Paper 059-30

# A Clever Demonstration of the SAS® SUBSTR Function

David J. Austin, Quintiles, Inc., Kansas City, MO


## ABSTRACT

The SAS® SUBSTR function differs from the substring function in other programming languages as it can be used on either side of the assignment operator. This paper demonstrates its practical usage by building programs to separate and manipulate text. Specific topics include: Proper Syntax Use; Comparison with the COLON MODIFIER; Left-hand Side Usage; Use in Conjunction with the INDEX Function; Usage on Both Sides; Usage in an Align MACRO; and finally, Simple Encryption Techniques for Creating Cryptograms. The demonstrations should be useful to both beginner and intermediate SAS users.


## INTRODUCTION: PROPER SYNTAX USE

The syntax for the SUBSTR function on the left-hand side of the equal sign is: **SUBSTR**( *string, starting position* <, *length*> ) = '*characters-to-replace'*, while the right-hand side syntax is: <*variable* = > **SUBSTR**( *string, starting position* <, *length*>). Syntax illustrated using angle brackets ('<' and '>') is optional. Notice, the parameter order is identical on both sides of the assignment operator. *String* refers to a variable containing the string or to the literal string. The *starting position* can be determined by counting the *string* characters from left to right. A *starting position* of one identifies the first character of the *string. Length* is an optional number of characters to extract. The default value of *length* operates on the remainder of the string. For this reason, *length* can also be thought of as the end position of the *string*. Note, SUBSTR is often used in conjunction with the INDEX function to determine *starting positions* and *lengths*.

## CREATE EXAMPLE DATA

```
data citizens;
  length name $20 dob_char $9 prez $4;

  name = 'WASHINGTON,GEORGE';   dob_char = '22FEB1732';  prez = 'YES' ;  output;
  name = 'JEFFERSON, THOMAS' ;  dob_char = '13APR1743';  prez = 'yes' ;  output;
  name = 'FRANKLIN, BENJAMIN';  dob_char = '17JAN1706';  prez = 'Nope';  output;
run;
```

Output from above code

```
EXAMPLE DATA

          name           dob_char     prez

  WASHINGTON,GEORGE      22FEB1732    YES
  JEFFERSON, THOMAS      13APR1743    yes
  FRANKLIN, BENJAMIN     17JAN1706    Nope
```

## LEFT-HAND SIDE USAGE

### EXAMPLE 1.1: COMPARISON WITH THE COLON MODIFIER

The use of SUBSTR on the left-hand side is similar to the use of the SAS COLON MODIFIER ( =: ). Both methods allow comparison of values based on the prefix of a text string. The most frequent use is to distinguish operator input from a variety of forms (for example, 'YES', 'Yes', 'Y', or 'y'). The advantage of using SUBSTR over the COLON MODIFIER is that SUBSTR can also be used in MACRO statements. Notice the variable PREZ in our sample data contains 'Yes' and 'No' values without any consistency. The code below turns the dirty data into a useful text message.

```
if upcase(prez)              =: 'N' then text_msg = 'Was not President of the USA';
if upcase(substr(prez, 1, 1)) =  'Y' then text_msg = 'Was President of the USA';
```

Output

```
EXAMPLE 1.1:  COMPARISON WITH THE COLON MODIFIER

          name           prez              text_msg

   WASHINGTON,GEORGE     YES     Was President of the USA
   JEFFERSON, THOMAS     yes     Was President of the USA
   FRANKLIN, BENJAMIN    Nope    Was not President of the USA
```

## EXAMPLE 1.2: COMPARISON WITH THE COLON MODIFIER IN A MACRO

In SAS version 8.2 the COLON MODIFIER does not work in macro code, and no ERROR or WARNING messages are placed into the log file. Apparently, SAS treats the colon as part of the compare string. The code below demonstrates that no PUT statements appear in the log stating that debug has been turned off:

```
%macro test(debug=N);
  %if %upcase(%substr(&debug, 1, 1)) = Y %then %do;
    %put MACRO:  Debug has been turned on (&debug);
  %end;

  %if &debug =: N %then %do;
    %put MACRO: Debug has been turned off (&debug);
  %end;

%mend;
%test(debug=N)
%test(debug=Y)
%test(debug=Yes)
%test(debug=No)
```

