PSTAT 130

SAS BASE PROGRAMMING

- Lecture 12 -

Dr. Julie Swenson

Objectives

- Do Loops
- Write to External Files
  - Create custom reports
- Reading Non-standard Data Files
  - Delimiters other than spaces
  - Mixed format records
  - Multiple records per line
- SAS Variable Lists
- Data Conversion
  - Numeric to character
  - Character to numeric

DO Loop Processing

- There are four kinds of DO loops in SAS:
  - DO-END
    - Executes statements as a unit, usually as a part of IF-THEN/ELSE statements.
  - Iterative DO
    - Executes a group of statements repetitively based on the value of an index variable.
  - DO WHILE
    - Executes a group of statements repetitively as long as the specified condition remains true. The condition is checked <u>before</u> each iteration of the loop.
  - DO UNTIL
    - Executes a group of statements repetitively until the specified condition is true. The condition is checked <u>after</u> each iteration of the loop.

DO-END

- For one executable statement, the IF-THEN statement does NOT require a DO-END statement

```
IF sex = 'Male' THEN Abbreviation = 'Mr. ';
```

- For more than one executable statement, the IF-THEN statement requires a DO-END statement

```
IF sex = 'Male' THEN
DO;
   Abbreviation = 'Mr. ';
   Salutation = 'Sir ';
END;
```

Iterative DO

- Use the iterative DO to repeat a group of statements a fixed number of times, based on the value of an index variable

- It helps us to perform repetitive calculations

Perform Repetitive Calculations

- Example
  - On January 1 of each year (2021-2023), $5,000 is invested in an account. Determine the value of the account after three years based on a constant annual interest rate of 7.5 percent.

```
data invest;
   do Year=2021 to 2023;
      Capital+5000;
      Capital+(Capital*.075);
   end;
run;
```

Year	Capital
2024	17364.61

Perform Repetitive Calculations

- Example
  - Now display the value of the account at the **end** of each year

```
data invest;
   do Year=2021 to 2023;
      Capital+5000;
      Capital+(Capital*.075);
      output;
   end;
run;
```

Year	Capital
2021	5375.00
2022	11153.13
2023	17364.61

Perform Repetitive Calculations

- Example
  - Create a program that shows, step by step
    - The summation of 1 to 10
    - The calculation of 10! (10 factorial)

```
data iterate;
   Summation = 0;
   Factorial = 1;
   n = 10;
   do i = 1 to n;
      Summation = Summation + i;
      Factorial = Factorial * i;
      output;
   end;
run;
```

Example Output

Obs	i	Summation	Factorial
1	1	1	1
2	2	3	2
3	3	6	6
4	4	10	24
5	5	15	120
6	6	21	720
7	7	28	5040
8	8	36	40320
9	9	45	362880
10	10	55	3628800

Forecasting Example

- In our last lecture, we created a data set containing weight projections for individuals.

- Let's look at a similar problem.
  - Use the `growth` data set in the data2 folder
  - Forecast the number of employees forward for the next three years, assuming an annual increase of 10%
  - Compare the old method vs. the new method

Forecasting Example

```
data forecast;
   set data2.growth(rename=(NumEmps=NewTotal));
   Increase = .1;
   Year=1;
   NewTotal=NewTotal*(1+Increase);
   output;
   Year=2;
   NewTotal=NewTotal*(1+Increase);
   output;
   Year=3;
   NewTotal=NewTotal*(1+Increase);
   output;
run;
```

```
data forecast;
   set data2.growth(rename=(NumEmps=NewTotal));
   Increase = .1;
   do Year=1 to 3;
      NewTotal=NewTotal*(1+Increase);
      output;
   end;
run;
```

More on the Iterative DO Statement

- The index variable can start or stop on any number

```
do i = 5 to 20;
   <SAS Statements>
end;
```

- The index variable can iterate in multiples using the BY option

```
DATA odd;
   do i = 1 to 100 by 2;
      output;
   end;
run;
```

- Index variables can take on *user-defined lists*

```
do Day = 'Mon','Tue','Wed','Thu','Fri','Sat','Sun';
   output;
end;
```

The Iterative DO Statement

- How many times will each DO loop execute?

```
do Month='JAN','FEB','MAR';
      3 times.
do Fib=1,2,3,5,8,13,21;
      7 times.
do i=Var1,Var2,Var3;
      3 times.
do j=BeginDate to Today() by 7;
      Unknown. The number of iterations depends
      on the values of BeginDate and Today().
do k=Test1-Test50;
      1 time. A single value of k is determined
      by subtracting Test50 from Test1.
```

DO-WHILE and DO-UNTIL Loops

- A DO-WHILE Loop executes a group of statements repetitively as long as the specified condition remains true. The condition is checked before each iteration of the loop.

