Tips & Tricks
Understanding Data Step Processing

M. Brüning – Boehringer Ingelheim Pharma GmbH & Co. KG

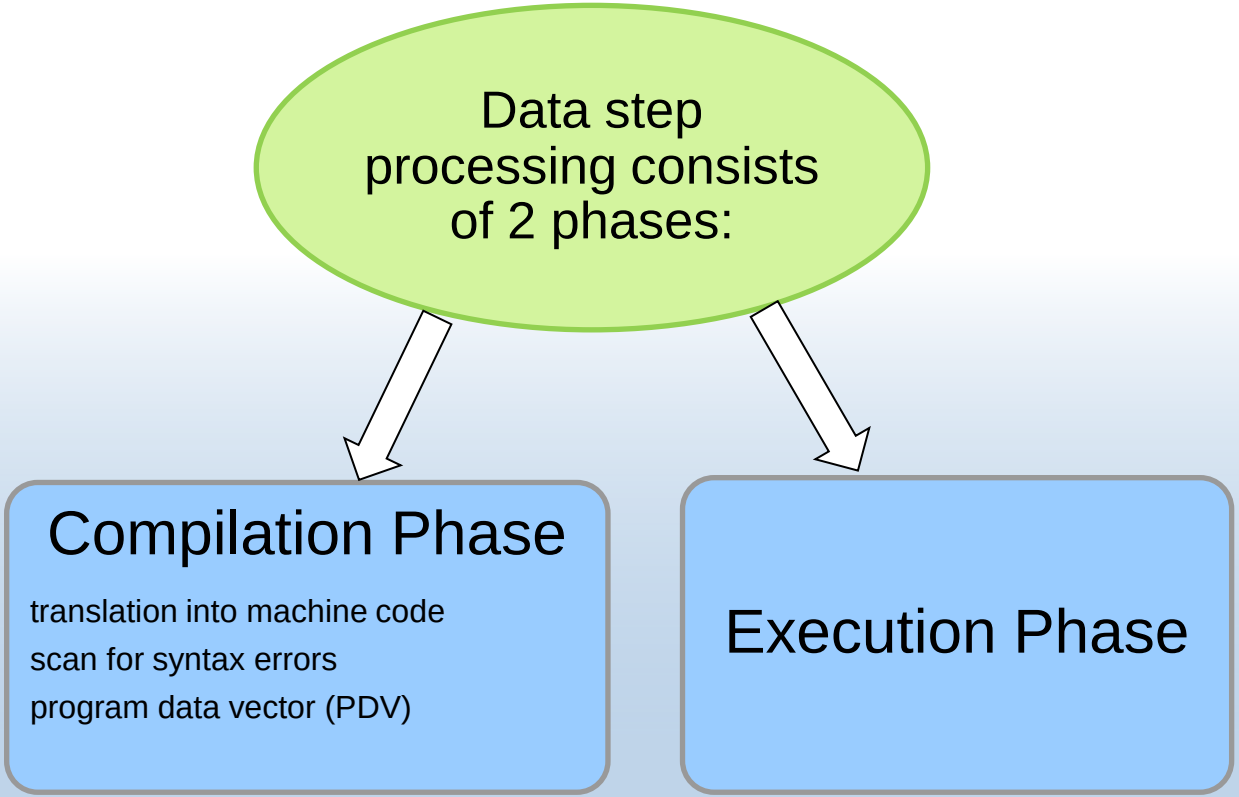
KSFE 2022

Boehringer Ingelheim

