Generating Customized Reports Using the INPUT Statements

Jiyuan Wu, Ph.D., The Gallup Organization, Rockville, MD 20850

ABSTRACT

There are two types of situations where the INPUT statements from the DATA step, designed for reading raw or external data, is one of the most efficient ways for outputting customized reports. You can read data from several lines and put them on a single line, or read data from a single line and put it on several lines. The real life data examples presented in the paper illustrate how to use a few lines of INPUT statement code to generate outputs of desire.

READING SEVERAL LINE TO OUTPUT A SINGLE LINE

A SAS data set or a text file may happen to be arranged such that related information is spread over several lines. On the other hand, sometimes the desired reports need to put them together into a single line.

For example, the following lines of data are produced by a statistical program called SUDAAN, which calculates appropriate standard errors for data from a complex sampling design. A portion of the original output from SUDAAN is shown below in Table One:

Table 1: Original SUDAAN Output

```
 EST.  S.E.

 6.5   4.2
 8.9   6.6
 2.8   2.8

 2.6   1.8
 2.3   2.3
 3.1   3.1
```

Given these data, the following report format is desired:

Table 2: Customized Layout of SUDAAN Output

OBS	TOT EST.	TOT S.E.	REG1 EST.	REG1 S.E.	REG2 EST.	REG2 S.E.
1	6.5	4.2	8.9	6.6	2.8	2.8
2	2.6	1.8	2.3	2.3	3.1	3.1

In other words, result for each survey item will be put on a single line, in the order of Grand Total, Region1, and then Region 2, with the percentages and standard errors placed side by side.

We introduce the following sample code that produces Table Two:

Program 1:

```
        1.  /*GENERATE A TEXT FILE FOR EST & SE*/
2.  DATA _NULL_;
        3.  SET SUDAAN;
        4.  FILE CUT;
        5.  PUT EST 4.1 ' ' SE 4.1;
6.  RUN;
        7.  /*ANOTHER SAS DATA, USING INPUT
            STATEMENT TO RE-ARRANGE OUTPUT*/
8.  DATA READ;
        9.  INFILE CUT;
        10. INPUT EST1 SE1 #2 EST2 SE2 #3 EST3
            SE3;
        11. LABEL EST1='TOT/EST.' EST2='REG1/EST.'
            EST3='REG2/EST.'
                12. SE1='TOT/S.E.'
                    SE2='REG1/S.E.'
                    SE3='REG2/S.E.';
        13. FORMAT EST1-EST3 SE1-SE3 4.1;
14. RUN;
        15. /*PRINT OUT THE SAS DATA SET*/
16. PROC PRINT DATA=READ LABEL SPLIT='/';
        17. FORMAT EST1-EST3 SE1-SE3 4.1;
18. RUN;
```

The key statement is the INPUT statement, using List style in this case. It tells SAS to read EST1 and SE1 from the first line, then move to Line 2 to

read EST2 and SE2, followed by Line 3 for EST3 and SE3. At each iteration, this INPUT statement will be repeated reading the next three lines of data and put them in the next line in the SAS data set. The process will continue until it reaches the end of the text file CUT.

Technically, there are three steps involved in the above code:

1. if you have SAS data, then based on which you produce a text file, as shown above by Line 2 to Line 6. Because the INPUT statement only reads raw (ASCII, text, flat) files, you need this step to produce the file to be used by INPUT. If your already have raw data, this step (Line 1-6) can be omitted.
2. you read the text file to produce the customized reports. The direct outcome of this step is a SAS data set, with the data arrangement of your desire.
3. once you have the SAS data, use PROC PRINT, PROC FSVIEW or PROC GCHARTS will give you the printout of the customized reports.

READING A SINGLE LINE TO OUTPUT SEVERAL LINES

The other major usage of INPUT statement is to read a single data line and then spread the data over several lines. Typically this need arises when the original data have long data lines, each holding several related variables, as shown in Table 3:

Table 3: The Original Layout – All Variables For an Area on a Single Line

AREA	SMALL	LARGE	PREMIUM	AVERAGE	SMALL	LARGE	PREMIUM	AVERAGE
4A	0.81	0.65	0.55	0.67	0.96	0.95	0.84	0.92
4B	0.88	0.72	0.58	0.73	0.98	0.97	0.79	0.91
4C	0.87	0.73	0.59	0.73	0.97	0.97	0.87	0.94

Each of the 10 data lines contains 8 variables that all belong to a same Area. The objective is to produce a report that breaks down each line into 2 for each area, making a total of 20 lines for the 10 areas, as shown below:

Table 4: New Layout – 2 Lines for Each Area

AREA	SMALL	LARGE	PREMIUM	AVERAGE
4A	0.81	0.65	0.55	0.67
4A	0.96	0.95	0.84	0.92
4B	0.88	0.72	0.58	0.73
4B	0.98	0.97	0.79	0.91
4C	0.87	0.73	0.59	0.73
4C	0.97	0.97	0.87	0.94

Such a report has been produced by the following code:

Program 2:

```
        1./*CREATE A TEXT FILE*/
2.DATA _NULL_;
         3.SET WOO.AREA;
         4.FILE WOOS;
         5.PUT AREA $2. ' ' SDIAG1
 LDIAG1 PDIAG1 ADIAG1 4.2 ' '
                  6.AREA $2. ' ' SDIAG2
          LDIAG2 PDIAG2 ADIAG2 4.2;
7.RUN;
         8./*READ BACK THE ASCII AND RE-
 ARRANGE THE LAYOUT*/
9.DATA READ;
         10.INFILE WOOS;
         11.INPUT AREA $2. SMALL LARGE
 PREMIER AVERAGE 4.2 +1 @@;
12.RUN;
        13./*PRINTING DATA WITH NEW
LAYOUT*/
14.PROC PRINT;
```

```
        15.ID AREA;
        16.VAR SMALL LARGE PREMIER AVERAGE;
        17.TITLE NEW LAYOUT: 2 LINES OF DATA FOR
 EACH AREA;
18.RUN;
```

The INPUT and PUT statements have been highlighted on Line 13 and 5, respectively. This program shows the power of INPUT statement: Through a single line of code, not only did we break down each line into 2, but we did so for each and every Area.

The above code contains two major steps. Step One is to produce a text file using the PUT statement on Line 5. There are 2 reasons for this step. First, the INPUT statement only works with raw data, thus a text file is required. Second, through this step, we can re-arrange data so that logically related variables stay together, which facilitates later breakdown.

The other point worth noting is that we have listed the Area variable twice on the PUT statement, one as the first variable and the other as the sixth variable. This is to facilitate the INPUT statement to produce identical values for Area after breaking each into two lines, as we will see more clearly later.

After producing the raw data file, we move to Step 2 to read the data in order to produce a SAS data set with the layout we desire. By using the double trailing @ we tell SAS to hold the record to allow further processing across iterations. Since there are 5 variables named in the INPUT statement before @@, after reading the first 5 variables in the line for the first area (='4A'), SAS will hold the line and move the pointer one column to the right (specified by the code of '+1' on Line 11) to correctly read the remaining 5 variables to the second line. Now we see why we have used the Area variable twice on the PUT statement. To ensure that the two lines will have identical values of the Area variable.

Author's Address

Jiyuan Wu
The Gallup Organization
One Church Street, Suite 204
Rockville, MD 20850
Email: jiyuan_wu@gallup.com