SESUG 2016
October 16th - 18th
Bethesda, Maryland

CC-171

Some _FILE_ Magic

Mike Zdeb
University at Albany School of Public Health

INFILE

- in a data step with an INPUT statement, SAS creates an input buffer where it holds your data prior to moving the values of variables into the program data vector

- you can access the contents of that buffer using the variable name _INFILE_

- _INFILE_ is an automatic variable whose value is accessible within a data step but is not output to any data set being created in the data step

INFILE

```
data names;
input @; ①
_infile_ = upcase(_infile_); ②
input name :$10. age city :$10. state :$2.; ③
datalines;
mike 25 albany ny
Sara 15 Washington DC
;
```

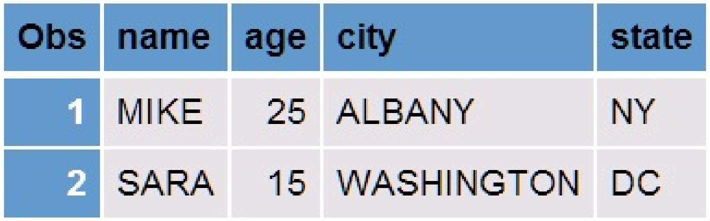
an INPUT statement with no variable names places a record from the DATALINES file into the input buffer ① ... the contents of the buffer is named _INFILE_ and all text in the buffer is converted to uppercase ② ... variables are read from the contents of the input buffer ③ that was "held" by the @ ① in the first INPUT statement

INFILE

original data ...

```
datalines;
mike 25 albany ny
Sara 15 Washington DC
;
```

data set ...



Obs	name	age	city	state
1	MIKE	25	ALBANY	NY
2	SARA	15	WASHINGTON	DC

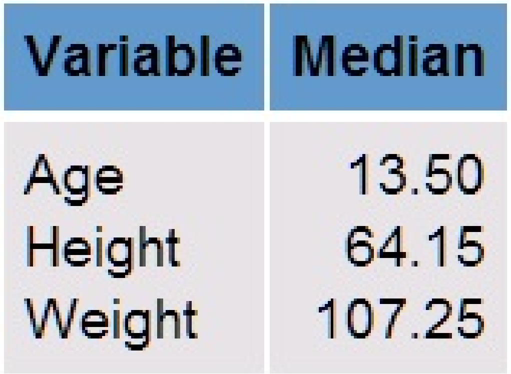
- in addition to SAS documentation, there are several papers describing uses of _INFILE_
- output buffer and _FILE_ discussed in SAS documentation, but cannot find any papers that describe possible uses of _FILE_ ... examples are used to show some "possibilities"

FILE

EXAMPLE #1 ... add a variable to a data set showing which values of numeric variables are below or at/above the median

data set SASHELP.CLASS has three numeric variables ... age (years), height (inches), weight (pounds) ... find median values for MALES

```
proc means data=sashelp.class
maxdec=2 median;
where sex eq 'M';
run;
```



Variable	Median
Age	13.50
Height	64.15
Weight	107.25

FILE

```
proc format; ①
value ag low-<13.5   = '0'  other = '1';
value ht low-<64.15  = '0'  other = '1';
value wt low-<107.25 = '0'  other = '1';
run;

filename nosee dummy; ②
```

three FORMATS are created that will divide values into those below and those at/above the median ① ... a FILENAME statement assigns the FILEREF NOSEE to the device type DUMMY ② (use of DUMMY specifies that any output written to the FILEREF NOSEE is discarded)

