2.;
```

PDV

Employee_ID N 8	First_Name $ 12	Last_Name $ 18	Gender $ 1
120102	Tom	Zhou	M

Salary N 8	Job_Title $ 25	Country $ 2
108255	Sales Manager	AU

Modified List Input: What could go wrong?

```
data work.sales;
infile datalines dlm=',' pad;
input employee_id first_name :$12. last_name :$18.
      gender :$1 salary job_title :$25. country :$2.
      birth_date :date. hire_date :mmddyy.;
datalines;
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
;
```

Uh oh!! We've accidently switched back to Column Input

employee_id	first_name	last_name	gender	salary	job_title	country	birth_date	hire_date
120102	Tom	Zhou	1	.	Zhou	M	.	.
120103	Wilson	Dawes	1	.	Dawes	M	.	.
120121	Irenie	Elvish	1	.	Elvish	F	.	.

Program11_error_2

Easy Mistake: Fixed!

```
data work.sales;
infile datalines dlm=',' pad;
input employee_id first_name :$12. last_name :$18.
      gender :$1. salary job_title :$25. country :$2.
      birth_date :date. hire_date :mmddyy.;
datalines;
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
;
```

employee_id	first_name	last_name	gender	salary	job_title	country	birth_date	hire_date
120102	Tom	Zhou	M	108255	Sales Manager	AU	4971	12205
120103	Wilson	Dawes	M	87975	Sales Manager	AU	-2535	6575
120121	Irenie	Elvish	F	26600	Sales Rep. II	AU	-4169	6575
120122	Christina	Ngan	F	27475	Sales Rep. II	AU	-523	8217
120123	Kimiko	Hotstone	F	26190	Sales Rep. I	AU	3193	10866

Program 11_good

In Summary – 4 Ways to Define Variable Lengths

1. Use the default length of 8 bytes
2. Use the informat length provided on the Input stmt
3. Use the informat length provided on an Informat stmt
4. Use a stand-alone Length statement
5. Use an Attrib statement

Reading Nonstandard Data

An informat is ***required*** to read nonstandard numeric data.

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
```

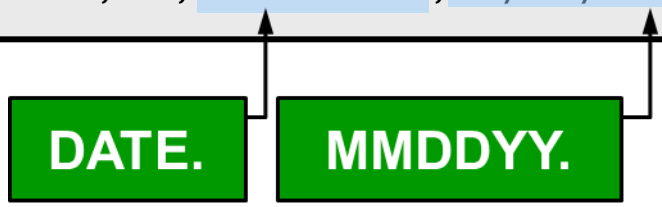
In this example, informats are needed to specify the style of the date fields so that they can be read and converted to SAS dates.

Informats for Nonstandard Data

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

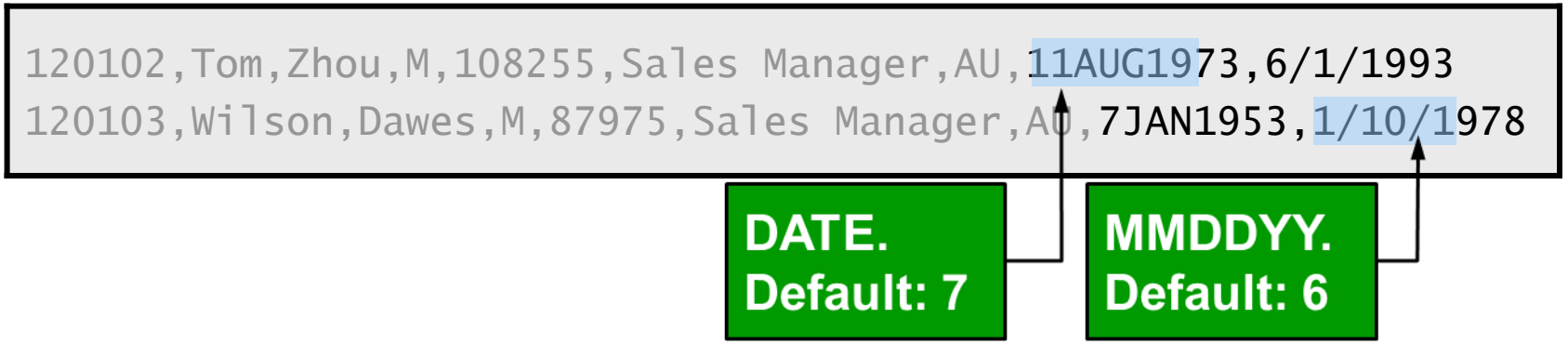
Partial **sales.csv**

```
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
```