- A DO-UNTIL Loop executes a group of statements repetitively until the specified condition is true. The condition is checked after each iteration of the loop.

Conditional Iterative Processing

- Example
  - Determine the number of years it would take for an account to exceed $1,000,000 if $5,000 is invested annually at 7.5 percent interest.

```
DATA invest;
   do while(Capital<1000000);
      Year+1;
      Capital+5000;
      Capital+(Capital*.075);
   end;
run;
```

Capital	Year
1047355.91	38

Iterative DO with Conditional Clause

- Example
  - Determine the return of the account again. Stop the loop if 25 years is reached or more than $250,000 is accumulated

```
data invest;
   do Year=1 to 25 until(Capital>250000);
      Capital+5000;
      Capital+(Capital*.075);
   end;
run;

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

Year	Capital
21	255594.86

More About the Data Step

READING FROM AN EXTERNAL FILE	WRITING TO AN EXTERNAL FILE
The DATA statement begins the DATA step.	The DATA statement begins the DATA step.
The INFILE statement identifies an external file to read with an INPUT statement.	The FILE statement identifies an external file to write with a PUT statement.
The INPUT statement describes the arrangement of values in the input data record.	The PUT statement describes the arrangement of values in the output data record.

- Note: INFILE vs. FILE, INFORMAT vs. FORMAT, INPUT vs. PUT
  - One controls data coming into the DATA step, the other controls data leaving the DATA step

Create Custom Reports

- You can use the `PUT` statement with the `DATA _NULL_` statement to write output in any specified format.
  - `DATA _NULL_` begins a DATA step but does NOT create a data set.
  - It writes output to a <u>specified file</u> using `PUT` statements
  - The `PUT` statements control the exact <u>location</u> and <u>format</u> of information in the output file

DATA _NULL_ Program and Output

```
data _null_;
   set data1.admit;
   if Sex = 'M' then Salutation = 'Mr.'; Else Salutation = 'Ms.';
   file '/home/user/admitreport.doc' PRINT;
   title;
   put @5 'Dear ' Salutation ' ' Name ':' //
       @5 'Your weight at admission was ' Weight 3. ' pounds.' /
       @5 'Your height at admission was ' Height 2. ' inches.' //
       @10 'Thank you for your ' fee dollar8.2 ' payment!';
   put _page_;
run;
```

```
Dear Mr.  Murray, W :

Your weight at admission was 168 pounds.
Your height at admission was 72 inches.

     Thank you for your   $85.20 payment!
```

Colon Modifier

- The Colon Modifier is used to read values only as far as the next delimiter
  - Allows for the use of Informats with List input to handle non-standard data values (i.e., values that do not match the format)

Colon Modifier: Character Example

- In List input, use the Colon Modifier to read character values longer than 8 characters (with no blank spaces). The ':' tells SAS to read in a value until it reaches a space:

```
data students;
input Name : $20. Gender : $6. Age 2.;
datalines;
Elizabeth female 23
Jonathan male 22
Zack male 19
;
run;
```

INFILE Statement Options

Problem	Option
Non-blank delimiters	DLM='*delimiter(s)*'
Missing data at end of row	MISSOVER
Missing data represented by consecutive delimiters **and/or** Embedded delimiters where values are surrounded by double quotes	DSD

Using Other Delimiters

- A delimiter is used to separate values of adjacent variables

- In SAS, the default delimiter is a “space,” but many other programs use commas or tabs

Space delimited vs. Comma delimited

```
John 22 M
Betty 19 F
Dave 18 M
```

```
John,22,M
Betty,19,F
Dave,18,M
```

The DLM= Option

- The DLM= option is used to specify a delimiter
  - The delimiter can be any character

- Example:

```
data new;
   infile 'students.txt' dlm=',';
   input Name $ Age Gender $;
run;
```