FILE

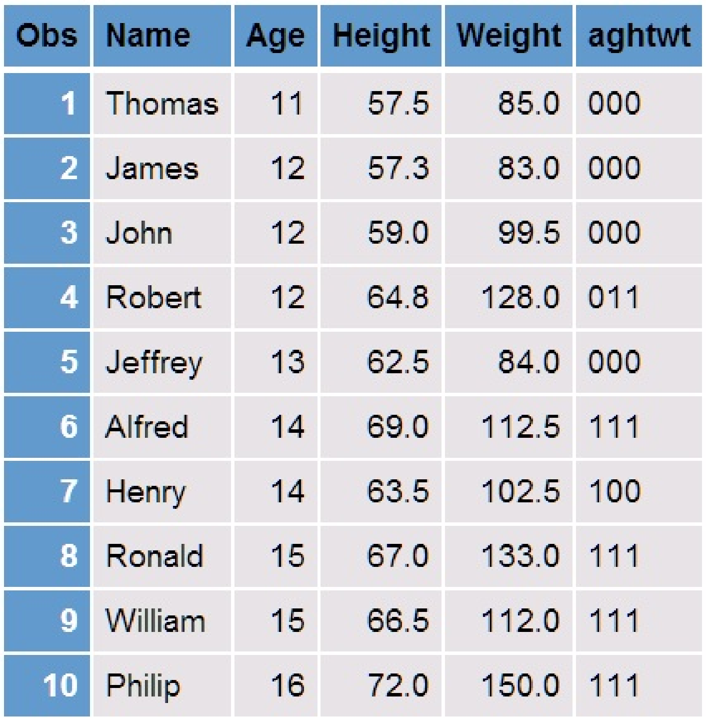
```
data males (drop=sex);
file nosee; ①
set sashelp.class (where=(sex eq 'M')); ②
put age ag. height ht. weight wt. @;  ③
aghtwt = _file_; ④
put; ⑤
run;
```

a FILE statement directs the output from PUT statements to FILEREF NOSEE ① ... observations for MALES are read ② ... a PUT statement with a trailing @ writes the formatted values of variables to the output buffer ③ (the @ holds the values in the buffer) ... the contents of the buffer has the variable name _FILE_ and it is assigned to the variable AGHTWT ④ ... the buffer is cleared with PUT ⑤

FILE

data set MALES

Thomas is below the median for all three variables, Robert is at/above the median for both HEIGHT and WEIGHT, *Henry* is at/above the median for only AGE, *William* is above the median for all three variables, etc.



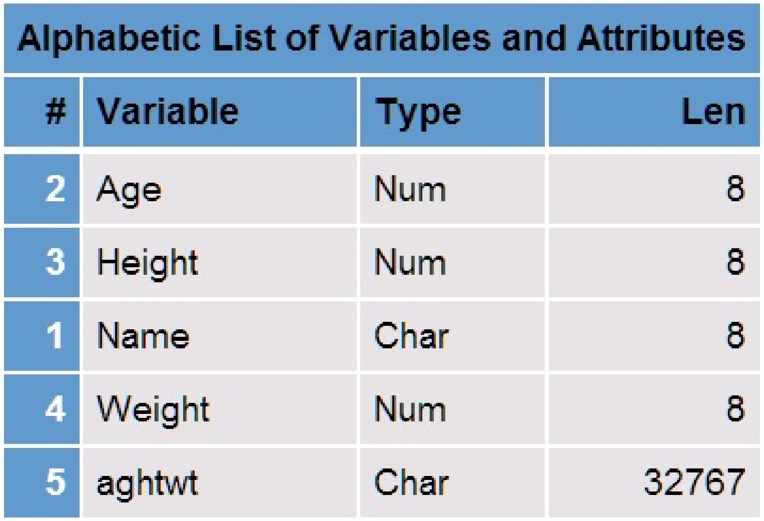
Obs	Name	Age	Height	Weight	aghtwt
1	Thomas	11	57.5	85.0	000
2	James	12	57.3	83.0	000
3	John	12	59.0	99.5	000
4	Robert	12	64.8	128.0	011
5	Jeffrey	13	62.5	84.0	000
6	Alfred	14	69.0	112.5	111
7	Henry	14	63.5	102.5	100
8	Ronald	15	67.0	133.0	111
9	William	15	66.5	112.0	111
10	Philip	16	72.0	150.0	111