The informat describes the data value and tells SAS how to convert it.

Using Informats to Read Nonstandard Data



- An informat is needed to read a nonstandard value.
- The width is optional when you use list input.
- If the width and the colon format modifier are omitted, the default width for that informat is used.

Modified List Input

The colon format modifier (:) tells SAS to read to the length of the informat or to the delimiter – whichever comes first.

```
input Employee_ID First_Name :$12.
      Last_Name :$18. Gender :$1.
      Salary Job_Title :$25. Country :$2.
      Birth_Date :date. Hire_Date:mmddyy.;
```

colon format modifier

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

WHERE versus Subsetting IF Statement

Step and Usage	WHERE	IF
PROC step	Yes	No
DATA step (source of variable)		
SET statement	Yes	Yes
assignment statement	No	Yes
INPUT statement	No	Yes

Additional SAS Statements

Additional SAS statements can be added to perform further processing in the DATA step. But be careful ….!

```
data work.sales;
   infile "&path\sales.csv" dlm=',';
   input Employee_ID First_Name :$12. Last_Name :$18.
         Gender :$1. Salary Job_Title :$25. Country :$2.
         Birth_Date :date. Hire_Date :mmddyy.;
   where Country='AU';
   keep First_Name Last_Name Salary
        Job_Title Hire_Date;
   label Job_Title='Sales Title'
         Hire_Date='Date Hired';
   format Salary dollar12. Hire_Date monyy7.;
run;
```

SAS reads from a raw data file.

