Having a Ball with Strings

Ronald Cody, Ed.D.
Robert Wood Johnson Medical School

Introduction

SAS® software is especially rich in its assortment of functions that deal with character data. This class of functions is sometimes called STRING functions. In this tutorial, we will demonstrate some of the more useful string functions.

Some of the functions we will discuss are: LENGTH, SUBSTR, COMPBL, COMPRESS, VERIFY, INPUT, PUT, TRANSLATE, TRANWRD, SCAN, TRIM, UPCASE, LOWCASE, REPEAT, || (concatenation), INDEX, INDEXC, AND SOUNDEX. Wow, did you realize there were so many string functions? Let's get started.

How Lengths of Character Variables are Set in a SAS Data Step

Before we actually discuss these functions, we need to understand how SAS software assigns storage lengths to character variables and what the LENGTH function does for us. Look at the following program:

```
DATA EXAMPLE1;
   INPUT GROUP $ @10 STRING $3.;
   LEFT  = 'X    '; *X AND 4 BLANKS;
   RIGHT = '    X'; *4 BLANKS AND X;
   C1 = SUBSTR(GROUP,1,2);
   C2 = REPEAT(GROUP,1);
   LGROUP = LENGTH(GROUP);
   LSTRING = LENGTH(STRING);
   LLEFT = LENGTH(LEFT);
   LRIGHT = LENGTH(RIGHT);
   LC1 = LENGTH(C1);
   LC2 = LENGTH(C2);
DATALINES;
ABCDEFGH 123
XXX       4
Y        5
;
PROC CONTENTS DATA=EXAMPLE1 POSITION;
   TITLE 'OUTPUT FROM PROC CONTENTS';
RUN;

PROC PRINT DATA=EXAMPLE1 NOOBS;
   TITLE 'LISTING OF EXAMPLE 1';
RUN;
```

One purpose of this example is to clarify the term LENGTH. If you look at the output from PROC CONTENTS, each of the variables is listed, along with a TYPE and LENGTH. Take a moment and look at the output from PROC CONTENTS below:

```
CONTENTS PROCEDURE

-----VARIABLES ORDERED BY POSITION-----
```

#	VARIABLE	TYPE	LEN	POS
1	GROUP	CHAR	8	0
2	STRING	CHAR	3	8
3	LEFT	CHAR	5	11
4	RIGHT	CHAR	5	16
5	C1	CHAR	8	21
6	C2	CHAR	200	29
7	LGROUP	NUM	8	229
8	LSTRING	NUM	8	237
9	LLEFT	NUM	8	245
10	LRIGHT	NUM	8	253
11	LC1	NUM	8	261
12	LC2	NUM	8	269

The column labeled LEN (Length) is the number of bytes needed to store the values for each of the variables listed. By default, all the numeric variables are stored in 8 bytes.

But, what about the storage lengths of the character variables? Look first at the two variables listed in the INPUT statement; GROUP and STRING. Since this is the first mention of these variables in this data step, their lengths will be assigned by the rules governing INPUT statements. Since no columns or INFORMATS were associated with the variable GROUP, its length will be set to 8 bytes, the default length for character variables in this situation. The variable STRING uses a $3. INFORMAT so its length will be set to 3 bytes. The length of LEFT and RIGHT are determined by the assignment statement. The storage lengths of C1 and C2 are more difficult to understand.

The variable C1 is defined to be a substring of the variable GROUP. The SUBSTR function takes the form:

```
SUBSTR(char_var,start,length);
```

This says to take a substring from char_var starting at the position indicated by the start argument for a length indicated by the length argument. Why then, is the length of C2 equal to 8 and not 2? The SAS compiler determines lengths at compile time. Since the starting position and length arguments of the SUBSTR function can be variable expressions, the compiler must set the length of C1 equal to the largest possible value it can ever attain, the length of GROUP.

The same type of logic controls the length of C2, defined by the REPEAT function. Since the number of additional replications is defined by the second argument of the REPEAT function, and this argument can be a variable expression, the compiler sets the length of C2 to the largest possible value, 200. Why 200? Because that is the maximum length of a character variable in the SAS system.