- "aha" moment ... PUT with @ and _FILE_ allow you to easily CONCATENATE the FORMATTED VALUES of variables

FILE

what is the LENGTH of the new variable AGHTWT ...

```
aghtwt = _file_;
```



Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
2	Age	Num	8
3	Height	Num	8
1	Name	Char	8
4	Weight	Num	8
5	aghtwt	Char	32767

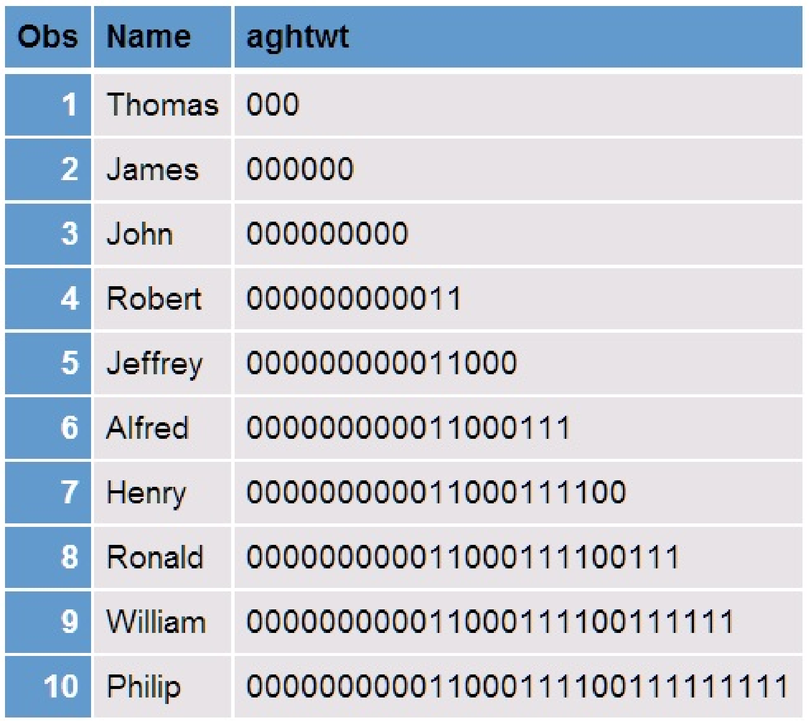
the new variable is the same length as that of the output buffer

when creating a new variable using _FILE_ you should add a LENGTH statement to the data step ...

```
length aghtwt $3;
```

FILE

without the second PUT (the one without the @), the output buffer is never cleared and the PUT statement with an @ keeps adding values to the output buffer (the variable _FILE_)



Obs	Name	aghtwt
1	Thomas	000
2	James	000000
3	John	000000000
4	Robert	000000000011
5	Jeffrey	000000000011000
6	Alfred	000000000011000111
7	Henry	000000000011000111100
8	Ronald	000000000011000111100111
9	William	000000000011000111100111111
10	Philip	000000000011000111100111111111

```
data males (drop=sex);
file nosee;
set sashelp.class (where=(sex eq 'M'));
put age ag. height ht. weight wt. @;
aghtwt = _file_;
run;
```

FILE

same result, less SAS code ...

```
data males (drop=sex);
set sashelp.class (where=(sex eq 'M'));
put @1 age ag. height ht. weight wt. @; ①
aghtwt = _file_;
run;
```

the data step has NO FILE statement, thus all PUT statements write to the LOG ... however, a PUT statement with an @ holds the PUT statement results in the output buffer and does not write to the LOG ① ... there is NO second PUT statement without the @ to clear the buffer since the PUT @1 always writes values to columns 1 through 3 ①