Output

```
EXAMPLE 1.2:  COMPARISON WITH THE COLON MODIFIER IN A MACRO

LOG Output

   36             %test(debug=N);
   37             %test(debug=Y);
   MACRO:  Debug has been turned on (Y)
   38             %test(debug=Yes);
   MACRO:  Debug has been turned on (Yes)
   39             %test(debug=No);
   40
```

## EXAMPLE 2: COMBINING FLAGS

Another use of SUBSTR on the left-hand side is to populate a string at a specific position or column. In this example, six separate flag variable values will be combined into one variable called ALLFLAGS. Here the *starting position* parameter is incremented upon each iteration of the DO LOOP.

```
length allflags $6;

array flags(*) flag1-flag6;
do i = 1 to dim(flags);
  substr(allflags, i, 1) = flags(i);
end;
```

```
EXAMPLE 2:  COMBINING FLAGS

patient    flag1    flag2    flag3    flag4    flag5    flag6    allflags

  101        1        0        1        0        1        0       101010
  102        1        1        0        0        1        1       110011
  103        1        0        0        0        1        1       100011
  104        1        1        1        1        0        1       111101
```

**RIGHT-HAND SIDE USAGE**

**EXAMPLE 3: CHARACTER DATE TO SAS DATE**

Not surprisingly, the predominant use for SUBSTR is extracting part of a string. The sample data contains the character variable DOB_CHAR which contains birthdates. Simply extract the day (DAY), month (MON), and year (YEAR) sections from the character date using SUBSTR. The DAY component has a *starting position* of 1 and a *length* of 2. The MON component has a *starting position* of 3 and a *length* of 3, while the YEAR component has a *starting position* of 6 and a *length* of 4. Since the YEAR section completes the remainder of the string, DOB_CHAR, the SUBSTR function can simply pass in the first two parameters and use the default value of *length*.

Notice DOB_CHAR meets the SAS DATE9 structure, but SAS has no DATE9 INFORMAT. However, a SASDATE variable can be created utilizing the DATE11 INFORMAT by concatenating a dash between each date component. This can be accomplished in one long statement. Simply extract the day, month, and year sections – similar to above - but without creating variables for each section. Concatenate dashes between each component and wrap the whole line inside an INPUT function with the DATE11 INFORMAT as the second parameter. This code replaces the common practice of assigning numbers to the month abbreviations before converting into a SAS date.

```
data example;
  set citizens(drop=prez);
  length day $2 mon $3 year $4;

  day  = substr(dob_char, 1, 2);
  mon  = substr(dob_char, 3, 3);  * Middle 3 characters starting at the third;
  year = substr(dob_char, 6);     * Default length (to the end of the string);

  sasdate = input((trim(left(substr(dob_char, 1, 2)))||'-'||
                  trim(left(substr(dob_char, 3, 3)))||'-'||
                  trim(left(substr(dob_char, 6)))), date11.);
run;
```

Output

```
EXAMPLE 3:  CHARACTER DATE TO SAS DATE

       name           dob_char     day    mon    year    sasdate

WASHINGTON,GEORGE     22FEB1732    22     FEB    1732     -83223
JEFFERSON, THOMAS     13APR1743    13     APR    1743     -79155
FRANKLIN, BENJAMIN    17JAN1706    17     JAN    1706     -92755
```

**RIGHT-HAND SIDE USAGE**

**EXAMPLE 4: IN CONJUNCTION WITH THE INDEX FUNCTION**

In the above example, extracting dates was straight-forward since each component had a constant (and known) *starting position* and *length.* When the desired substring extraction point varies, it is often necessary to use a delimiter found within the string to determine the appropriate *starting position* or *length*. The appropriate values can be calculated dynamically using the INDEX function to find where to begin and end a substring. In the example data, the variable NAME contains the first and last names separated by a comma. The position of the comma will be used to determine the *length* of the last name and the *starting position* of the first name. For SURNAME subtract 1 from the *length* returned by the index function to avoid inclusion of the comma. To eliminate the comma from FNAME add 1 to the *starting position*. The *length* of the first name is unknown so use the default value for the third parameter. There

may or may not be a space after the comma, so the TRIM & LEFT functions are invoked to remove all extraneous spaces.

```
data namepart;
  set citizens(keep=name);
  length surname fname $20;
  surname = substr(name, 1, index(name, ',') - 1);
  fname   = trim(left(substr(name, index(name, ',') + 1)));
run;
```

```
EXAMPLE 4: IN CONJUNCTION WITH THE INDEX FUNCTION

       name             surname       fname

WASHINGTON,GEORGE     WASHINGTON    GEORGE
JEFFERSON, THOMAS     JEFFERSON     THOMAS
FRANKLIN, BENJAMIN    FRANKLIN      BENJAMIN
```

## USAGE ON BOTH SIDES

## EXAMPLE 5: CAPITALIZATION

SUBSTR can be used on both sides of the equal sign in the same statement. This is analogous to the 'x = x + 1' statement. To begin capitalization, SURNAME and FNAME are first converted to all lowercase letters. On the left-hand side only the first letter is modified, while on the right-hand side only the first letter is converted to uppercase letters.

```
data example;
  set namepart(drop=name);
  length surname2 fname2 $20;
  surname2 = lowcase(surname);
  substr(surname2, 1, 1) = upcase(substr(surname2, 1, 1));
  fname2   = lowcase(fname);
  substr(fname2, 1, 1)   = upcase(substr(fname2, 1, 1));
run
```

```
EXAMPLE 5: CAPITALIZATION

 surname       fname       surname2      fname2

WASHINGTON    GEORGE      Washington    George
JEFFERSON     THOMAS      Jefferson     Thomas
FRANKLIN      BENJAMIN    Franklin      Benjamin
```

This method is obsolete when using SAS version 9.1. A new function named PROPCASE has been added. Capitalization can now be performed in one simple step.

## PRACTICAL USAGE

## EXAMPLE 6: USAGE IN AN ALIGN MACRO

A great use for SUBSTR is aligning decimals. Notice how the following report output looks unprofessional. This is because the decimal points are not properly aligned. The BEFORE variable contains values with pre-determined decimal precision. If the variable BEFORE were of type numeric, then the decimal points would be aligned, but all precision would be lost. Conversely, a variable of character type retains the precision, but the alignment is lost. I think using SUBSTR is a clever method to resolve this dilemma.

```
EXAMPLE 6:  USAGE IN AN ALIGN MACRO

stat      before

N            26
Mean      110.2
SD        124.02
Min          7.0
```

```
Median      80.0
Max          600.0
```

❶ First find the position of the decimal using the INDEX function. Store this value in the variable DOT. If the value contains no decimal point then the value of DOT will be zero. ❷ Next take the length of the substring preceding the decimal point. LENINT stores the length of the integer portion. Third, ❸ determine the maximum length of LENINT for all observations by using the MAX function. ❹ After processing all observations, output the largest integer length, MAXINT, as a MACRO variable of the same name. A second DATA STEP calculates the difference, ❺ DIFFINT, between the MACRO variable MAXINT and LENINT. DIFFINT determines how many leading characters are needed to properly align the decimal points.

One is subtracted from DIFFINT because of the behavior of the REPEAT function. The arguments for the REPEAT function are (*string*, *n*). The value of TEMP after this statement, *'temp = repeat( '+', 0 )'* is '+'. Remember, the string value is always output once, even if zero is passed in! The value of the second parameter determines how many times the string is *repeated* - not how many times the string is output. A caveat of the REPEAT function is that the second argument must be greater than or equal to zero. Therefore step ❻ only performs the REPEAT function under these conditions. Step ❼ concatenates the pad characters to the front of the new variable named AFTER. The code and output appears below.

```
data outdata;
  set indata end=eof;
  retain maxint 0;

❶ dot = index(before, '.');
❷ if dot ne 0 then lenint = length(trim(left(substr(before, 1, (dot - 1)))));
  else lenint = length(trim(left(before)));

❸ maxint = max(maxint, lenint);
❹ if eof then call symput("maxint", maxint);
run;

data outdata;
  set outdata;
  length after $15;
  if stat ne '' and before  ne '' then do;
  ❺ diffint = &maxint - lenint - 1;
  ❻  if diffint >= 0 then do;
  ❼   after = repeat(" ", diffint)||trim(left(before));
    end;
    else do;
      after = trim(left(before));
    end;
  end;
run;
```