There is a lesson here: Always use a LENGTH statement for any character variables that do not derive their length elsewhere. For example, to set the length of C2 to 16, you would write:

```
LENGTH C2 $ 16;
```

The LENGTH function does not, as you might guess, return the storage length of a character variable. Instead, it returns the length of a character string, not including trailing blanks.

The value of LLEFT and LRIGHT are 1 and 5 respectively, for every observation. This demonstrates that the trailing blanks in LEFT are not counted by the LENGTH function while the leading blanks in RIGHT are. The table below summarizes the values returned by the LENGTH function for the remaining variables:

Obs	GROUP	LGROUP	STRING	LSTRING
1	abcdefgh	8	123	3
2	xxx	3	4	1
3	y	1	5	1

Obs	C1	LC1	C2	LC2
1	ab	2	abcdefghabcdefgh	16
2	xx	2	xxx xxx	11
3	y	1	y y	9

The values of LGROUP and LSTRING are easy to understand. The value of LC1 is 1 for the third observation since C1 is only 1 byte in length in the third observation. The values for LC2 are more complicated. The REPEAT function says to take the original value and repeat it n times. So, for the first observation, LC2 is 16 (2 times 8). For observations 2 and 3, the trailing blanks come into play. In observation 2, the value of GROUP is 'XXXbbbbb' (where the b's stand for blanks). When we repeat this string one additional time, we get: 'XXXbbbbbXXXbbbbb'. Not counting the trailing blanks, we have a length of 8 + 3 = 11. Using the same logic for the third observation, we have a 'Y' followed by 7 blanks repeated once. Not counting the last 7 trailing blanks, we have a length of 8 + 1 = 9.

With these preliminaries out of the way, we can now begin our tour of some of the very useful string functions available in SAS software.

Working with Blanks

This example will demonstrate how to convert multiple blanks to a single blank. Suppose you have some names and addresses in a file. Some of the data entry clerks placed extra spaces between the first and last names and in the address fields. You would like to store all names and addresses with single blanks. Here is an example of how this is done:

```
DATA EXAMPLE2;
   INPUT #1 @1  NAME    $20.
         #2 @1  ADDRESS $30.
         #3 @1  CITY    $15.
            @20 STATE    $2.
            @25 ZIP      $5.;
   NAME = COMPBL(NAME);
   ADDRESS = COMPBL(ADDRESS);
   CITY = COMPBL(CITY);
DATALINES;
RON CODY
89 LAZY BROOK ROAD
FLEMINGTON          NJ   08822
BILL     BROWN
28   CATHY   STREET
```

```
NORTH   CITY       NY   11518
;
PROC PRINT DATA=EXAMPLE2;
   TITLE 'EXAMPLE 2';
   ID NAME;
   VAR ADDRESS CITY STATE ZIP;
RUN;
```

This seemingly difficult task is accomplished in a single line using the COMPBL function. It COMPresses BLanks. How useful!

How to Remove Characters from a String

A more general problem is to remove selected characters from a string. For example, suppose you want to remove blanks, parentheses, and dashes from a phone number that has been stored as a character value. Here comes the COMPRESS function to the rescue! The COMPRESS function can remove any number of specified characters from a character variable. The program below uses the COMPRESS function twice. The first time, to remove blanks from the string; the second to remove blanks plus the other above mentioned characters. Here is the code:

```
DATA EXAMPLE3;
   INPUT PHONE $ 1-15;
   PHONE1 = COMPRESS(PHONE);
   PHONE2 = COMPRESS(PHONE,'(-) ');
DATALINES;
(908)235-4490
(201) 555-77 99
;
PROC PRINT DATA=EXAMPLE3;
   TITLE 'LISTING OF EXAMPLE 3';
RUN;
```

The variable PHONE1 has just blanks removed. Notice that the COMPRESS function does not have a second argument here. When it is omitted, the COMPRESS function removes only blanks. For the variable PHONE2, the second argument of the COMPRESS function containes a list of the characters to remove: left parenthesis, blank, right parenthesis, and blank. This string is placed in single or double quotes.