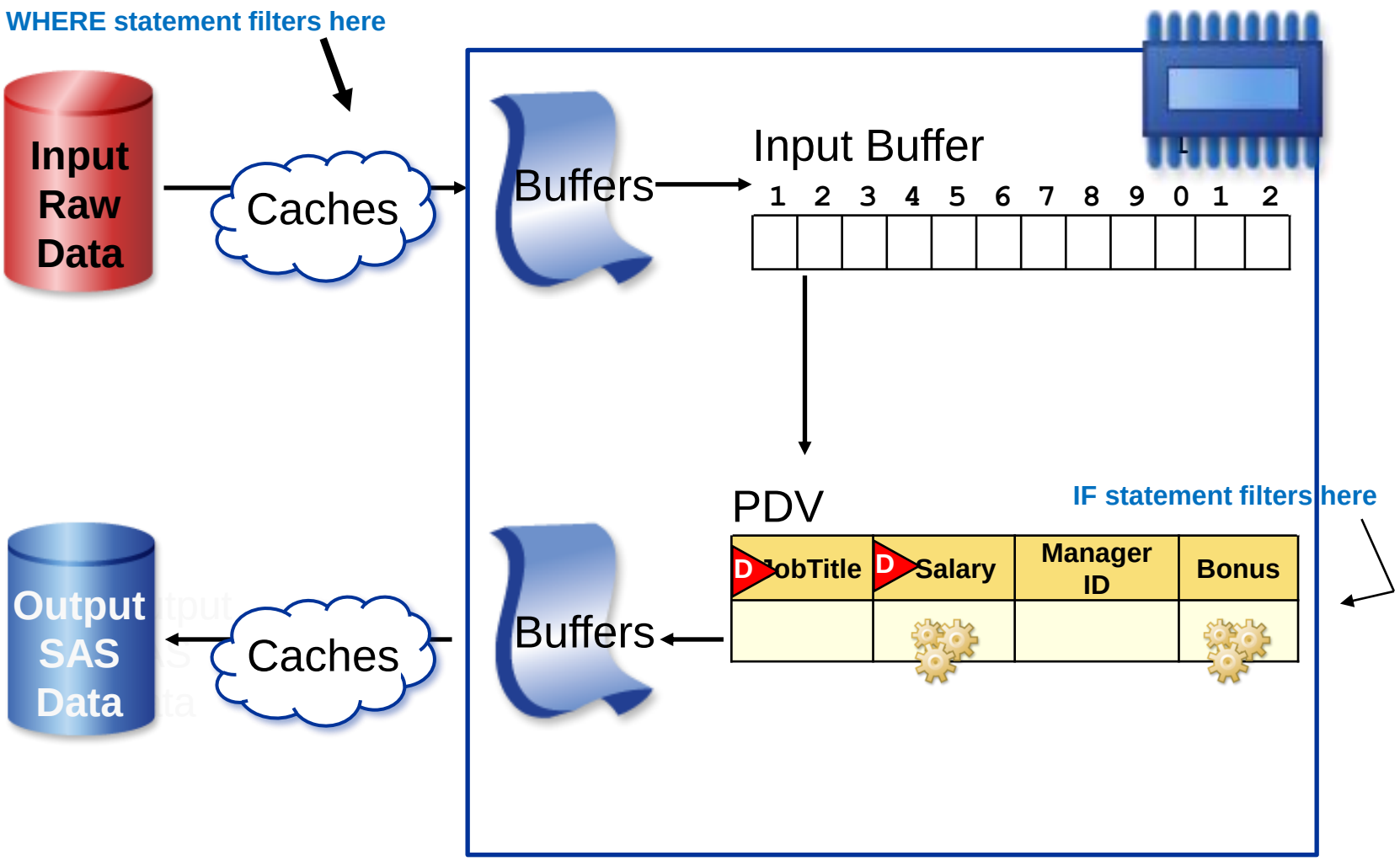
Program 12_error

You may not use a WHERE statement when referring to computed variables or raw data.

```
75          data work.sales;
76             infile datalines dlm=',' pad;
77             input employee_id first_name :$12. last_name :$18.
78                   gender :$1. salary job_title :$25. country :$2.
79                   birth_date :date. hire_date :mmddyy.;
80          where country = 'AU';
ERROR: No input data sets available for WHERE statement.
81          keep first_name last_name salary
82               job_title hire_date;
83          label job_title='Sales Title'
84                hire_date='Date Hired';
85          format salary dollar12. hire_date monyy7.;
```

What's the difference between reading a SAS table vs raw data?

When reading raw data ….



Additional SAS Statements

Additional SAS statements can be added to perform further

```
data work.sales;
   infile "&path\sales.csv" dlm=',';
   input Employee_ID First_Name :$12. Last_Name :$18.
         Gender :$1. Salary Job_Title :$25. Country :$2.
         Birth_Date :date. Hire_Date :mmddyy.;
   if Country='AU';
   keep First_Name Last_Name Salary
        Job_Title Hire_Date;
   label Job_Title='Sales Title'
         Hire_Date='Date Hired';
   format Salary dollar12. Hire_Date monyy7.;
run;
```

A subsetting IF statement is used to subset observations when SAS reads from a raw data file.

Program 12_good

Viewing the Output

```
proc print data=work.sales label;
run;
```

Partial PROC PRINT Output

Obs	first_name	last_name	salary	job_title	country	hire_date
1	Tom	Zhou	$108,255	Sales Manager	AU	JUN1993
2	Wilson	Dawes	$87,975	Sales Manager	AU	JAN1978
3	Irenie	Elvish	$26,600	Sales Rep. II	AU	JAN1978
4	Christina	Ngan	$27,475	Sales Rep. II	AU	JUL1982
5	Kimiko	Hotstone	$26,190	Sales Rep. I	AU	OCT1989

Program 12_good

Modified List Input: What else could go wrong?

