Series # 1: **date: 10Feb,2006.**

CONTENTS : 1. Input, put, inputc, inputn, putc & putn functions.

2. Call execute routine.

INPUT FUNCTION :

Syntax : destination_data = input(source_data, informat);

LOGICAL ANALYSIS of how the input function can work:

The input function always takes the source_data as character data. The destination_data can either be numeric or character. So how are we going to decide whether the destination data going to be character or numeric. So the informat is what will decide. The data_type in which the informat absorbs the data decides the data_type of the destination_data. Therefore if the informat is numeric then the destination_data is going to be numeric and if the informat is character then the destination_data is going to be character.

MATHEMATICALLY :

The input function is a one to many correspondence. (character to character/numeric). So we can say to differentiate the correspondence we use the informat as a differentiator.

PROGRAM

```
/*****************************************************************/
/*                  input function
/*****************************************************************/
data one;/* in input the format decides whether the result is char or
num */
      salary = '10000';
      salc = input(salary,$5.); * character --> character. informat-->
character;
      saln = input(salary,5.);  * character --> numeric. informat →
numeric;
      put salary= salc= saln=;
run;
/*****************************************************************/
```

OUTPUT

```
salary=10000 salc=10000 saln=10000
```

Dataset

salary	salary2	salc1	salc2
10000	10000	$10,000	10000

PUT FUNCTION

Syntax : destination data = put (source_data, format);

LOGICAL ANALYSIS of how the put function works :

The put function takes the source_data as character or numeric but the destination_data is always the character data. Here the format helps in getting a numeric format to a character destination_data. For e.g. salary in numeric type can be applied a format dollar8. and stored in character form in the salaryc variable.

The format is character if the source_data is character and numeric if the source data is numeric.

MATHEMATICALLY:

The Put function is many to one correspondence.

PROGRAM:

```
/*****************************************************************/
 *                          put function                         *
/*****************************************************************/
data output.one;
      salaryc = 10000;
      salaryn = '10000';
      salc1 = put(salaryc,DOLLAR10.2);* numeric --> character. format
has to be numeric;
      salc2 = put(salaryn,$10.);* character --> character. format has
to be character;
run;
/*****************************************************************/
```

OUTPUT:

```
salc1=$10,000.00 salc2=10000
```

DATASET:

salary	salary2	salc1	salc2
10000	10000	$10,000.00	10000

SCL FUNCTION WHICH ARE DERIVED FROM INPUT & PUT FUNCTION.

Functions inputc, inputn, putc and putn are derived SCL function from the input and the put functions. They are meant to convert the data from one form(numeric/char) to other form(char/numeric) and we can also have the corresponding format. The main advantage of these SCL functions is that they can be used in the macro open code using the %sysfunc. Note the informat given as the second argument to the SCL function is given in quotes.

INPUTC FUNCTION & INPUTN FUNCTION.

Syntax : destination_data = inputc(source_data, informat)
Syntax : destination_data = inputn (source_data , informat);

PUTC FUNCTION & PUTN FUNCTION.

Syntax : destination_data = putc(source_data, format)
Syntax : destination_data = putn (source_data , format);

Note : If the INPUTC/INPUTN function returns a value to a variable that has not yet been assigned a length, by default the variable length is determined by the length of the first argument.

PROGRAM:

```
data output.inputcnf;
      salaryc = '10000';
      salaryd = '$10,000';
      * inputc : character --> character : informat --> character;
      sal_cha1 = inputc(salaryc, '$6.');
      * inputn : character --> numeric : informat --> numeric;
      sal_num1 = inputn (salaryd, 'dollar7.');
      * putc : character --> character : format --> character ;
      sal_cha2 = putc(sal_cha1,'$6.');
      * putn : numeric --> character : format --> numeric;
      sal_cha3 = putn(sal_num1,'dollar7.');
run;
```

The inputc function is that half part of the input function where the source_data is character, the destination data is also character and the informat that we use is the character informat. This is quite evident from the character ‘c’ in the function name.

ABSTRACTION:

The main utility of inputc/inputn is to convert a character(with format ‘A’) to a (plain) character or numeric data.(here plain means without any format). And the utility of the putc/putn function is to convert a (plain) character/numeric data into a character data with a particular format.

Formatted char data → using inputc/inputn → plain character/numeric data resp.
Plain char/numeric data → using putc/putn → formatted char data.

NOTE: The SCL functions enables you to specify a character informat at run time

CALL EXECUTE:

The main use of the call execute routine is to execute a macro in the data step it is one of the interfaces to the macro from the data step.

To Understand call execute routine we need remember some facts. The data step is first compiled and during the compilation stage the PDV is being constructed.

The call execute is useful when you want to execute a macro conditionally.

1. If call execute produces macro language elements those elements execute immediately.
2. If call execute produces SAS language statements or if the macro language elements generate SAS language elements then they execute at the end of the data step boundary.

Because macro references execute immediately and SAS statements do not execute until after a step boundary, you cannot use CALL EXECUTE to invoke a macro that contains references for macro variables that are created by CALL SYMPUT in that macro.

PROGRAM :

```
%macro test;
      %put in the macro;
%mend;
data one;
      input name $;
      put 'in the data step';
      call execute('%test');
datalines;
sachin
gadkar
;
run;

%macro testing;
      data _null;
            put 'in the macro';
      run;
%mend;
data two;
      input name $;
      put 'in the data step';
      call execute('%testing');
datalines;
sachin
gadkar
;
run;
```

LOG :

```
440  %macro test;
441      %put in the macro;
442  %mend;
443  data one;
444      input name $;
445      put 'in the data step';
446      call execute('%test');
447  datalines;

in the data step
in the macro
in the data step
in the macro
NOTE: The data set WORK.ONE has 2 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


NOTE: CALL EXECUTE routine executed successfully, but no SAS statements were generated.
450  ;
451  run;
452
453  %macro testing;
454      data _null;
455          put 'in the macro';
456      run;
457  %mend;
458  data two;
459      input name $;
460      put 'in the data step';
461      call execute('%testing');
462  datalines;

in the data step
in the data step
NOTE: The data set WORK.TWO has 2 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds


NOTE: CALL EXECUTE generated line.
465  ;
1   + data _null;         put 'in the macro';     run;

in the macro
NOTE: The data set WORK._NULL has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


2   + data _null;         put 'in the macro';     run;
```

```
in the macro
NOTE: The data set WORK._NULL has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


    466  run;
```

Sachin Gadkar.
Cognizant Tech.Soln.