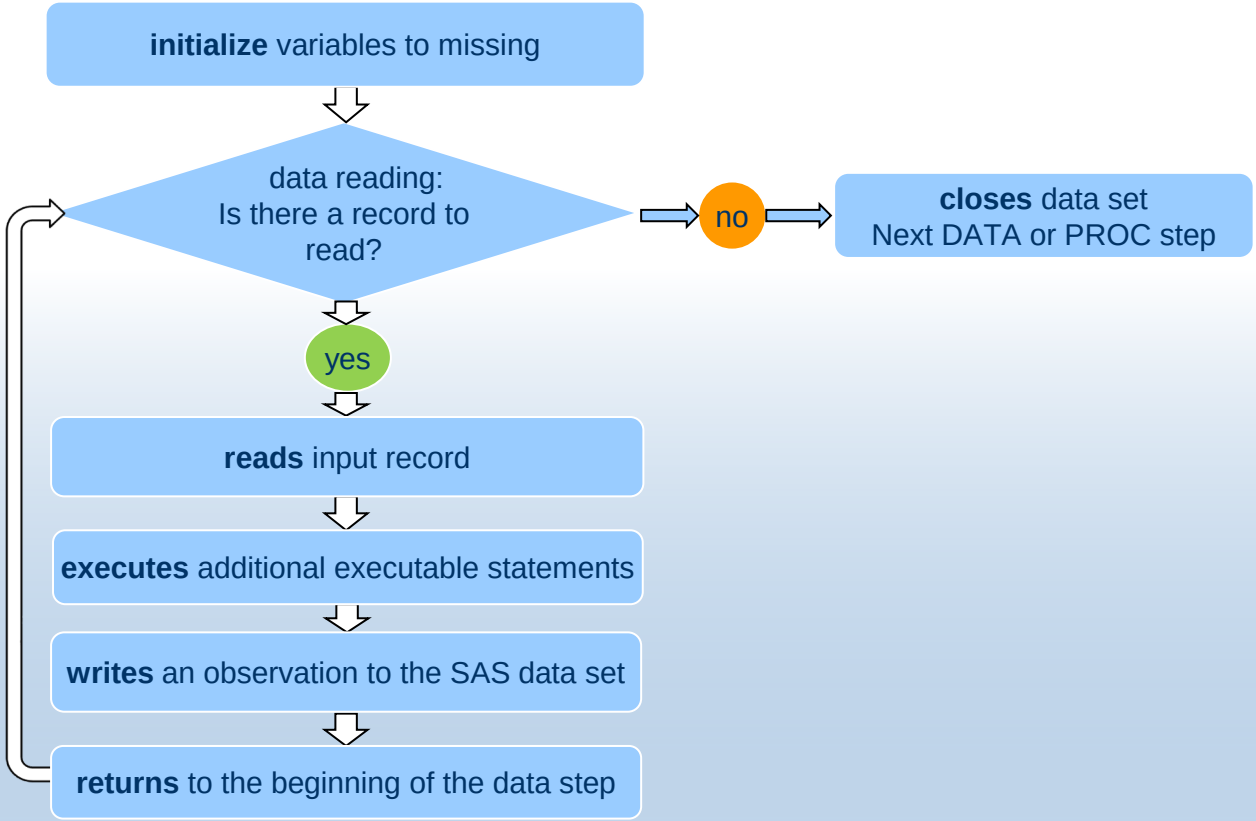
Agenda

1. • Data Step Processing
2. • Vertical dataset concatenation
3. • Multiple sets
4. • Many to many relationship
5. • LAG and DIF functions