List Input with Missing Values - Mid Record

```
data work.contacts;
   length name $20 phone mobile $14;
   infile datalines dlm=',';
   input name $ phone $ mobile $;
datalines;
James Kvarniq,(704) 293-8126,(701) 281-8923
Sandrina Stephano,,(919) 271-4592
Cornelia Krahl,(212) 891-3241,(212) 233-5413
Karen Ballinger,,(714) 644-9090
Elke Wallstab,(910) 763-5561,(910) 545-3421
;
```

proc print data=contacts

Obs	name	phone	mobile
1	James Kvarniq	(704) 293-8126	(701) 281-8923
2	Sandrina Stephano	(919) 271-4592	Cornelia Krahl
3	Karen Ballinger	(714) 644-9090	Elke Wallstab

Program 13_error

Consecutive Delimiters in List Input

List input treats two or more consecutive delimiters as a single delimiter and not as a missing value.

Partial phone2.csv

```
          1    1    2    2    3    3    4    4
1---5----0----5----0----5----0----5----0----5
Sandrina Stephano,, (919) 271-4592
Cornelia Krahl,(212) 891-3241,(212) 233-5413
```

When there is missing data in a record, SAS does the following:

- loads the next record to finish the observation
- writes a note to the log
- SAS wants to fill the PDV!

Consecutive Delimiters in List Input – Use the DSD option …. Much better

```
data work.contacts;
   length name $20 phone mobile $14;
   infile datalines dsd;
   input name $ phone $ mobile $;
datalines;
James Kvarniq,(704) 293-8126,(701) 281-8923
Sandrina Stephano,, (919) 271-4592
Cornelia Krahl,(212) 891-3241,(212) 233-5413
Karen Ballinger,, (714) 644-9090
Elke Wallstab,(910) 763-5561,(910) 545-3421
;
```

name	phone	mobile
James Kvarniq	(704) 293-8126	(701) 281-8923
Sandrina Stephano	(919) 271-4592	Cornelia Krahl
Karen Ballinger	(714) 644-9090	Elke Wallstab

Program 13_good

DSD Option

Use the DSD option to correctly read **phone2.csv**.

```
data work.contacts;
   length Name $ 20 Phone Mobile $ 14;
   infile "&path\phone2.csv" dsd;
   input Name $ Phone $ Mobile $;
run;
```

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

The DSD (Delimiter Sensitive Data) option does the following:

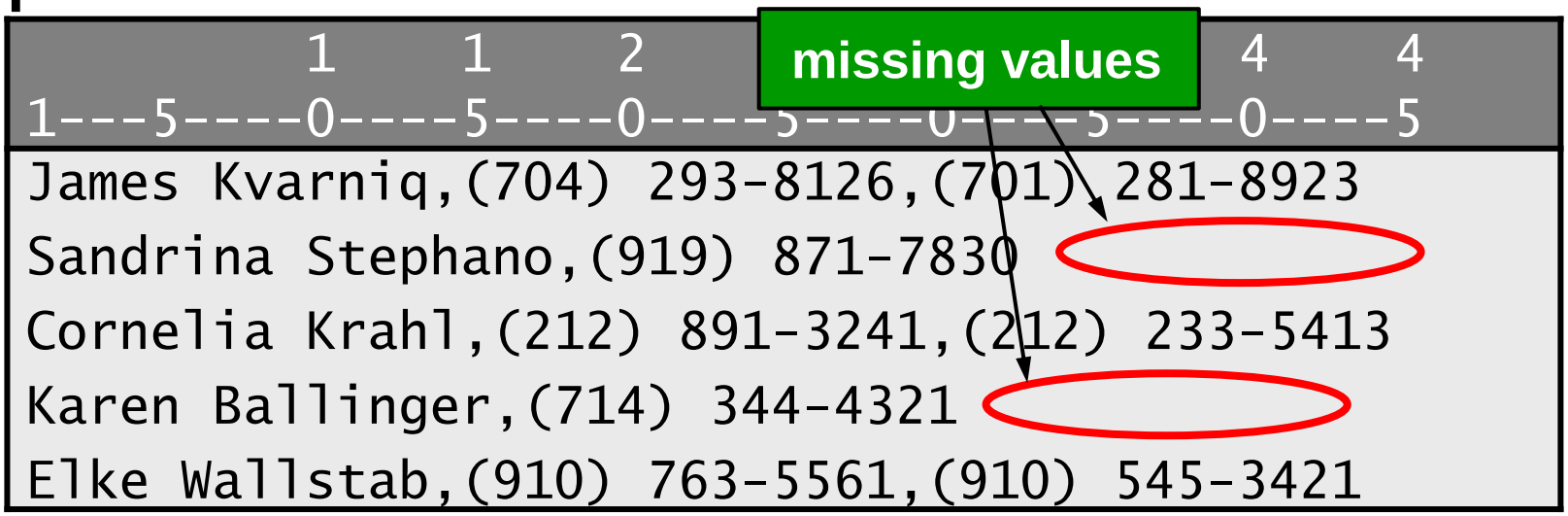
- sets the default delimiter to a comma
- treats consecutive delimiters as missing values
- enables SAS to read values with embedded delimiters if the value is enclosed in quotation marks

Program 13_error

List Input with Missing Values at End of Record

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

phone.csv



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

Missing Data at Record's End

```
data contacts;
   length name $ 20 phone mobile $14;
   infile datalines dlm=',';
   input name $ phone $ mobile $;
datalines;
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
;
```

name	phone	mobile
James Kvarniq	(704) 293-8126	(701) 281-8923
Sandrina Stephano	(919) 871-7830	Cornelia Krahl
Karen Ballinger	(714) 344-4321	Elke Wallstab

Program 14_error

Problem Fixed - MISSOVER Option

The *MISSOVER option* prevents SAS from loading a new record when the end of the current record is reached, setting the missing variable's value to missing.

```
data contacts;
   length name $ 20 phone mobile $14;
   infile datalines dlm=',' missover;
   input name $ phone $ mobile $;
datalines;
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
;
```

name	phone	mobile
James Kvarniq	(704) 293-8126	(701) 281-8923
Sandrina Stephano	(919) 871-7830	
Cornelia Krahl	(212) 891-3241	(212) 233-5413

Program 14_good

Another Use of the Trailing @: List Input with Missing Values at End of Record

```
data vacation_preferences;
  infile datalines dlm=',' missover;
  input name $ (place1-place5) (:$15.) @;
datalines;
Rich,Mallorca,Peru,Uruguay
Michelle,Disneyland,Disneyworld
Pei,Taiwan,Las Vegas,Yosemite,Mexico City,Japan
Earl,Carson City
;
```

Obs	name	place1	place2	place3	place4	place5
1	Rich	Mallorca	Peru	Uruguay		
2	Michelle	Disneyland	Disneyworld			
3	Pei	Taiwan	Las Vegas	Yosemite	Mexico City	Japan
4	Earl	Carson City				

Program 15

Reading Repeated Raw Data Fields

```
data sweet_charity;
    infile datalines;
    input charity :$3. amt :4.;
datalines;
AQI  495  CCI  200  CNI 249
CS  279  CU  780   DAI
875 ES  290  FFC 0  MI 745
SBA 900 V2 550 YYCR 0
;
```

Proc print data=sweet_charity;

Obs	charity	amt
1	AQI	495
2	CS	279
3	875	.
4	SBA	900

We have lost the 2nd and 3rd charity names and contributions on each record.