Character Data Verification

A common task in data processing is to validate data. For example, you may want to be sure that only certain values are present in a character variable. In the example below, only the values 'A', 'B', 'C', 'D', and 'E' are valid data values. A very easy way to test if there are any invalid characters present is shown next:

```
DATA EXAMPLE4;
   INPUT ID $ 1-3 ANSWER $ 5-9;
   P = VERIFY(ANSWER,'ABCDE');
   OK = P EQ 0;
DATALINES;
001 ACBED
002 ABXDE
003 12CCE
004 ABC E
;
PROC PRINT DATA=EXAMPLE4 NOOBS;
   TITLE 'LISTING OF EXAMPLE 4';
RUN;
```

The workhorse of this example if the VERIFY function. It is a bit complicated. It inspects every character in the first argument and, if it finds any value not in the verify string (the second argument), it will return the position of the first offending value. If all the values of the string are located in the verify string, a value of 0 is returned. In the first observation, P and OK will be 1; in the second observation, P will be a 3 (the position of the 'X') and OK will be 0; in the third observation, P will be 1 and OK will be 0; finally, in the fourth observation, P will be a 4 and OK will be 0.

Substring Example

We mentioned in the Introduction that a substring is a part a longer string (although it can actually be the same length but this would not be too useful). In this example, you have ID codes which contain in the first two positions, a state abbreviation. Furthermore, positions 7-9 contain a numeric code. You want to create two new variables; one containing the two digit state codes and the other, a numeric variable constructed from the three numerals in positions 7,8, and 9. Here goes:

```
DATA EXAMPLE5;
   INPUT ID $ 1-9;
   LENGTH STATE $ 2;
   STATE = SUBSTR(ID,1,2);
   NUM = INPUT(SUBSTR(ID,7,3),3.);
DATALINES;
NYXXXX123
NJ1234567
;
```

```
PROC PRINT DATA=EXAMPLE5 NOOBS;
   TITLE 'LISTING OF EXAMPLE 5';
RUN;
```

Creating the state code is easy. We use the SUBSTR function. The first argument is the variable from which we want to extract the substring, the second argument is the starting position of the substring, and the last argument is the length of the substring (not the ending position as you might guess). Also note the use of the LENGTH statement to set the length of STATE to 2 bytes.

Extracting the three digit number code is more complicated. First we use the SUBSTR function to pull out the three numerals (numerals are character representations of numbers). However, the result of a SUBSTR function is always a character value. To convert the character value to a number, we use the INPUT function. The INPUT function takes the first argument and "reads" it as if it were coming from a file, according to the informat listed as the second argument. So, for the first observation, the SUBSTR function would return the string '123' and the INPUT function would convert this to the number 123. As a point of interest, you may use a longer informat as the second argument without any problems. For example, the INPUT statement could have been written as:

```
INPUT (SUBSTR(ID,7,3),8.);
```

and everything would have worked out fine. This fact is useful in situations where you do not know the length of the string ahead of time.

Using the SUBSTR Function on the Left-Hand Side of the Equal Sign

There is a particularly useful and somewhat obscure use of the SUBSTR function that we would like to discuss next. You can use this function to place characters in specific locations within a string by placing the SUBSTR function on the left hand side of the equals sign (in the older manuals I think this was called a SUBSTR pesudo function).

Suppose you have some systolic blood pressures (SBP) and diastolic blood pressures (DBP) in a SAS data set. You want to print out these values and star high values with an asterisk. Here is a program that uses the SUBSTR function on the left of the equals sign to do that:

```
DATA EXAMPLE6;
   INPUT SBP DBP @@;
   LENGTH SBP_CHK DBP_CHK $ 4;
   SBP_CHK = PUT(SBP,3.);
   DBP_CHK = PUT(DBP,3.);
   IF SBP GT 160 THEN
      SUBSTR(SBP_CHK,4,1) = '*';
   IF DBP GT 90 THEN
      SUBSTR(DBP_CHK,4,1) = '*';
DATALINES;
120 80 180 92 200 110
;
PROC PRINT DATA=EXAMPLE6 NOOBS;
   TITLE 'LISTING OF EXAMPLE 6';
RUN;
```

We first need to set the lengths of SBP_CHK and DBP_CHK to 4 (three spaces for the value plus one for the possible asterisk). Next, we use a PUT function to perform a numeric to character conversion. The PUT function is, in some ways, similar to the INPUT function. It "writes out" the value of the first argument, according to the FORMAT specified in the second argument. By "write out" we actually mean assign the value to the variable on the left of the equal sign. The SUBSTR function then places an asterisk in the fourth position when a value of SBP is greater than 160 or a value of DBP is greater than 90.

