How to Easily Remove the Last Character from a String in SAS

Authored by

stats writer

November 21, 2025

Working with character data, often referred to as strings, is an indispensable part of data management and cleansing within the SAS System. A very common requirement for data preprocessing is the need to remove an erroneous or trailing character from the end of a variable's value. In SAS, this precise modification is elegantly handled by combining the **SUBSTR()** function with the **LENGTH()** function. This pairing provides the dynamic control necessary to truncate the string by exactly one character, regardless of the original length of the data value.

The **SUBSTR()** function is specifically designed to extract a segment--or substring--from a source string. It requires three critical arguments: the name of the source string (the variable being modified), the starting position for the extraction, and the total length of the desired output. To successfully remove the last character, the key manipulation occurs in the third argument. By calculating the total length of the source string using the **LENGTH()** function and then subtracting one, we instruct SAS to return a substring that includes every character up to the penultimate position.

This approach ensures maximum generality and robustness. If the source string contains 15 characters, the LENGTH() function returns 15, and the calculation `15 - 1` results in 14. Consequently, **SUBSTR()** extracts the first 14 characters, effectively dropping the 15th and final character. This dynamic calculation is performed for every observation processed during the Data Step, making it an efficient solution for large-scale data cleaning operations where string lengths may vary significantly.

The Core Mechanism: Using SUBSTR() and LENGTH()

The easiest and most versatile way to remove the last character from a string in SAS is to use the **SUBSTR()** function paired with the **LENGTH()** function. This method is incorporated directly within a standard Data Step, allowing for variable assignment and dataset creation.

The following basic syntax demonstrates the implementation of this logic, typically used when reading from an `original_data` source and writing the cleaned results to a `new_data` target:

```
data new_data;
set original_data;
string_var = substr(string_var, 1, length(string_var)-1);
run;
```

This syntax efficiently executes the required transformation. It extracts the substring starting precisely at the first character (position 1) and continues up to the second to last character (determined by the length minus one). This calculated extraction successfully eliminates the final character, producing a clean output string. This methodology is preferred because it handles the

variable length of the source strings automatically, avoiding the need for conditional logic or manual character counting.

Detailed Breakdown of the Code Logic

The core statement, `string_var = SUBSTR(string_var, 1, LENGTH(string_var)-1);`, performs several critical operations sequentially within the iterative processing of the Data Step. First, SAS evaluates the inner function, `LENGTH()`, which returns an integer representing the total count of non-blank characters in the current value of `string_var`. This integer is then decremented by 1.

This resulting number--one less than the total length--is passed as the third argument to the **SUBSTR()** function. With the starting position fixed at 1 (the first character), **SUBSTR()** extracts exactly the required number of characters, omitting the final character. For example, if a variable holds "DataX" (length 5), the calculation is `5 - 1 = 4`, and **SUBSTR()** returns the first four characters: "Data".

It is important to ensure that the variable being modified is a character type. Furthermore, if the original variable had a fixed length defined (e.g., `$CHAR10`), the new value assigned to it will occupy the beginning of that length, and the remaining space will be padded with blanks, which is standard SAS character handling behavior. If the objective is to also remove any potentially resulting trailing blanks, a subsequent application of the **TRIM()** function might be necessary, though usually, **SUBSTR()** itself handles the returned length appropriately, avoiding excessive padding when assigning the result back to the same variable name.

Example: Remove Last Character from String in SAS

To provide a concrete illustration, let us use a scenario involving a dataset containing basketball team statistics. We hypothesize that data entry errors led to an unwanted trailing character ('x') being appended to the `team` variable for every observation. This character must be removed to restore data accuracy.

We first create the dataset, `my_data`, which contains the erroneous team names:

```
/*create dataset*/
data my_data;
input team $ points;
datalines;
Mavsx 113
Pacersx 95
Cavsx 120
```

Lakersx 114
Heatx 123
Kingsx 119
Raptorsx 105
Hawksx 95
Magicx 103
Spursx 119
;
run;

```
/*view dataset*/
proc print data=my_data;
```

The execution of this initial Data Step and subsequent `PROC PRINT` procedure results in the raw table shown below, clearly demonstrating the persistent trailing 'x' in the team names.

Obs	team	points
1	Mavsx	113
2	Pacersx	95
3	Cavsx	120
4	Lakersx	114
5	Heatx	123
6	Kingsx	119
7	Raptorsx	105
8	Hawksx	95
9	Magicx	103
10	Spursx	119

Notice carefully that each string value in the **team** column contains an 'x' as the last character, which requires automated removal before the data can be reliably analyzed.

Implementing the Transformation Logic

We proceed by applying the combined **SUBSTR()** and **LENGTH()** functions within a new Data Step. This code block generates a new dataset, `new_data`, where the team names are correctly formatted.

We use the **SUBSTR** function to dynamically remove this last character from each string in the **team** column, overwriting the variable with the cleaned value:

```
/*create new dataset where last character in each string of team column is removed*/
data new_data;
set my_data;
team = substr(team, 1, length(team)-1);
run;

/*view new dataset*/
proc print data=new_data;
```

The resulting output clearly demonstrates the successful application of the string manipulation logic. Every team name has been shortened by exactly one character.

Obs	team	points
1	Mavs	113
2	Pacers	95
3	Cavs	120
4	Lakers	114
5	Heat	123
6	Kings	119
7	Raptors	105
8	Hawks	95
9	Magic	103
10	Spurs	119

Notice that the final character ('x') has been removed from each string in the **team** column across all observations. This process highlights the efficiency of utilizing dynamic length calculation in SAS for character trimming tasks.

Detailed Review of the SUBSTR() Arguments

To fully grasp the functionality, a final review of the arguments used in the **SUBSTR()** function confirms why this method is so precise for last-character removal. The syntax structure is always **SUBSTR(Source, Position, N)**:

The **SUBSTR** function uses the following basic syntax:

SUBSTR(Source, Position, N)

where:

Source: The string variable (e.g., `team`) we are analyzing.
Position: The starting point for reading the substring, set to `1` to begin at the very start of the string.
N: The required number of characters to read, calculated as `LENGTH() - 1`.

By implementing **`substr(team, 1, length(team)-1)`**, we are instructing the SAS system to extract the substring from each value in the **team** column starting from the first character and ending precisely one position before the total length is reached. This calculated precision ensures that the last character is systematically excluded, regardless of the complexity or length of the source data.

Conclusion: Achieving Efficiency in String Handling

Mastering string manipulation is critical for maintaining data quality in SAS. The combination of the **SUBSTR()** function with the dynamic length calculation provided by **LENGTH()** offers the most reliable, efficient, and scalable solution for removing the last character from any character variable. This method bypasses complex conditional logic and provides a clear, concise statement that achieves the desired truncation effect across massive datasets with varying string lengths.

This technique is a cornerstone of effective data preparation in the SAS programming environment. Understanding how to leverage these intrinsic functions allows analysts and developers to quickly resolve common data cleanliness issues, such as extraneous delimiters or trailing codes, ensuring that datasets are perfectly structured for subsequent statistical analysis or reporting.

The following tutorials explain how to perform other common tasks in SAS: