SAS: Remove Last Character from String

Authored by

Mohammed looti

November 16, 2025

In advanced statistical computing and enterprise data management, proficiency in handling character data is essential, especially when utilizing robust software like SAS. A frequently encountered yet critical task during data preparation is the manipulation of text variables, often requiring the standardization of entries by removing extraneous characters. This comprehensive guide provides a precise and highly efficient solution to a common challenge: reliably removing the final character from a string within the SAS environment. Mastering this technique is fundamental for effective data cleaning, ensuring the integrity and precision necessary for rigorous statistical analysis and reporting.

The most robust and recommended approach for this operation hinges on the powerful synergy between two core SAS functions: the character extraction tool, **SUBSTR** (Substring), and the measurement tool, **LENGTH**. These functions work in concert to provide a dynamic and accurate methodology for extracting a specific portion of a character variable based on its calculated position and desired length. A deep conceptual understanding of how these functions interact is paramount for any data professional aiming to execute advanced string manipulation techniques flawlessly in SAS programming.

Throughout this article, we will meticulously dissect the mechanical principles governing these essential tools. We will then proceed to demonstrate their practical application through a detailed, real-world example, followed by a discussion of best practices for managing character variables in complex data projects. By the time you complete this guide, you will possess the knowledge to precisely trim trailing characters from any string variable across your SAS datasets, dramatically enhancing the reliability and efficiency of your overall data preparation workflow.

The Core SAS String Functions: SUBSTR and LENGTH

Before diving into the implementation details for removing the trailing character, it is vital to establish a crystal-clear understanding of the two primary functions that facilitate this operation: **SUBSTR** and **LENGTH**. These functions are cornerstones of SAS programming, utilized extensively for character data transformation, parsing, and formatting across various processing steps. Their efficient application allows programmers to interact with strings not merely as monolithic blocks of text, but as sequences of characters that can be precisely measured, indexed, and manipulated.

These tools are central to any task involving the modification or extraction of character components. Whether you are standardizing addresses, parsing complex identifiers, or, as in this case, performing targeted data cleaning, the ability to measure string length and extract specific sections based on position is indispensable. The combined use of **LENGTH** and **SUBSTR** offers a level of positional control that fixed-length extraction methods simply cannot match, providing adaptability crucial for handling real-world, variable-length data.

Detailed Overview of the SUBSTR Function

The SUBSTR function stands as the primary mechanism for extracting a specific sequence of characters, or a substring, from a larger character expression. Its operation is inherently positional, requiring three crucial arguments to accurately define the boundaries of the desired extraction. The general syntax, written as `SUBSTR(Source, Position, N)`, clearly specifies exactly which segment of the source string will be returned. A thorough understanding of each parameter's role is essential for achieving precise data manipulation:

Source: This mandatory input specifies the character variable or literal text from which the extraction process will originate.
Position: This numeric argument defines the exact starting point (index) for the extraction. It is crucial to remember that in SAS, character indexing begins at `1` for the very first character.
N: This optional numeric argument specifies the total count of characters to be extracted starting from the defined **Position**. If this parameter is intentionally omitted, **SUBSTR** defaults to extracting all remaining characters from the specified **Position** until the absolute end of the **Source** string.

For instance, if a SAS user executes the command `SUBSTR('ADVANCED', 4, 3)`, the resultant output would be 'VAN'. This example clearly demonstrates how the function initiates its count at the fourth character ('A') and proceeds to return the next three characters in sequence. This powerful and flexible capability ensures that **SUBSTR** is the core function utilized when our objective is to isolate a large section of a string while deliberately excluding the final character by limiting the extraction length.

The Importance of the LENGTH Function

The **LENGTH** function serves as the critical, complementary tool to **SUBSTR** in this specific operation. The LENGTH function is structurally simple, designed solely to return a numeric value that represents the byte length of a given character expression. Its straightforward syntax, `LENGTH(argument)`, accepts either a character variable or a literal string as input. For example, if a variable named `ProductCode` contains the value 'PCD-4590', executing `LENGTH(ProductCode)` would yield the numeric result `8`.

Despite its simplicity, the role of the **LENGTH** function becomes absolutely critical when processing strings that have variable lengths across observations. Since our goal is to remove the last character regardless of the string's total size, we cannot use a fixed extraction length. Instead, we must dynamically calculate the total number of characters, subtract one from that total, and pass this precisely calculated value to the **SUBSTR** function. This dynamic determination ensures that the operation is accurate, robust, and scalable, guaranteeing that exactly one character is removed from the tail of the string, whether the original string is composed of five characters or five

hundred. This precise calculation forms the mathematical and logical core of the complete character trimming solution.

Constructing the Essential SAS Syntax

With the roles of both **SUBSTR** and **LENGTH** clearly defined, we can now assemble the precise SAS code necessary for trimming the trailing character. This transformation is conventionally executed within a DATA step, which functions as the procedural engine in SAS for reading and transforming data observation by observation. The operation involves assigning a modified version of the existing character variable back to itself, ensuring that only the first N-1 characters are retained, where N represents the original length of the string.