Doing the Previous Example the Hard Way

It is both interesting and instructive to obtain the results above without using the SUBSTR function. We are not doing this just to show you a hard way to accomplish something we already did. Rather, this alternative solution uses a number of character functions that can be demonstrated. Here is the program:

```
DATA EXAMPLE7;
   INPUT SBP DBP @@;
   LENGTH SBP_CHK DBP_CHK $ 4;
   SBP_CHK = PUT(SBP,3.);
   DBP_CHK = PUT(DBP,3.);
   IF SBP GT 160 THEN SBP_CHK =
      SUBSTR(SBP_CHK,1,3) || '*';
   IF DBP GT  90 THEN DBP_CHK =
```

```
      TRIM(DBP_CHK) || '*';
DATALINES;
120 80 180 92 200 110
;
;
PROC PRINT DATA=EXAMPLE7 NOOBS;
   TITLE 'LISTING OF EXAMPLE 7';
RUN;
```

Not really more complicated but maybe just not as elegant as the first program. This program uses the concatenation operator (||) to join the 3 character blood pressure value with an asterisk. Since SBP_CHK and DBP_CHK were both assigned a length of 4, we wanted to be sure to concatenate at most the first 3 bytes with the asterisk. Just for didactic purposes, we did this two ways. For the SBP_CHK variable, we used a SUBSTR function to extract the first 3 bytes. For the DBP_CHK variable, the TRIM function was used. The TRIM function removes trailing blanks from a character string.

Unpacking a String

To save disk storage, you may want to store several single digit numbers in a longer character string. For example, storing five numbers as numeric variables with the default 8 bytes each would take up 40 bytes of disk storage per observation. Even reducing this to 3 bytes each would result in 15 bytes of storage. If, instead, you store the five digits as a single character value, you need only 5 bytes.

This is fine, but at some point, you may need to get the numbers back out for computation purposes. Here is a nice way to do this:

```
DATA EXAMPLE8;
   INPUT STRING $ 1-5;
DATALINES;
12345
8 642
;
DATA UNPACK;
   SET EXAMPLE8;
   ARRAY X[5];
   DO J = 1 TO 5;
      X[J] = INPUT(
      SUBSTR(STRING,J,1),1.);
   END;
   DROP J;
RUN;

PROC PRINT DATA=UNPACK NOOBS;
   TITLE 'LISTING OF UNPACK';
RUN;
```

We first created an array to hold the five numbers, X1 to X5. Don't be alarmed if you don't see any variables listed on the ARRAY statement. ARRAY X[5]; is equivalent to ARRAY X[5] X1-X5; We use a DO loop to cycle through each of the 5 starting positions corresponding to the five numbers we want. As we mentioned before, since the result of the SUBSTR function is a character value, we need to use the INPUT function to perform the character to numeric conversion.

Parsing a String

Parsing a string means to take it apart based on some rules. In the example to follow, five separate character values were place together on a line with either a space, a comma, a semicolon, a period, or an explanation mark between them. You would like to extract the five values and assign them to five character variables. Without the SCAN function this would be hard; with it, it's easy:

```
DATA EXAMPLE9;
   INPUT LONG_STR $ 1-80;
   ARRAY PIECES[5] $ 10
      PIECE1-PIECE5;
   DO I = 1 TO 5;
      PIECES[I] =
         SCAN(LONG_STR,I,',;.! ');
   END;
   DROP LONG_STR I;
DATALINES4;
THIS LINE,CONTAINS!FIVE.WORDS
ABCDEFGHIJKL XXX;YYY
;;;;
PROC PRINT DATA=EXAMPLE9 NOOBS;
   TITLE 'LISTING OF EXAMPLE 9';
RUN;
```

Before we get to a discussion of the SCAN function, we need a brief word about DATALINES4 and the four semicolons ending our data. If you have data values that may include semicolons, you cannot use a simple DATALINES (or CARDS) statement since the semi-colon would signal the end of your data. Instead the statement DATALINES4 (or CARDS4) is used. This causes the program to continue reading data values until four semicolons are read.