FILE

look at the log …

```
344   data males (drop=sex);
345   set sashelp.class (where=(sex eq 'M'));
346   put @1 age ag. height ht. weight wt. @;
347   aghtwt = _file_;
348   run;
```

```
111
```

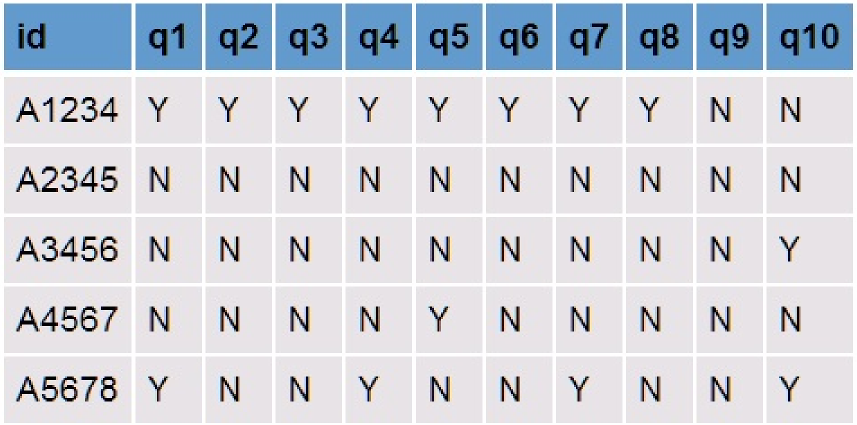
one line written to the LOG (from the last observation … clears the output buffer)

```
NOTE: There were 10 observations read from the
data set SASHELP.CLASS. WHERE sex='M';
NOTE: The data set WORK.MALES has 10 observations
and 5 variables.
```

FIND, WHICHC, IN OPERATOR

EXAMPLE #2 ... searching for variable values

simple task ... given data set ANSWERS ... find observations with at least one answer that is "Y"



id	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10
A1234	Y	Y	Y	Y	Y	Y	Y	Y	N	N
A2345	N	N	N	N	N	N	N	N	N	N
A3456	N	N	N	N	N	N	N	N	N	Y
A4567	N	N	N	N	Y	N	N	N	N	N
A5678	Y	N	N	Y	N	N	Y	N	N	Y

data set ANSWERS

FIND, WHICHC, IN OPERATOR

```
data atleast1y;
set answers;
if find(catt(of q:),'Y');
run;
```

concatenate values of variables q1-q10 and use FIND to search for "Y"

```
data atleast1y;
set answers;
if whichc('Y', of q:);
run;
```

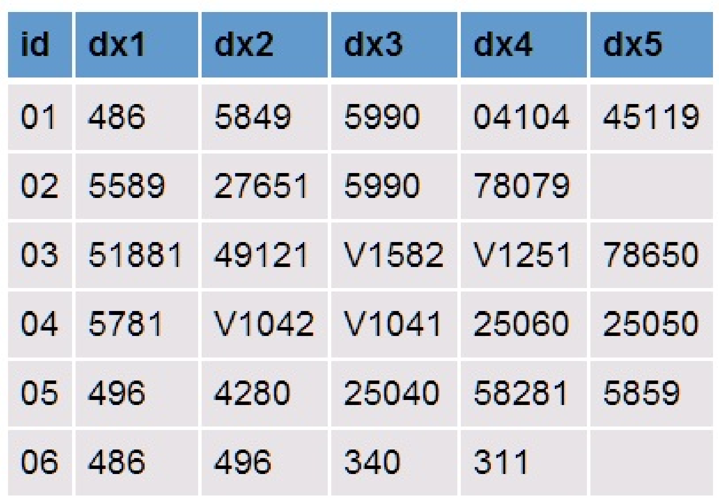
use WHICHC function see if any value of q1-q10 is equal to "Y"

```
data atleast1y;
set answers;
array q(10);
if 'Y' in q;
run;
```

use an IN operator to search an array (values q1-q10) for "Y"

