The Stata Journal (2011)
11, Number 2, pp. 318–320

Stata tip 98: Counting substrings within strings

Nicholas J. Cox
Department of Geography
Durham University
Durham, UK
n.j.cox@durham.ac.uk

Consider the following problem, based on a real one reported on Statalist. A user has string date–times in differing forms; let us imagine they are in a variable, `sdate`. The desire is for everything to be `DMY hms`—that is, day, month, year, space, hours, minutes, and seconds. Examples of data are

“25/12/2010 11:22”
“25/12/2010 11:22:33”
“25/12/2010 11:22:33:444”

Compared with the standard, the first example lacks seconds; the second example is fine; and the third example includes milliseconds, but expressed in a nonstandard way. Stata expects seconds and milliseconds in the decimal form “33.444”.

Some cleanup is required. One way forward is to note that the number of colons (`:`) present within the string is diagnostic of whether action is required and what to do. So how do you count substrings (in this case, just a colon) within strings? The problem has been aired in this journal at least once before (Cox 2011), but it occurs sufficiently often that it deserves a further flag. The main solution tends to provoke the reaction “Yes, of course!”, whether because people know it already or they now see that it is obvious and direct. However, people (including myself) have often supposed that the problem requires a more complicated approach than it really does.

Faced with this kind of problem, an experienced user tends to browse the help for string functions to see if there is a function dedicated to this problem, but in this case browsing will be in vain. But as so often happens, combining different functions is more successful. Consider how you would count for yourself. You would naturally work from one end of the string to the other, noting each occurrence. It is immaterial whether you count from left to right or from right to left, but note that Stata, by default, works on strings from left to right.

Stata has a function, `subinstr()`, that looks for occurrences of substrings within strings and replaces them with a specified substring (often just an empty string, `""`). This function gives us a solution. Consider the calculation

```
length(sdate) - length(subinstr(sdate, ":", "", .))
```

As with elementary algebra, working from the inside of complicated expressions outward is a good tactic for understanding. The function call

```
subinstr(sdate, ":", "", .)
```

takes in this example a named variable, `sdate`, and substitutes an empty string, `""`, for every occurrence of the single-character substring that is a colon, `":"`. In other words, it deletes every colon. The length of the result is shorter than the original by how many colons were found, which could be 0, 1, 2, 3, and so forth.

We do not always need to get Stata to do this and remove those substrings. In this example, and in many others, doing so would just mess up our data and make our problem more difficult. We just need to get Stata to tell us what the result would be. So, to conclude the example, we could count the colons, fix problem cases, and check to see if that strategy worked everywhere:

```
generate ncolons = length(sdate) - length(subinstr(sdate, ":", "", .))
replace sdate = sdate + ":30" if ncolons == 1
replace sdate = reverse(subinstr(reverse(sdate), ":", ".", 1)) if ncolons == 3
generate double ndate = clock(sdate, "DMY hms")
format ndate %tC
list sdate if missing(ndate)
```

The trickiest detail here is that to change the last colon of three to a period, we reverse the string, change the first colon, and reverse it back again. This is another common example of combining string functions.

Counting substrings that are two or more characters long introduces two extra twists to the problem.

First, let us focus on a standard elementary puzzle. How many occurrences of `"ana"` are there in `"banana"`? One tricksy answer is two—namely, `"*ana**"` and `"***ana"`—but Stata's answer using the `subinstr()` trick will have to be one. Once the first `"ana"` is blanked out, `"bna"` is all that is left. It seems rare in data management that we want the more generous answer, but how would we do it?

In essence, we need to test each possible position. For that, we need to know the length of the string—in our example, a string variable (say, `svar`)—to be searched. Either we know the precise type of a string variable in advance—say, `str12`—or we will need to look it up using an extended macro function. Let's choose the second and marginally more difficult case for a string variable. We will keep going with the silly example of counting as many occurrences of `"ana"` as possible.

```
local length = substr("`: type svar´", 4, .) - length("ana") + 1
gen ana_count = 0
qui forval i = 1/`length´ {
        replace ana_count + (strpos(svar, "ana") == `i´)
}
```

How does this approach work? We are testing each possible position. We start at the first character, but we need not go all the way up to the end, because for example, the last possible position in which `"ana"` could occur within a `str12` is position 10 (not position 9!) We initialize a count variable as 0 and bump it up by 1 every time the position is indeed a starting position for the substring. That follows from the expression `(strpos(svar, "ana") == ‘i’)` evaluating to `1` when true and `0` when false, so we end up counting occurrences.

That case is clearly more complicated than the case of counting occurrences disjointly. Fortunately, it appears to arise much less frequently.

Second, remember to adjust for the length of substring when counting substrings using the `subinstr()` method. `length("banana") - length(subinstr("banana", "ana", "", .))` yields `3` because three characters were removed, but just one occurrence of the substring was. In general, divide by `length("`*substring*`")`. If you know what it is, there is a small efficiency gain in saying so. For example, you should not write

```
gen n_ana = (length(svar) - length(subinstr(svar, "ana", "", .))) / length("ana")
```

because that obliges Stata to calculate `length("ana")` for every observation. Even in a programming situation where the particular substring is not predictable in advance, a calculation like

```
local sslen = length("substring")
```

allows the result to be used in macro form in a command:

```
gen n_substring = (length(svar) - ///
  length(subinstr(svar, "substring", "", .))) / `sslen´
```

That way, the contents of the macro are substituted before the `generate` command gets to work so that it uses a known constant rather than an expression to be evaluated for every observation.

The problem could get more complicated, yet. For example, the substring concerned might vary from observation to observation, as when conventions about reporting are variable in the data. So long as the substring is known and included within a variable (say, `ssvar`), this is no more difficult than any previous problem.

```
gen sscount = (length(svar) - ///
  length(subinstr(svar, ssvar, "", .))) / length(ssvar)
```

Alternatively, we might be counting different possible types of substring, for which we just need to cycle over all the possibilities.

Reference

Cox, N. J. 2011. Speaking Stata: MMXI and all that: Handling Roman numerals within Stata. *Stata Journal* 11: 126–142.