Dealing with Blanks: Leading, Trailing, In-Between

Virginia Chen, ERS Group, Washington DC

ABSTRACT

Dealing with blanks sometimes can be a headache. Very often, the data you receive might contain unwanted spaces, and you want to remove them to make your data clean. Fortunately SAS provides a variety of useful character functions to handle blanks in a character string. This paper compares some frequently used functions that remove leading blanks, trailing blanks or multiple blanks in the middle of a string. The paper also discusses an efficient way to concatenate character strings after removing blanks. The purpose is to make data manipulation more accurate, efficient and ultimately a breeze for your daily data work. The functions we will discuss include TRIM, TRIMN, STRIP, LEFT, COMPRESS, COMPBL, and a few concatenation functions including CAT, CATT, CATS, and CATX. The intended audience is beginning to intermediate SAS users with good knowledge of Base SAS.

COMPARISON 1: TRIM vs. TRIMN

The first comparison is between the TRIM and TRIMN functions. Both TRIM and TRIMN remove trailing blanks from a character string. The only difference is how they deal with blank strings. If there is a blank string variable, the TRIM function returns one blank whereas the TRIMN function returns no blank characters. The following sample data and the program will illustrate how to use these two functions to remove trailing blanks.

Example 1: Create Sample Data and Compare TRIM and TRIMN

```
data sample;
input string $char14.;

datalines;
Mary Smith          /* contains trailing blanks */
    John Brown      /* contains leading blanks  */
  Alice Park        /* contains leading and trailing blanks */
 Tom    Wang        /* contains leading, trailing and multiple blanks in between */
                    /* contains a blank string */
;

data sample;
set sample;
   original   = '*' || string        || '*';
   trim       = '*' || trim(string)  || '*';
   trimn      = '*' || trimn(string) || '*';
run;

proc print noobs data=sample (drop=string);
       title2 'Output of TRIM and TRIMN';
run;
```

```
                      Table 1
                Output of TRIM and TRIMN

     original         trim                trimn

 *Mary Smith    *     *Mary Smith*        *Mary Smith*
 *     John Brown*    *     John Brown*   *     John Brown*
 *   Alice Park   *   *   Alice Park*     *   Alice Park*
 * Tom    Wang    *   * Tom    Wang*      * Tom    Wang*
 *              *     * *                 **
```

In the output of Table 1, the variable ORIGINAL contains the original values of the string (including blanks). The variables TRIM and TRIMN have the trailing blanks removed for the first four records. For the last record (blank string), the TRIM function returns one blank whereas the TRIMN function returns no blank character.

COMPARISON 2: STRIP vs. TRIM(LEFT) or TRIMN(LEFT)

Similar to the TRIM function, the STRIP function allows you to remove trailing blanks. In addition, STRIP removes leading blanks too. Therefore, for strings that lack leading blanks but have at least one non-blank character, the STRIP and TRIM functions return the same value. For blank strings, both the STRIP and TRIMN functions return the same value (zero blank characters).

Another function called the LEFT function allows you to left align a character and remove leading blanks. If we use the LEFT function and the TRIM function together, we can first remove leading blanks and then remove trailing blanks, which will return the same results as the STRIP function. We'll use the same data in the previous discussion to demonstrate how to use these functions to remove both leading and trailing blanks.

Example 2: Compare STRIP, TRIM(LEFT) and TRIMN(LEFT)

```
data sample;
set sample;
   strip      = '*' || strip(string)       || '*';
   trim_left  = '*' || trim(left(string))  || '*';
   trimn_left= '*' || trimn(left(string)) || '*';
run;

proc print data=sample noobs;
       title2 'Output of STRIP, TRIM(LEFT) and TRIMN(LEFT)';
       var original strip trim_left trimn_left;
run;
```

```
                              Table 2
            Output of STRIP, TRIM(LEFT) and TRIMN(LEFT)

    original        strip           trim_left       trimn_left

*Mary Smith    *    *Mary Smith*    *Mary Smith*    *Mary Smith*
*    John Brown*    *John Brown*    *John Brown*    *John Brown*
*  Alice Park  *    *Alice Park*    *Alice Park*    *Alice Park*
* Tom   Wang   *    *Tom   Wang*    *Tom   Wang*    *Tom   Wang*
*              *    **              * *             **
```

In this output, the variables STRIP (column 2) and TRIMN_LEFT (column 4) contain exactly the same values. The TRIM_LEFT variable (column 3) is different from the previous two variables only in the last record where the former keeps one blank for this blank string and the latter returns no blank characters. The advantage of using the STRIP function is that it runs faster than TRIM(LEFT) or TRIMN(LEFT) (SAS OnlineDoc® 9.1.3). If you have a large data set, it is recommended to use the STRIP function to remove leading and trailing blanks.