Data step processing



Execution phase

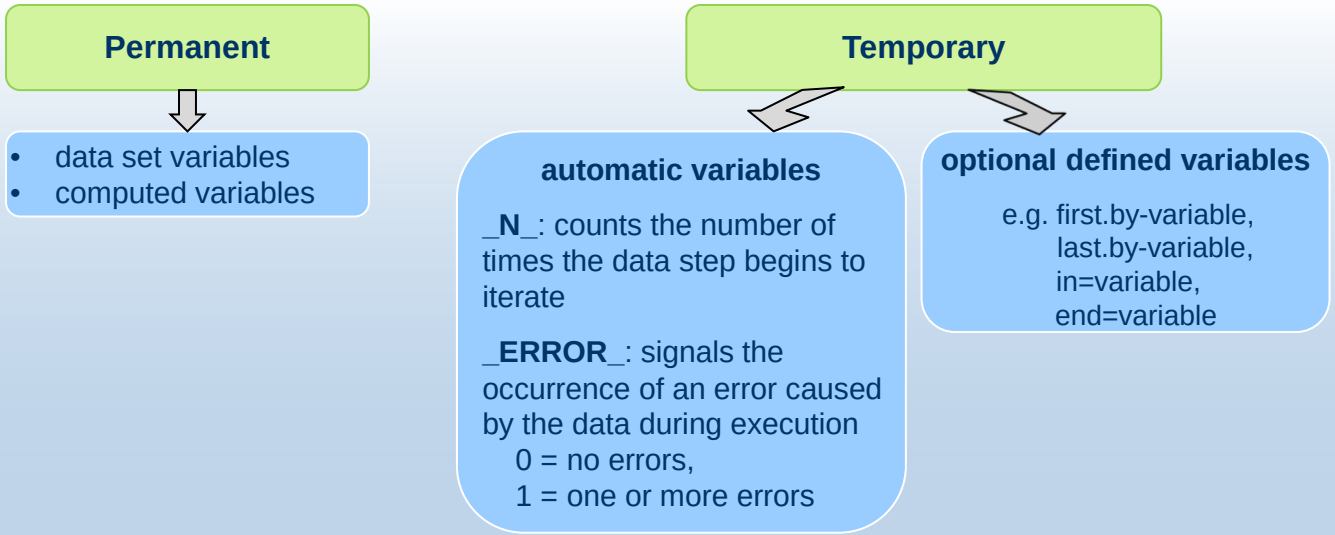


Program Data Vector - PDV

The **Program Data Vector** is a logical area of memory that is created during the data step processing.

SAS builds a SAS dataset by reading one observation at a time into the PDV and, unless given code to do otherwise, writes the observation to a target dataset.

The program data vector contains two types of variables.



Vertical dataset concatenation
SET statement with and without BY statement

male: sorted by name

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female: sorted by name

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data male_female;
  set male female;
run;
```

```
data male_female_by_name;
  set male female;
  by name;
run;
```

male_female: sorted by sex, name

	Name	Sex	Age	Height
1	James	M	12	.
2	Jeffrey	M	13	.
3	John	M	12	.
4	Robert	M	12	.
5	Thomas	M	11	.
6	Jane	F	.	59.8
7	Janet	F	.	62.5

male_female_by_name: sorted by name

	Name	Sex	Age	Height
1	James	M	12	.
2	Jane	F	.	59.8
3	Janet	F	.	62.5
4	Jeffrey	M	13	.
5	John	M	12	.
6	Robert	M	12	.
7	Thomas	M	11	.

Multiple sets

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
run;
```

```
data multiple2;
  set female; set male;
run;
```

multiple1

	Name	Sex	Age	Height
1	Jane	F	12	59.8
2	Janet	F	13	62.5

multiple2

	Name	Sex	Height	Age
1	James	M	59.8	12
2	Jeffrey	M	62.5	13

multiple sets

- SAS encounters the end of file marker on the smallest file
- reads the first observation from 1st dataset into Program Data Vector (PDV)
- reads the first observation from 2nd dataset into Program Data Vector (PDV)
- SAS overwrites values of common variables from 1st dataset with new values from 2nd dataset;

Multiple sets – PDV - Compilation phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_					
1	0					

1. SAS creates a PDV containing the automatic variables _N_ and _ERROR_
2. SAS scans each statement for syntax errors e.g. missing semicolons, invalid statements

Multiple sets – PDV - Compilation phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newar=1;
  if sex=„F“ then newar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age		
1	0					

1. SAS creates a PDV containing the automatic variables _N_ and _ERROR_
2. SAS scans each statement for syntax errors e.g. missing semicolons, invalid statements
3. While compiling SAS adds a position to the PDV for each variable
   - in the 1st input dataset

Multiple sets – PDV - Compilation phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newar=1;
  if sex=„F“ then newar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	
1	0					

1. SAS creates a PDV containing the automatic variables _N_ and _ERROR_
2. SAS scans each statement for syntax errors e.g. missing semicolons, invalid statements
3. While compiling SAS adds a position to the PDV for each variable
   - in the 1st input dataset
   - in the 2nd input dataset

Multiple sets – PDV - Compilation phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0					

1. SAS creates a PDV containing the automatic variables _N_ and _ERROR_
2. SAS scans each statement for syntax errors e.g. missing semicolons, invalid statements
3. While compiling SAS adds a position to the PDV for each variable
   - in the 1st input dataset
   - in the 2nd input dataset
   - that is created in the data step

Multiple sets – PDV - Compilation phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0					

1. SAS creates a PDV containing the automatic variables _N_ and _ERROR_
2. SAS scans each statement for syntax errors e.g. missing semicolons, invalid statements
3. While compiling SAS adds a position to the PDV for each variable
   - in the 1st input dataset
   - in the 2nd input dataset
   - that is created in the data step
4. SAS completes the compile phase at the bottom of the data step. The output data set does not yet contain any observations, because SAS has not yet begun executing the program.

Multiple sets – PDV - Execution phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0			.	.	.

1. The data step executes once for each observation in the input data set
2. At the beginning of the execution phase SAS sets all data set variables in the PDV to missing

Multiple sets – PDV - Execution phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0	James	M	12	.	.