IN OPERATOR+COLON MODIFIER

more complex task ... given data set DIAGNOSES ... find observations with at least one diagnosis that STARTS with the string "250" (diabetes) ... complex with CAT and FIND functions, cannot use WHICHC, easy with IN operator



id	dx1	dx2	dx3	dx4	dx5
01	486	5849	5990	04104	45119
02	5589	27651	5990	78079	
03	51881	49121	V1582	V1251	78650
04	5781	V1042	V1041	25060	25050
05	496	4280	25040	58281	5859
06	486	496	340	311	

data set DIAGNOSES

```
data diabetes;
set diagnoses;
array dx(5);
if '250' in : dx;
run;
```

the colon modifier after the IN operator limits the search to the first three characters of the various diagnoses

FILE

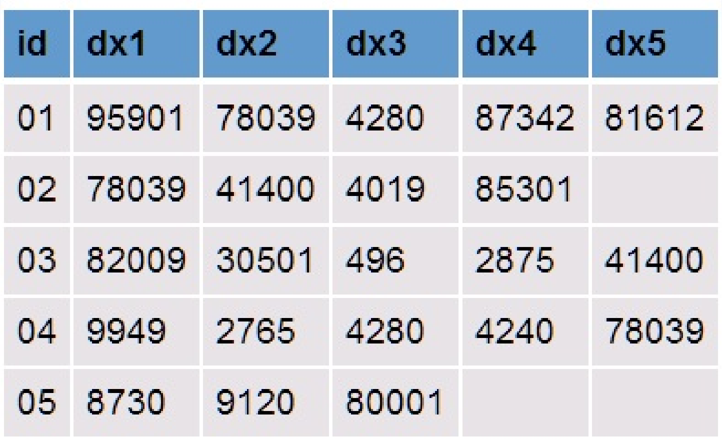
task common to both simple search (look for variables with a value of "Y") and more complex search (look for variables with a value that starts with "250") ... SEARCH MANY VARIABLES for a SINGLE VALUE

what about SEARCHING MANY VARIABLES for MANY VALUES, the equivalent of ...

```
if <many values> in <many variables>;
```

FILE

search for diabetes was a search for one value ... "250"



id	dx1	dx2	dx3	dx4	dx5
01	95901	78039	4280	87342	81612
02	78039	41400	4019	85301	
03	82009	30501	496	2875	41400
04	9949	2765	4280	4240	78039
05	8730	9120	80001		

data set DIAGNOSES ... find TBI

search for traumatic brain injury (TBI) is a search for multiple values ... "800"-"80199", "803"-"80499" , "850"-"85419", "9501"-"95039", "95901", "99555" ... for the first observation in data set DIAGNOSES, that would look like ...

```
if <95901, 78039, 4280, 87342, 81612> in
<800-80199,  803-80499 , 850-85419,  9501-95039,
95901, 99555>;
```

FILE

solution with a FORMAT, _FILE_, and FIND

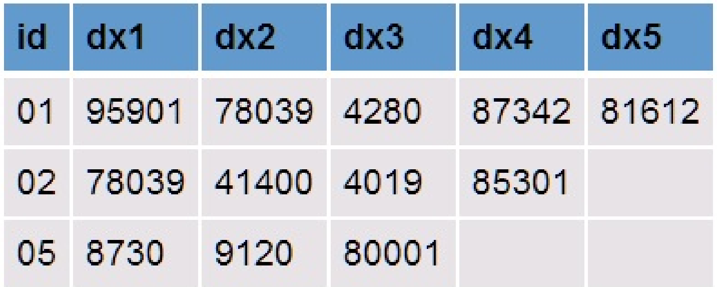
first, create a FORMAT with ranges and individual values that indicate TBI ...

```
proc format;
value $tbi
'800'-'80199', '803'-'80499' , '850'-'85419',
'9501'-'95039', '95901' , '99555' = '1'
other = '0' ;
run;
```