The function:

```
SCAN(char_var,n,'list-of-delimiters');
```

returns the nth "word" from the char_var, where a "word" is defined as anything between two delimiters. If there are fewer than n words in the character variable, the SCAN function will return a blank.

By placing the SCAN function in a DO loop, we can pick out the nth word in the string.

Locating the Position of One String Within Another String

Two somewhat similar functions, INDEX and INDEXC can be used to locate a string, or one of several strings within a longer string. For example, if you have a string 'ABCDEFG' and want the location of the letters DEF (starting position 4), the following INDEX function could be used:

```
INDEX('ABCDEFG','DEF');
```

This would return a value of 4. If you want to know the starting position of any one of several strings, the INDEXC function can be used. As an example, if you wanted the starting position of either 'BC', or 'FG' in the string 'ABCDEFG', you would code:

```
INDEXC('ABCDEFG','BC','FG');
```

The function would return a value of 2, the starting position of 'BC'. Here is a short program which demonstrate these two functions:

```
DATA EX_10;
   INPUT STRING $ 1-10;
   FIRST = INDEX(STRING,'XYZ');
   FIRST_C =
      INDEXC(STRING,'X','Y','Z');
DATALINES;
ABCXYZ1234
1234567890
ABCX1Y2Z39
ABCZZZXYZ3
;
PROC PRINT DATA=EX_10 NOOBS;
   TITLE 'LISTING OF EXAMPLE 10';
RUN;
```

FIRST and FIRST_C for each of the the 4 observations are:

```
OBS  FIRST     FIRST_C
 1     4          4
 2     0          0
 3     0          4
 4     7          4
```

When the search fails, both functions return a zero.

Changing Lower Case to Upper Case and Vice Versa

The two companion functions UPCASE and LOWCASE do just what you would expect. These two functions are especially useful when data entry clerks are careless and a mixture of upper and lower cases values are entered for the same variable. You may want to place all of your character variables in an array and UPCASE (or LOWCASE) them all. Here is an example of such a program:

```
DATA EX_11;
   LENGTH A B C D E $ 1;
   INPUT A B C D E X Y;
DATALINES;
M f P p D 1 2
m f m F M 3 4
;
DATA UPPER;
   SET EX_11;
   ARRAY ALL_C[*] _CHARACTER_;
   DO I = 1 TO DIM(ALL_C);
      ALL_C[I] = UPCASE(ALL_C[I]);
   END;
   DROP I;
RUN;

PROC PRINT DATA=UPPER NOOBS;
   TITLE 'LISTING OF UPPER';
RUN;
```

This program uses the _CHARACTER_ keyword to select all the character variables. The result of running this program is to convert all values for the variables A,B,C, D, and E to upper case. The LOWCASE function could be used in place of the UPCASE function if you wanted all your character values in lower case.

Substituting One Character for Another

A very handy character function is TRANSLATE. It can be used to convert one character to another in a string. For example, suppose you recorded multiple choices on a test as

1,2,3,4, or 5 which represented the letters 'A' through 'E' respectively. When you print out the character values, you want to see the letters rather than the numerals. While formats would accomplish this very nicely, it also serves as an example for the TRANSLATE function. Here is the code:

```
DATA EX_12;
   INPUT QUES : $1. @@;
   QUES =
     TRANSLATE(QUES,'ABCDE','12345');
DATALINES;
1 4 3 2 5
5 3 4 2 1
;
PROC PRINT DATA=EX_12 NOOBS;
   TITLE 'LISTING OF EXAMPLE 12';
RUN;
```

The syntax for the TRANSLATE function is:

```
TRANSLATE(char_var,to_string,from_string);
```

Each value in from_string will be translated to the corresponding value in the to_string.