The core logic dictates that we initiate the character extraction from the very first position (index 1) and continue the extraction for a length precisely equal to the total length of the string minus one. This dynamic calculation ensures that the extraction stops exactly before the final character, effectively discarding it. The following syntax represents the canonical and most efficient method for achieving this common data cleaning requirement:

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

A careful analysis of the key assignment line reveals the power and efficiency of this solution. The SET statement initiates the reading of data records sequentially. The assignment statement, `string_var = SUBSTR(...)`, then executes the transformation. Internally, `LENGTH(string_var)` first provides the total character count. Subtracting `1` yields the exact desired length for the truncated substring. Finally, the SUBSTR function performs the extraction, starting at position `1` and continuing up to the calculated length. This robust logic guarantees flawless execution across all observations in the dataset, provided the input string is not empty.

Practical Demonstration: Cleaning Trailing Data

To fully anchor this theoretical understanding, we will now apply the technique to a common, practical data preparation scenario. Consider a situation where data was imported from a legacy system or external source, resulting in an unwanted, uniform trailing character mistakenly appended to a descriptive field. In our example, we create a sample SAS dataset named `my_data` that details NBA teams and scores, where an erroneous 'x' character has been uniformly affixed to every team name in the `team` variable.

The following code snippet demonstrates the creation of this sample dataset, clearly illustrating the corrupted character variable that requires correction:

```
/*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;
```

When this code is successfully executed, the resulting output confirms the existence of the data anomaly:

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

As clearly visible in the output table, the trailing 'x' at the end of every value in the **team** column renders the data inaccurate and unsuitable for tasks such as linking with external lookup tables or generating standardized reports. This necessitates a precise data cleaning step. This specific requirement--removing a known number of characters from the absolute end of a string--is the ideal scenario for employing the combined power of the **SUBSTR** and **LENGTH** functions, allowing us to efficiently restore the original, accurate team names.

Executing the Transformation in a DATA Step

We now proceed to rigorously apply the derived formula to our sample dataset. The objective is to perform the character trimming operation within a new DATA step. This methodology ensures that the original `my_data` remains untouched, while a new, clean version, `new_data`, is created. This practice of preserving source data is strongly recommended in professional data management to guarantee traceability and reproducibility of all transformation results.

The code presented below demonstrates the implementation, where the `team` variable is immediately overwritten with the result of the combined **SUBSTR** and **LENGTH** calculation. This concise, single line of code dynamically and efficiently handles all string lengths across every observation:

```
/*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;
```

Upon successful execution of the DATA step, the subsequent output confirms the corrected and cleaned dataset:

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

This visualization definitively confirms that the transformation was a complete success: the redundant trailing character 'x' has been systematically eliminated from every entry in the **team** column. The resultant team names are now standardized and accurate. This powerful yet simple combination of **SUBSTR** and LENGTH function clearly underscores the remarkable efficiency of SAS for rapid and precise manipulation of character string variables, effectively preparing the data for subsequent analytical procedures or crucial reporting tasks.

Advanced Robustness and Alternative Methods

While the **SUBSTR** and **LENGTH** combination is undoubtedly the most efficient method for the targeted removal of the last character, a proficient SAS programmer must always account for potential edge cases and be aware of alternative tools for more complex manipulation tasks. Ensuring code robustness requires proactively anticipating data anomalies, such as empty fields or missing values.

For example, if a variable inherently holds an empty string (a character variable with no content), the LENGTH function correctly returns 0. Consequently, the calculation `LENGTH - 1` results in a negative number (-1). Fortunately, the SUBSTR function is deliberately engineered to handle such non-positive length arguments gracefully; it simply returns a null or empty string. This behavior

prevents the generation of runtime errors and ensures data consistency even when encountering zero-length or missing character fields, adding significant stability to the primary solution.

It is important to note that while this method is optimal for trimming the absolute last character, other scenarios might necessitate different specialized tools. If the primary goal is to remove trailing whitespace characters, the **TRIM** function is a much more direct and appropriate solution. Conversely, if the task involves removing a known pattern or suffix only when it appears at the end (e.g., removing ".dat" from a list of filenames), functions like **TRANWRD** or the powerful regular expression (PRX functions) should be considered. Nevertheless, for the specific, defined task of reducing the string length by exactly one character from the right, the combination of **SUBSTR** and **LENGTH** remains the gold standard due to its unmatched simplicity, efficiency, and processing speed.

Summary and Conclusion

The ability to efficiently manage and transform character variables is absolutely fundamental to conducting reliable data cleaning and subsequent statistical analysis within SAS. The specific requirement to remove the final character of a string is a ubiquitous data preparation task, often driven by formatting inconsistencies or errors originating during data import. As this guide has comprehensively demonstrated, the synergistic application of the **SUBSTR** and LENGTH function provides the most direct, elegant, and reliable solution available in the SAS toolset.

The entire success of this technique hinges on the dynamic calculation: **LENGTH** determines the full extent of the string, and **SUBSTR** utilizes that exact measurement, minus one, to accurately extract the desired substring. This methodology ensures that the code adapts seamlessly and correctly to character variables of any size. By mastering this core function combination, data professionals can guarantee that their character data is accurately formatted and prepared, leading directly to more trustworthy statistical outcomes and consistently high-quality reporting.

We strongly recommend integrating this precise technique into your regular data preparation repertoire. Further exploration of the comprehensive SAS string functions documentation will significantly broaden your overall data manipulation capabilities, solidifying your expertise in advanced SAS programming.

Additional Resources

For those interested in further expanding their SAS expertise and mastering related data transformation techniques, the following tutorials explain how to perform other common tasks and delve deeper into advanced functionalities:

SAS String Functions Documentation

Advanced Data Cleaning Techniques in SAS
Introduction to Regular Expressions in SAS