1. The data step executes once for each observation in the input data set (5 times for „male“)
2. At the beginning of the execution phase SAS sets all data set variables in the PDV to missing
3. The SET statement reads the first observation and writes the values to the PDV
   - 1st dataset

Multiple sets – PDV - Execution phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0	Jane	F	12	59.8	.

1. The data step executes once for each observation in the input data set (5 times for „male“)
2. At the beginning of the execution phase SAS sets all data set variables in the PDV to missing
3. The SET statement reads the first observation and writes the values to the PDV
   - 1st dataset
   - 2nd dataset

Multiple sets – PDV - Execution phase

male

	Name	Sex	Age
1	James	M	12
2	Jeffrey	M	13
3	John	M	12
4	Robert	M	12
5	Thomas	M	11

female

	Name	Sex	Height
1	Jane	F	59.8
2	Janet	F	62.5

```
data multiple1;
  set male; set female;
  if sex=„M“ then newvar=1;
  if sex=„F“ then newvar=2;
run;
```

	Name	Sex	Age	Height	newvar
1	Jane	F	12	59.8	2
2	Janet	F	13	62.5	2

_N_	_ERROR_	Name	Sex	Age	Height	newvar
1	0	Jane	F	12	59.8	2

1. The data step executes once for each observation in the input data set (5 times for „male“)
2. At the beginning of the execution phase SAS sets all data set variables in the PDV to missing
3. The SET statement reads the first observation and writes the values to the PDV
   - 1st dataset
   - 2nd dataset
4. The assignment statement executes to compute the first value of *„newvar”*

Many to many relationship - Execution phase

treatment

	animalno	treatdc
1	1	A
2	2	A
3	2	B

	animalno	pardc
1	1	HR
2	1	MAP
3	2	HR
4	2	MAP

paramete
r

```
data many_to_many_failed;
  merge treatment parameter;
  by animalno;
run;
```

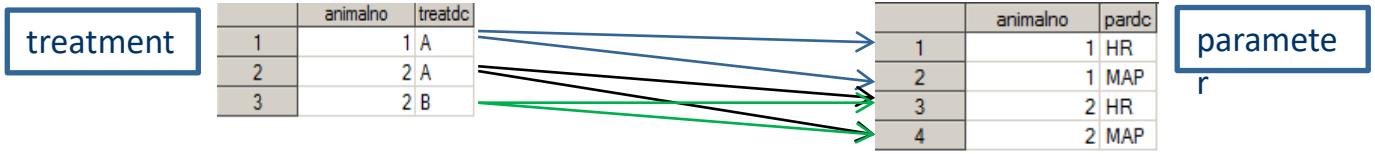
_N_	_ERROR_	animalno	treatdc	pardc
1	0	1	A	HR
2	0	1	A	MAP
3	0	2	A	HR
4	0	2	B	MAP

!!! merge does not work !!!

```
NOTE: MERGE statement has more than one data set with repeats of
BY values.
```

SAS reads observations sequentially and once an observation is read into the PDV, it is never re-read

Many to many relationship - Execution phase



Solution with DATA STEP

nobs -> number of observations in the dataset
point -> retrieving the i-th observation

```
data many_to_many_ok(drop=_animalno);
  set treatment;
  do i=1 to num; /* nobs=4 in parameter */
    set parameter
       (rename=(animalno=_animalno))
        nobs = num point = i;
    if animalno=_animalno then output;
  end;
run;
```

Solution with PROC SQL

```
proc sql noprint;
  create table many_to_many_sql as
    select treatment.*, parameter.pardc
    from treatment as t
    full join parameter as p
    on t.animalno=p.animalno;
quit;
```

_N_	_ERROR_	animalno	treatdc	pardc
1	0	1	A	HR
2	0	1	A	MAP
3	0	2	A	HR
4	0	2	A	MAP
5	0	2	B	HR
6	0	2	B	MAP

LAG function

LAG<n>(argument)

- **n -> number of lagged values**
- **argument -> number or character**

- **Each occurrence of a LAG*n* function generates its own queue**
- *n* is the length of the queue
- the LAG function is executable
- storing values at the bottom of the queue and returning values from the top of the queue occurs **only when the function is executed**
- a LAG*n* function that is executed **conditionally** will store and return only values from the observations for which the condition is satisfied.