```
Mary,21,Female
John,22,Male
David,18,Male
```

Missing Data at the End of a Row

- By default, when there is missing data at the end of a row
  1. SAS loads the next record to finish the observation
  2. A note is written to the log
  3. SAS loads a new record at the top of the DATA step and continues processing

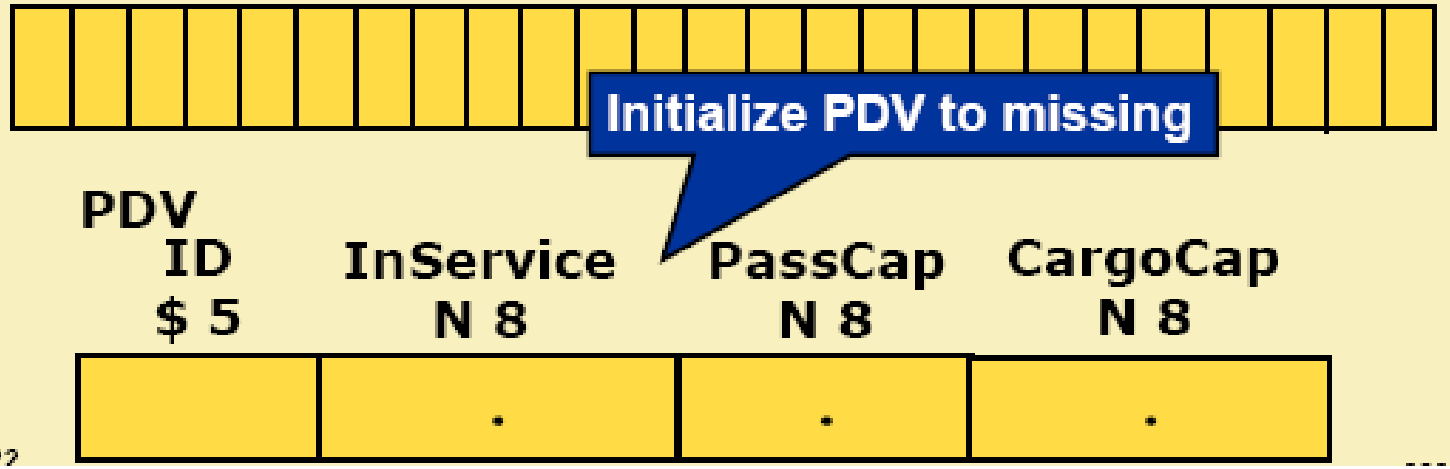
Missing Data at the End of a Row

Raw Data File

```
50001 , 4feb1989,132
50002, 11nov1989,152, 540
50003, 22oct1991,90, 530
50004, 4feb1993,172
50005, 24jun1993, 170, 510
50006, 20dec1994, 180, 520
```

```
data airplanes3;
   length ID $ 5;
   infile 'raw-data-file'
          dlm=',';
   input ID $
         InService : date9.
         PassCap CargoCap;
run;
```

Input Buffer



Missing Data at the End of a Row

```
proc print data=airplanes3 noobs;
run;
```

PROC PRINT Output

ID	In Service	Pass Cap	Cargo Cap
50001	10627	132	50002
50003	11617	90	530
50004	12088	172	50005
50006	12772	180	520

Solution: MISSOVER Option

- The MISSOVER option tells SAS it should not read from the next line if there are any missing values

```
data airplanes3;
   infile 'airdata2.txt' dlm=',' missover;
   input ID $
   InService : date9.
   PassCap CargoCap;
run;
```

ID	In Service	Pass Cap	Carg Cap
50001	10627	132	.
50002	10907	152	540
50003	11617	90	530
50004	12088	172	.
50005	12228	170	510
50006	12772	180	520

The DSD Option

- The DSD option:
  - Sets the default delimiter to a comma
  - Treats consecutive delimiters as missing values

```
5 0 0 0 1 , 4feb1989 ,,  5 3 0
```

  - Enables SAS to read values with embedded delimiters if the value is surrounded by double quotes

Mixed Record Types

- What if not all records have the same format?

Data file

```
101 USA 1-20-1999 3295.50
3034 EUR 30JAN1999 1876,30
101 USA 1-30-1999 2938.00
128 USA 2-5-1999 2908.74
1345 EUR 6FEB1999 3145,60
109 USA 3-17-1999 2789.10
```

Desired Output

Sales ID	Location	Sale Date	Amount
101	USA	14264	3295.50
3034	EUR	14274	1876.30
101	USA	14274	2938.00
128	USA	14280	2908.74
1345	EUR	14281	3145.60
109	USA	14320	2789.10

The Trailing @ Modifier

- The trailing @ modifier tells SAS to hold the current input buffer line for further processing:

```
data sales;
   infile '/home/user/sales.dat';
   input SalesID $ Location $ @;
      if location='USA' then
         input SaleDate : mmddyy10.
               Amount;
      else if Location='EUR' then
         input SaleDate : date9.
               Amount : commax8.;
Run;
```

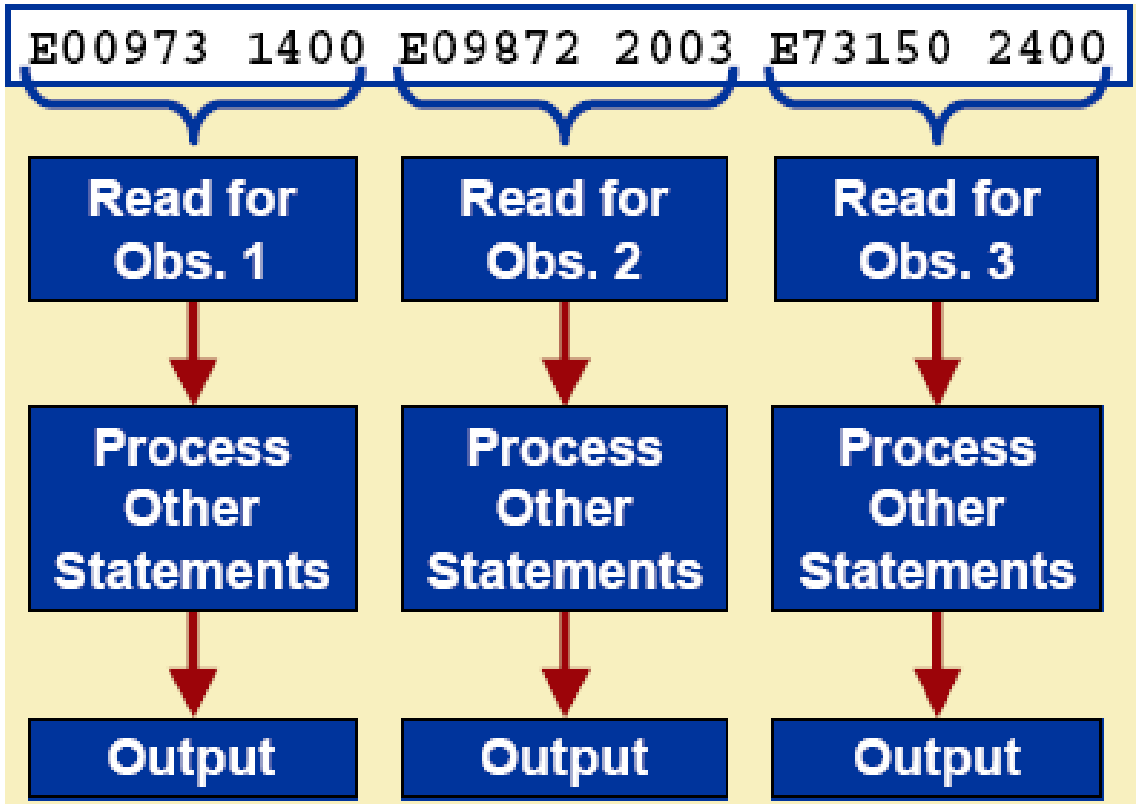
Mixed Record Types

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

PROC PRINT Output

Sales ID	Location	Sale Date	Amount
101	USA	14264	3295.50
3034	EUR	14274	1876.30
101	USA	14274	2938.00
128	USA	14280	2908.74
1345	EUR	14281	3145.60
109	USA	14320	2789.10

Multiple Observations Per Line



The Double Trailing @ Modifier

```
data work.retire;
    length EmpID $ 6;
    infile 'raw-data-file';
    input EmpID $ Contrib @@;
run;
```

Hold until end of record.

Trailing @ vs. Double Trailing @

Option	Effect
Trailing @ **INPUT** *var-1*... @;	Holds raw data record until 1) an INPUT statement with no trailing @ 2) the bottom of the DATA step.
Double trailing @ **INPUT** *var-1* ... @@;	Holds raw data records in the input buffer until SAS reads past the end of the line.

SAS Variable Lists

- Numbered range lists
- Name range lists
- Name prefix lists
- Special SAS name lists

Numbered Range List

- Allows you to refer to a set variables that <u>all start with the same variable name, and all end with a number</u>

- Example:

```
input Salesman $ Week1 Week2 Week3 Week4...Week52;
```

- A numbered range list refers to all the weeks as follows:

```
PROC print;
   var Week1-Week52;
run;
```

Name Range List

- Allows you to refer to a series of variables that <u>appear in consecutive order in the data set</u>
- Example:

```
input Salesman $ Mon Tues Wed Thurs Fri Sat Sun Region;
```

- A name range list refers to all the days as follows:

```
PROC print;
   var Mon--Sun;
run;
```

Note: `Salesman-Numeric-Region` includes only Numeric variables in the range, and `Salesman-Character-Region` includes only character variables in the range

Name Prefix List

- Some SAS functions and statements allow you to refer to all variables that begin with a specified character string.

- Example:

```
sum(of SALES:)
```

  - This calculates the sum of all the variables that begin with 'SALES', such as SALES_JAN, SALES_FEB, and SALES_MAR.

Special SAS Names List

- Special SAS name lists include
  - `_NUMERIC_`
    - Specifies all numeric variables that are already defined in the current DATA step.
  - `_CHARACTER_`
    - Specifies all character variables that are already defined in the current DATA step.
  - `_ALL_`
    - Specifies all variables that are already defined in the current DATA step.

Data Conversion

- In some instances, you may need to convert one data type into another.

- You can convert data types
  - Implicitly by allowing the SAS System to do it for you
  - Explicitly with these functions:
    - INPUT character-to-numeric conversion
    - PUT numeric-to-character conversion.

Automatic Character-to-Numeric Conversion

- The **`data2.salary1`** data set contains a character variable `GrossPay`. Compute a 10 percent bonus for each employee.
- What will happen when the character values of `GrossPay` are used in an arithmetic expression?

ID	GrossPay
$11	$5
201-92-2498	52000
482-87-7945	32000
330-40-7172	49000

```
data bonuses;
   set data2.salary1;
   Bonus=.10*GrossPay;
run;
```

Automatic Character-to-Numeric Conversion

- SAS automatically converts a character value to a numeric value when the character value is used in a numeric context, such as
  - Assignment to a numeric variable
  - An arithmetic operation
  - Logical comparison with a numeric value
  - A function that takes numeric arguments

The INPUT Function

- The INPUT function uses an informat to convert a character string into a number

```
NumVar = INPUT(source,informat);
```

```
data conversion;
   CVar1='32000';
   CVar2='32,000';
   CVar3='03may2008';
   CVar4='050308';
   NVar1=input(CVar1,5.);
   NVar2=input(CVar2,comma6.);
   NVar3=input(CVar3,date9.);
   NVar4=input(CVar4,mmddyy6.);
run;
```

CVar1	CVar2	CVar3	CVar4	NVar1
32000	32,000	03may2008	050308	32000
NVar2	NVar3	NVar4		
32000	17655	17655		

Character-to-Numeric Conversion

- The **`data2.salary2`** data set also contains a character variable `GrossPay`. Now use this data set to compute a 10 percent bonus for each employee.

- What will happen when the character values of `GrossPay` are used in an arithmetic expression?

Failed Character-to-Numeric Conversion

- This approach <u>will not</u> work because `GrossPay` has non-numeric characters (commas).

```
data bonuses;
   set data2.salary2;
   Bonus=.10*GrossPay;
run;

proc print data=bonuses;
run;
```

PROC PRINT Output

ID	GrossPay	Bonus
201-92-2498	52,000	.
482-87-7945	32,000	.
330-40-7172	49,000	.

Explicit Character-to-Numeric Conversion

```
data bonuses;
   set data2.salary2;
   Bonus=.10*input(GrossPay,comma6.);
run;

proc print data=bonuses;
run;
```

PROC PRINT Output

ID	GrossPay	Bonus
201-92-2498	52,000	5200
482-87-7945	32,000	3200
330-40-7172	49,000	4900

Automatic Numeric-to-Character Conversion

- SAS automatically converts a numeric value to a character value when the numeric value is used in a character context, such as
  - assignment to a character variable
  - a concatenation operation
  - a function that accepts character arguments

The PUT Function

- The PUT function uses a SAS format to convert a number into a character string:

```
CharVar = PUT(source,format);
```

- Example
  - `Birth = 12862`
  - `BirthDate = PUT(Birth,mmddyy8.)` = "03/20/95"

INPUT vs. PUT Functions

- These are NOT the same as the INPUT and PUT statements
- Functions
  - The INPUT function converts character data into numeric data
  - The PUT function converts numeric data into character data
- Statements
  - The INPUT statement controls how data is read into a data step
  - The PUT statement controls how data is written out of a data step