Output

```
EXAMPLE 6:  USAGE IN AN ALIGN MACRO

stat      before     after

N            26       26
Mean      110.2      110.2
SD        124.02     124.02
Min          7.0       7.0
Median     80.0       80.0
Max         600.0    600.0
```

## EXAMPLE 7: SIMPLE ENCRYPTION TECHNIQUES FOR CREATING CRYPTOGRAMS

A fun use for SUBSTR is to create cryptograms. Programming puzzles can challenge our programming skill set and provide an opportunity to sharpen our abilities. ❶ Begin by initializing the variable CODE to 26 dashes. Initializing it to blanks has caused problems in the past as some functions automatically trim leading blanks. ❷ Loop 26 times one for each letter of the alphabet. ❸ Generate random whole numbers between 1 and 26. ❹ Variable K will be sequentially ordered from 1 to 26, while X will be random numbers between 1 and 26. If K and X are the same number, then no encryption takes place. If this happens you want to return and generate another random number without incrementing K. ❺ If K and X are not equal a test must be performed to verify the value of X has not been previously assigned. If the number had been assigned then the value of CODE at *starting position* X would not be equal to a dash. ❻ Unfortunately, this method can create an endless loop if the last random number equals 26. Then the letter 'Z' would not be encrypted. The SUBSTR on the REVERSE of DECODE tests for this possibility. If the last letter is not encrypted the seed number is incremented by 2 and the program continues looping until DECODE is completely encrypted into CODE.

```
decode = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
seed = 12345;
j = 1;

leng = length(trim(left(decode)));
do while(j > 0);
❶ code=repeat('-', leng-1);
  k = 1;
❷ do until( k > leng);
❸   x = int(1+(leng)*ranuni(seed));
❹   if x ne k or k eq leng then do;
❺     if substr(code, x, 1) eq '-' then do;
        substr(code, x, 1) = substr(decode, k, 1);
        k + 1;
      end;
    end;
  end;

❻ if substr(code, leng, 1) eq substr(trim(left(reverse(decode))), 1, 1) then do;
    j + 1;
    seed + 2;
  end;
  else do;
    j = 0;
  end;
end;
```

The above code is the heart of a cryptogram program. The DATA SET containing the unencrypted text message was used to create the following cryptogram. It is the epitaph of a prominent Philadelphia citizen. Enjoy!

Solve this cryptogram to discover the epitaph of a prominent citizen of Philadelphia.

```
EXAMPLE 7:  SIMPLE ENCRYPTION TECHNIQUES FOR CREATING CRYPTOGRAMS


BDH RNIX NQ

R. QYTZJKUZ, WYUZBHY

KUJH BDH FNCHY NQ TZ NKI RNNJ

UBE FNZBHZBE BOYZ NOB

TZI EBYUWB NQ UBE KHBBHYUZM & MOUKIUZM

KUHE DHYH. QNNI QNY LNYGE

QNY, UB LUKK TE DH RHKUHCHI

TWWHTY NZFH GNYH

UZ T ZHL TZI GNYH HKHMTZB HIUBUNZ

FNYYHFBHI TZI UGWYNCHI

RX BDH TOBDNY

RHZATGUZ QYTZJKUZ

(FDYUEB FDOYFD ROYUTK MYNOZIE;

WDUKTIHKWDUT, WHZZEXKCTZUT)
```

## CONCLUSION

SUBSTR is not only useful for separating and manipulating text strings, but also for creating SAS date variables, checking MACRO parameter 'toggle' switches for 'Y' or 'N' values, aligning decimals in character variables, and for combining flags and rearranging or encrypting strings. The knowledge to perform these functions goes beyond the traditional use of SUBSTR, and requires an understanding of using SUBSTR function on all sides of the assignment operator.

## REFERENCES

Cody, Ron. December, 2004, "Using the PROPCASE function from '*SAS Functions by Example'* ", **http://support.sas.com/publishing/bbu_tip/cody_propcaseFunction.html**.

McDonald, Paul, D. November 1, 2002, "The SAS SUBSTR Function", www.**spikeware**.com/ftp/ppt/sw008.ppt, SPIKEware, Inc., Schaumburg, IL, USA.

SAS OnlineDoc version 8 (1999), SAS Language Reference Concepts: Expressions: SAS Operators in Expressions.

## CONTACT INFORMATION

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

David J. Austin
Quintiles, Inc.
Web: http://home.kc.rr.com/existentialist