COMPARISON 3: COMPRESS vs. COMPBL

Another useful function to deal with blanks is the COMPRESS function. It removes any specified characters (eg. space, dash or parenthesis) from a character string. If you do not specify characters to remove, the COMPRESS function removes only blanks by default. The COMPBL function, similar to the COMPRESS function, compresses blanks yet it does not compress a single blank in a string. In other words, the COMPBL function allows you to compress multiple blanks into a single blank and has no effect on a single blank (Howard, 1999). The following two DATA steps illustrate how to use these two functions to remove blanks and special characters.

Example 3.1: Use COMPRESS to Remove Blanks and Compare with COMPBL

```
data sample;
set sample;
   compress   = '*' || compress(string) || '*';
   compbl     = '*' || compbl(string)   || '*';
run;

proc print data=sample noobs;
       title2 'Output of COMPRESS and COMPBL';
       var original compress compbl;
run;
```

```
                      Table 3.1
            Output of COMPRESS and COMPBL

    original        compress       compbl

*Mary Smith    *    *MarySmith*    *Mary Smith *
*    John Brown*    *JohnBrown*    * John Brown*
*  Alice Park  *    *AlicePark*    * Alice Park *
* Tom   Wang   *    *TomWang*      * Tom Wang *
*              *    **             * *
```

In this example, the variable COMPRESS has all the blanks removed from the string. The variable COMPBL has multiple blanks compressed into a single blank and keeps the original single blank. For example, the third record (Alice Park) has two leading blanks, one blank between “Alice” and “Park”, and two trailing blanks in the ORIGINAL field. Using the COMPBL function will compress the two leading blanks into one, keep the single blank in between, and compress the two trailing blanks into one. If a string is blank as in the last record, the COMPBL function returns a single blank character.

Example 3.2: Use COMPRESS to Remove Specified Characters

```
data zipcode;
input zipcode $14.;

       zipcode1 = compress(zipcode);           /* to remove blanks */
       zipcode2 = compress(zipcode,' ()?');    /* to remove blanks, () and ? */
       zipcode3 = compress(zipcode,'- ()?');   /* to remove dash, blanks, () and ? */

datalines;
22168- 12 34
22168- (1234?)
;

proc print data=zipcode noobs;
       title2 "Listing of Zipcodes";
run;
```

```
                          Table 3.2
                     Listing of Zipcodes

   zipcode          zipcode1        zipcode2      zipcode3

22168- 12 34      22168-1234       22168-1234     221681234
22168- (1234?)    22168-(1234?)    22168-1234     221681234
```

The variable ZIPCODE is the original value. The variable ZIPCODE1 shows the results after we compress the blanks. Notice that the COMPRESS function does not have a second argument for this variable; therefore the default character to remove is blank. What if we want to remove blanks and the special characters (parentheses and question mark) in the second row? To accomplish this task, we can create a new variable ZIPCODE2 and include a list of the characters to remove in the second argument. Since we have specified a few special characters in the second argument, blank is no longer a default character to remove. We have to explicitly include it in the second argument (Cody, 2006). Finally, if you just want to keep numbers in your zip code list, you can remove all the characters by adding them to the second argument. The results are shown in the last column, ZIPCODE3.

COMPARISON 4: CAT, CATT, CATS, CATX

Having introduced the above functions that deal with blanks, we would like to compare them with the new concatenation functions introduced in SAS® 9 that are powerful in terms of joining strings and automatically removing blanks. Traditionally, we use the concatenation operator (||) to join strings, as demonstrated in the Example 1, 2 and 3.1. Although the syntax works fine, it looks a little tedious to code, especially when you have long character strings to join. Using the new concatenation functions can simplify your coding and make your data manipulation more flexible. Below are a list of these functions and examples of how they work.

- CAT function concatenates character strings without removing leading or trailing blanks.
- CATT function concatenates character strings and removes trailing blanks.
- CATS function concatenates character strings and removes leading and trailing blanks.
- CATX function concatenates character strings, removes leading and trailing blanks, and inserts separators between each string.