FILE

next, use the FORMAT in a data step ...

```
data tbi;
set diagnoses;
put @1 (dx1-dx5) ($tbi.) @; ①
if find(_file_,'1'); ②
run;
```



id	dx1	dx2	dx3	dx4	dx5
01	95901	78039	4280	87342	81612
02	78039	41400	4019	85301	
05	8730	9120	80001		

data set TBI

a PUT statement writes a string of 1s and 0s to the output buffer (formatted values of the diagnoses, 1 indicates TBI) ① ... a FIND function looks for 1s in the output buffer ②

FILE

EXAMPLE #3 ... search for variable values (TBI) and add a variable (values 1, 0, X) that indicates if a diagnosis is TBI, not TBI, or missing ... a combination of examples #1 and #2

```
proc format;
value $tbi
'800'-'80199', '803'-'80499' , '850'-'85419',
'9501'-'95039', '95901' , '99555' = '1' ①
other = '0' ②
' ' ='X'; ③
run;
```

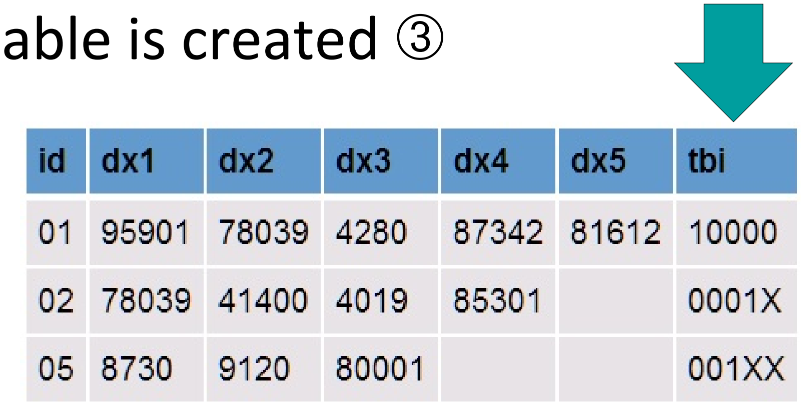
FORMAT differentiates among TBI ①, not TBI ②, and missing ③

FILE

```
data tbi;
length tbi $5; ①
set diagnoses;
put @1 (dx1-dx5) ($tbi.) @; ②
if find(_file_,'1'); ②
tbi = _file_; ③
run;
```

a LENGTH statement sets the length of the new variable TBI ① ... a PUT statement writes a string of 1s and 0s to the output buffer and FIND locates TBI ② ... a new variable is created ③

locations of TBI diagnoses indicated with 1s in variable TBI



id	dx1	dx2	dx3	dx4	dx5	tbi
01	95901	78039	4280	87342	81612	10000
02	78039	41400	4019	85301		0001X
05	8730	9120	80001			001XX

CONCLUSION

- VARIOUS USES OF THE CONTENTS OF THE INPUT BUFFER CREATED WITH AN INPUT STATEMENT AND ACCESSED VIA THE VARIABLE _INFILE_ HAVE BEEN SHOWN IN SEVERAL PAPERS

- THIS PRESENTATION (AND PAPER) DEMONSTRATE THAT THE CONTENTS OF THE OUTPUT BUFFER CREATED WITH A PUT STATEMENT CAN BE ACCESSED VIA THE VARIABLE _FILE_

- SEVERAL EXAMPLES DEMONSTRATED HOW USEFUL THE VARIABLE _FILE_ CAN BE (AND THERE ARE MORE USES SHOWN IN THE PAPER)

ACKNOWLEDGMENTS

- Thanks to ***HOWARD SCHREIER*** who first made me aware of possible uses of the output buffer (the variable _FILE_) in a SAS-L posting

ACKNOWLEDGMENTS

CONTACT INFORMATION

Mike Zdeb

msz03@albany.edu