Another interesting application of the TRANSLATE function is create dichotomous numeric variables from character variables. For example, you may want to set values of 'N' to 0 and values of 'Y' to 1. Although this is easily done with IF-THEN/ELSE statements, let's see if we can do it using the TRANSLATE function. Here goes:

```
DATA EX_13;
   LENGTH CHAR $ 1;
   INPUT CHAR @@;
   X = INPUT(
       TRANSLATE(
       UPCASE(CHAR),'01','NY'),1.);
DATALINES;
N Y n y A B 0 1
;
PROC PRINT DATA=EX_13 NOOBS;
   TITLE 'LISTING OF EXAMPLE 13';
RUN;
```

The UPCASE function sets all values to upper case. Next, the TRANSLATE function converts values of 'N' to '0' and 'Y' to '1'. Finally, the INPUT function converts the numerals '0' and '1' to the numbers 0 and 1 respectively.

Substituting One Word for Another in a String

A relatively new function (as of version 6.07 on personal computers), TRANWRD (translate word), can perform a search and replace operation on a string variable. For example, you may want to standardize addresses by converting the words 'Street', 'Avenue', and 'Road' to the abbreviations 'St.', 'Ave.', and 'Rd.' respectively. Look at the following program:

```
DATA CONVERT;
   INPUT @1 ADDRESS $20. ;
   *** Convert Street, Avenue and
   Boulevard to their abbreviations;
   ADDRESS =
   TRANWRD(ADDRESS,'Street','St.');
   ADDRESS =
   TRANWRD (ADDRESS,'Avenue','Ave.');
   ADDRESS =
   TRANWRD (ADDRESS,'Road','Rd.');
DATALINES;
89 Lazy Brook Road
123 River Rd.
12 Main Street
;
PROC PRINT DATA=CONVERT;
   TITLE 'Listing of Data Set
CONVERT';
RUN;
```

The syntax of the TRANWRD function is:

```
TRANWRD (char_var,'find_str','replace_str');
```

That is, the function will replace every occurrence of find_str with replace_str. Notice that the order of the find and replace strings are reversed compared to the TRANSLATE function where the to_string comes before the from_string as arguments to the function. In this example, 'Street' will be converted to 'St.', 'Avenue' to 'Ave.', and 'Road' to 'Rd.'. The listing below confirms this fact:

```
Listing of Data Set CONVERT

OBS    ADDRESS

 1     89 Lazy Brook Rd.
 2     123 River Rd.
 3     12 Main St.
```

Soundex Conversion

SAS software provides a soundex function which returns the soundex equivalent of a name. Soundex equivalents of names allow you to match two names from two different

sources even though they might be spelled differently. Great care is needed when using this function since many very dissimilar looking names may translate to the same soundex equivalent. The soundex equivalent of most names will result in strange looking codes such as C3 or A352. Here is a sample program and the results of the soundex translations:

```
DATA EX_14;
   LENGTH NAME1-NAME3 $ 10;
   INPUT NAME1-NAME3;
   S1 = SOUNDEX(NAME1);
   S2 = SOUNDEX(NAME2);
   S3 = SOUNDEX(NAME3);
DATALINES;
CODY KODY CADI
CLINE KLEIN CLANA
SMITH SMYTHE ADAMS
;
PROC PRINT DATA=EX_14 NOOBS;
   TITLE 'LISTING OF EXAMPLE 14';
RUN;
```

This program with result in the following soundex matches:

Name	Soundex Equivalent
CODY	C3
KODY	K3
CADI	C3
CLINE	C45
KLEIN	K45
CLANA	C45
SMITH	S53
SMYTHE	S53
ADAMS	A352

Conclusions

So ends our tour through some of the more useful character functions. So go out there and have a ball with strings!

Ronald P. Cody, Ed.D.
Robert Wood Johnson Medical School
Department of Environmental and Community Medicine
675 Hoes Lane
Piscataway, NJ 08822
(908)235-4490
cody@umdnj,edu