Example 4.1: Compare CAT, CATT, CATS, CATX

```
data sample;
       set sample;
       length cat catt cats $16 catx $20;
       text='Hello';
       cat =cat ('*',string,'*');       /* (= ||)                 */
       catt=catt('*',string,'*');       /* (= TRIM || or TRIMN ||) */
       cats=cats('*',string,'*');       /* (= STRIP |||)          */
       catx=catx('!',text,string);      /* (= STRIP || separator) */
run;

proc print data=sample noobs;
       var    cat catt cats catx;
       title2 "Output of Concatenation Functions";
run;
```

```
                              Table 4.1
                   Output of Concatenation Functions

      cat           catt                cats            catx

*Mary Smith    *    *Mary Smith*        *Mary Smith*    Hello!Mary Smith
*    John Brown*    *    John Brown*    *John Brown*    Hello!John Brown
*  Alice Park  *    *  Alice Park*      *Alice Park*    Hello!Alice Park
* Tom   Wang   *    * Tom   Wang*       *Tom   Wang*    Hello!Tom   Wang
*              *    **                  **              Hello
```

In this output, the variable CAT has the string joined with the two asterisks. Using the CAT function is equivalent to using the concatenation operator. No blanks are removed in this step. The variable CATT has the trailing blanks removed and the two asterisks joined to the string. Think of the CAT**T** function as **T**RIM(N) and CAT, which returns the same results as TRIM(N) and ||, as shown in Table 1. The variable CATS has both leading and trailing blanks removed, and the two asterisks joined to the string. Think of the CAT**S** function as **S**TRIP and CAT, which returns the same results as STRIP and ||, as shown in Table 2. The CATX function is similar to CATS; however, it allows you to insert a separator between each string. For example, we inserted an exclamation mark between “Hello” and each name, and stripped off any leading and trailing blanks using the CATX function. The results are displayed in the last field. In terms of efficiency, it is faster to use the CAT, CATT, CATS, and CATX functions than the TRIM and LEFT functions (SAS OnlineDoc® 9.1.3). Notice that the default length for the CAT, CATT, CATS, and CATX functions is **200**. If you want to shorten the length, simply use the LENGTH statement to specify a length for your variables.

Example 4.2: Use the "OF" syntax for variable lists

Another advantage of using the concatenation functions is the flexibility of using the “OF” operator to simplify the variable lists if you have a list of variables in numeric order. The following data set contains data of partial addresses, and we want to concatenate them to create a complete mailing list. We will use three types of delimiters to join these strings and review their results in Table 4.2.

```
data mailing;
       length mail_1 - mail_3 $35;
       address1='123 Main St.';
       address2='Eden';
       address3='VT';
       address4='05060';

       mail_1=catx(' ', of address1-address4);  /* Insert space as separator */
       mail_2=catx(',', of address1-address4);  /* Insert comma as separator */
       mail_3=catx(', ', of address1-address4); /* Insert comma and space as separator */
run;
```

```
proc print data=mailing noobs;
      var mail_1 mail_2 mail_3;
      title2 "Listing of Data Set: Mailing";
run;
```

```
                                  Table 4.2
                        Listing of Data Set: Mailing

         mail_1                      mail_2                        mail_3

123 Main St. Eden VT 05060    123 Main St.,Eden,VT,05060    123 Main St., Eden, VT, 05060
```

REFERENCE TABLES

To help you review the functions we discussed above, we have created two tables below for your quick checks. An "x" mark means that a particular task (on the top row) can be performed by a function (on the far left column).

Table 5: Functions that deal with blanks

Function	Remove Leading	Remove Between	Remove Trailing	If a string is blank, return no characters	Compress Multiple Blanks to a Single Blank
TRIM			x		
TRIMN			x	x	
STRIP	x		x	x	
COMPRESS	x	x	x	x	
COMPBL					x

Table 6: Functions that deal with blanks and concatenation

Function	Remove Leading	Remove Between	Remove Trailing	If a string is blank, return no characters	Equivalent Codes
CAT					||
CATT			x	x	TRIM (or TRIMN) ||
CATS	x		x	x	STRIP ||
CATX	x		x		STRIP || and add separator

CONCLUSION

Hope you have enjoyed this journey to the character functions that deal with blanks. Without understanding these functions, you might leave any unwanted blanks in your output and mess up your deliverable reports or a sort or merge. Although data manipulation can be a pain, using these functions appropriately will make your life easier!

REFERENCES

- SAS OnlineDoc® 9.1.3, SAS Institute Inc. Cary, NC. http://support.sas.com/onlinedoc/913/docMainpage.jsp

- Cody, Ronald, "An Introduction to SAS® Character Functions", Proceedings of the SAS Global Forum 2006 http://www2.sas.com/proceedings/sugi31/247-31.pdf

- Howard, Neil, "Introduction to SAS Functions", Proceedings of the SAS Global Forum 1999 http://www2.sas.com/proceedings/sugi24/Begtutor/p57-24.pdf

ACKNOWLEDGMENTS

I would like to thank ERS Director, Paul White, and ERS Principals, Edward Bierhanzl and Rick Holt, for their review and helpful comments on this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Virginia Chen
Enterprise: ERS Group
Address: 2100 M Street, NW
City, State ZIP: Washington, DC 20037
Work Phone: 202-328-1515 ext.506
Fax: 202-462-0594
E-mail: vchen@ersgroup.com
Web: www.ersgroup.com