Program 16_error

Reading Repeated Raw Data Fields

```
data sweet_charity;
   infile datalines;
   input charity :$3. amt :4. @;
datalines;
AQI  495  CCI  200  CNI 249
CS  279  CU  780   DAI
875 ES  290  FFC 0  MI 745
SBA 900 V2 550 YYCR 0
;
```

Remember: the single trailing @ "hold" is released at the step boundary, so ineffective here.

Proc print data=sweet_charity;

Obs	charity	amt
1	AQI	495
2	CS	279
3	875	.
4	SBA	900

Hmm, still not working…

Program 16_error

Reading Repeated Raw Data Fields

```
data sweet_charity;
   infile datalines;
   input charity :$3. amt :4. @@;
datalines;
AQI  495  CCI  200  CNI 249
CS  279  CU  780   DAI
875 ES  290  FFC 0  MI 745
SBA 900 V2 550 YYCR 0
;
```

Obs	charity	amt
1	AQI	495
2	CCI	200
3	CNI	249
4	CS	279
5	CU	780
6	DAI	875
7	ES	290
8	FFC	0
9	MI	745
10	SBA	900
11	V2	550
12	YYC	0

The double trailing @@

- Works same as @ but maintains it's hold across executions of the DATA step.

Program 16_good

The Final Scoop on Trailing @'s

When you use a single trailing @@ (aka, "soft hold")

- Execution of a subsequent INPUT statement without a trailing @ turns off the holding effect of the trailing @
- A new execution of the DATA step turns off the holding effect of the trailing @
- A new record is loaded into the input buffer if the end-of-record is encountered on the current record

The Final Scoop on Trailing @'s

When you use a double trailing @ (aka, "hard hold"):

- Works like the single trailing @ except it also holds the data line in the input buffer across executions of the DATA step
- Usually is used to read multiple SAS observations from a single data line
- Should not be used with the @ pointer control, column input, or the MISSOVER option
- Execution of a subsequent INPUT statement without a double trailing @ turns off the holding effect of the double trailing @
- A new record is loaded into the input buffer if the end-of-record is encountered on the current record

INFILE Options

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

Option	Description
DLM=	Specifies an alternate delimiter.
DSD	Sets the default delimiter to a comma, treats consecutive delimiters as missing values, and allows embedded delimiters when the data value is enclosed in quotation marks.
MISSOVER	Sets variables to missing if the end of the record is reached before finding values for all fields.

In Summary: Default Rules for Using List Input

1) The file must have a delimiter. Otherwise, SAS cannot possibly know where a variable's value begins and ends.

2) The default delimiter is a blank. Use the DLM= option to specify other field delimiters.

3) Consecutive delimiters can separate variables – meaning consecutive delimiters count as a single delimiter.

4)Use a $ after a variable name for character fields.

5)All variables have a default length of 8 bytes. Field width can be greater than 8 with truncation resulting for character variable. To change:

a. Use a LENGTH statement (eg., length name $20)
b. Use an INFORMAT statement (eg., informat salary comma8.)
c. Use a format modifier (eg, input salary : comma8.)

6) By default you can read only standard character or numeric data, unless you supply an informat. This can be done via the:

a. Format modifier
b. INFORMAT statement

7)The INPUT statement reads as many fields as needed to fill the PDV

8) You must read fields in order, left to right.

9) By default, you must specify missing data values (character or numeric) with a period. If no period supplied, use the DSD option on a INFILE statement to treat consecutive delimiter as a missing value.

Reading Raw Data: Additional Resources

- Type “how to read raw data into a SAS dataset” in any browser
- YouTube Channel
- communities.sas.com
- A supplementary case study at the end of the “SAS Programming: Essentials” class
- The SAS Book Store: https://support.sas.com/en/software/base-sas-support.html#documentation
- The SAS Help Center: https://go.documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lepg/n02z8w5et43i1wn1eto968a6zrpi.htm
-