LAG function - DIF function

	visit	result	subjid
1	0	80	1
2	1	85	1
3	2	87	1
4	3	90	1
5	4	88	1

```
data lag_dif;
  set lag_dif_data;
  lag1result = lag(result);  dif1result = dif1(result);
  lag2result = lag2(result); dif2result = dif2(result);
  lag3result = lag3(result); dif3result = dif3(result);
run;
```

	visit	result	subjid	lag1result	dif1result	lag2result	dif2result	lag3result	dif3result
1	0	80	1	.	.	.	.	.	.
2	1	85	1	80	5	.	.	.	.
3	2	87	1	85	2	80	7	.	.
4	3	90	1	87	3	85	5	80	10
5	4	88	1	90	-2	87	1	85	3

LAG function - condition

phase

	animalno	phaseno	startdate
1	1	2	14OCT2012
2	1	1	04OCT2012
3	2	3	19JUN2012
4	2	2	29MAY2012
5	2	1	02MAY2012

What we want to do: each phase ends with start of previous phase. Calculate end date of all phases except last phase. end date of last phase is empty

```
data phase_end_failed;
  set phase;
  by animalno descending phaseno startdate;

  if first.animalno then enddate=.;
  else enddate = lag(startdate);
run;
```

!!! Lag within a condition !!!

	animalno	phaseno	startdate	enddate
1	1	2	14OCT2012	.
2	1	1	04OCT2012	.
3	2	3	19JUN2012	.
4	2	2	29MAY2012	04OCT2012
5	2	1	02MAY2012	29MAY2012

lag within a condition -> only observations which fulfill the condition are used for the lag function

LAG function - condition

```
data phase_end_ok;
  set phase;
  by animalno descending phaseno startdate;

  lagdate = lag(startdate);

  if first.animalno then do;
     enddate=.;
     lagif=lag(startdate);
  end;
  else do;
    enddate=lagdate;
    lagelse=lag(startdate);
  end;
run;
```

	animalno	phaseno	startdate
1	1	2	14OCT2012
2	1	1	04OCT2012
3	2	3	19JUN2012
4	2	2	29MAY2012
5	2	1	02MAY2012

	animalno	phaseno	startdate	lagdate	enddate	lagif	lagelse	
1	1	2	14OCT2012	.	.	.	.	if
2	1	1	04OCT2012	14OCT2012	14OCT2012	.	.	else
3	2	3	19JUN2012	04OCT2012	.	14OCT2012	.	if
4	2	2	29MAY2012	19JUN2012	19JUN2012	.	04OCT2012	else
5	2	1	02MAY2012	29MAY2012	29MAY2012	.	29MAY2012	else

LAG function “looks back”. How to "look ahead“

MERGE the SAS data set with itself, using a one-on-one MERGE with no BY statement

```
data phase_end_ok2;
  merge phase_ascending
        phase_ascending(firstobs=2
                        rename=(startdate = nextdate)
                        keep=startdate);

  ******** no BY statement ********;

run;

data duration2;
  set phase_end_ok2;
  by animalno;

  if not last.animalno then duration = nextdate - startdate;
run;
```

	animalno	phaseno	startdate	nextdate	duration
1	1	1	04OCT2012	14OCT2012	10
2	1	2	14OCT2012	02MAY2012	.
3	2	1	02MAY2012	29MAY2012	27
4	2	2	29MAY2012	19JUN2012	21
5	2	3	19JUN2012	.	.

LAG function – condition: „if“ and „where“

	animalno	phaseno	startdate
1	1	2	14OCT2012
2	1	1	04OCT2012
3	2	3	19JUN2012
4	2	2	29MAY2012
5	2	1	02MAY2012

```
data phase_end_if;
  set phase;
  lagdate = lag(startdate);
  if animalno=2;
  format lagdate date9.;
run;
```

	animalno	phaseno	startdate	lagdate
1	2	3	19JUN2012	04OCT2012
2	2	2	29MAY2012	19JUN2012
3	2	1	02MAY2012	29MAY2012

All observations are read into PDV, the subsetting **IF** is executed against each input observation, and the LAGged value of startdate will be captured.

```
data phase_end_if;
  set phase;
  lagdate = lag(startdate);
  where animalno=2;
  format lagdate date9.;
run;
```

	animalno	phaseno	startdate	lagdate
1	2	3	19JUN2012	.
2	2	2	29MAY2012	19JUN2012
3	2	1	02MAY2012	29MAY2012

WHERE is executed before any observations are read into PDV, there are no LAGged values for the first observation selected.

Thank you!