

The coolstr package*

nsetzer

January 10, 2007

The coolstr package is a “sub” package of the cool package that seemed appropriate to publish independently since it may occur that one wishes to include the ability to check strings without having to accept all the overhead of the cool package itself.

1 Basics

Strings are defined as a sequence of characters (not TeX tokens). The main purpose behind treating strings as characters rather than tokens is that one can then do some text manipulation on them.

2 Descriptions

`\substr` `\substr{⟨string⟩}{⟨start index⟩}{⟨num char⟩}` gives at most $\|⟨num char⟩\|$ characters from `⟨string⟩`.

if `⟨start index⟩` is greater than zero, and `⟨num char⟩` is greater than zero, `\substr` gives at most `⟨num char⟩` starting with index `⟨start index⟩` and going to the end of the string.

if `⟨start index⟩` is greater than zero, and `⟨num char⟩` is less than zero, `\substr` gives at most $-⟨num char⟩$ characters and going to the beginning of the string

if `⟨start index⟩` is less than zero, and `⟨num char⟩` is greater than zero, `\substr` gives at most `⟨num char⟩` characters starting at the $-⟨start index⟩$ character from the end of the string and going to the end of the string

if `⟨start index⟩` is less than zero, and `⟨num char⟩` is less than zero, `\substr` gives at most $-⟨num char⟩$ characters starting at the $-⟨start index⟩$ character from the end of the string and going to the beginning of the string

There are two special, non-numeric values that `⟨char num⟩` may take. They are `end` or `beg`, and they will always go to the end or beginning of the string, respectively

3 Test Cases

3.1 `\substr`

`\substr`

*This document corresponds to cool v2.1, dated 2007/01/08.

<code>\substr{12345}{1}{2}</code>	12
<code>\substr{12345}{3}{5}</code>	345
<code>\substr{12345}{3}{end}</code>	345
<code>\substr{12345}{3}{beg}</code>	123
<code>\substr{12345}{-2}{1}</code>	4
<code>\substr{12345}{3}{-2}</code>	23
<code>\substr{12345}{-2}{-2}</code>	34
<code>\substr{12345}{0}{5}</code>	(the null string)
<code>\substr{12345}{2}{0}</code>	(the null string)

3.2 `\isdecimal`

2.345	is decimal
2.4.5	not a decimal
+ -2.45	not a decimal
+2.345	is decimal
-2.345	is decimal
2.345-	not a decimal
2.4+4.	not a decimal
+4.	is decimal
4.	is decimal
+ .7	is decimal
.3	is decimal
4	is decimal
	<code>\newcommand{\numberstore}{4.5}</code>
<code>\numberstore</code>	is decimal

3.3 `\isnumeric`

4.5	is numeric
4.5e5	is numeric
+4.5e5	is numeric
4.5e+5	is numeric
+4.5e+5	is numeric
4.5E5	is numeric
-4.5E5	is numeric
4.5E-5	is numeric
-4.5E-5	is numeric
4.5.E-5	not numeric
abcdefg	not numeric
abcE-5	not numeric

3.4 \isint

```

4          is integer
+4         is integer
4.5        not integer
4.5e5      not integer
+4.5e5     not integer
4.5e+5     not integer
+4.5e+5    not integer
4.5E5      not integer
-4.5E5     not integer
4.5E-5     not integer
-4.5E-5    not integer
4.5.E-5    not integer
abcdefg    not integer
abcE-5     not integer
           \renewcommand{\numberstore}{4}
\numberstore is integer

```

4 Acknowledgments

Thanks to J. J. Weimer for the comments and aid in coding. Also thanks goes to Abraham Weishaus for pointing out a bug in `\strlenstore`

5 Implementation

This is just an internal counter for dealing with the strings; most often used for the length

```
1 \newcounter{COOL@strlen}%
```

`\setstrEnd` `\setstrEnd{<string>}` allows the user to set the end of a string ‘character’ in the rare event that the default value actually appears in the string. The default value is

```

2 \newcommand{\COOL@strEnd}{\%\%\%}
3 \newcommand{\COOL@intEnd}{\%@\%@\%@}
4 \let\COOL@strStop=\relax

```

and may be changed by the following command (which utilizes the `\renewcommand`):

```
5 \newcommand{\setstrEnd}[1]{\renewcommand{\COOL@strEnd}{#1}}
```

This area defines the core technology behind the `coolstr` package: the string “gobbler”.

```
6 \newcounter{COOL@strpointer}
```

Now we come to “the gobbler”—a recursive function that eats up a string. It must be written in \TeX primitives.

The idea behind this is that “the gobbler” eats up everything before the desired character and everything after the desired character.

```

7 \def\COOL@strgobble[#1]#2#3{%
8 \ifthenelse{equal{#3}{\COOL@strEnd}}{%

```

```

9 {%
10 \ifthenelse{\value{COOL@strpointer}=#1}%
11 {%
12 #2%
13 }%
14 % Else
15 {%
16 }%
17 }%
18 % Else
19 {%
20 \ifthenelse{\value{COOL@strpointer}=#1}%
21 {%
22 #2%
23 }%
24 % Else
25 {%
26 }%
27 \stepcounter{COOL@strpointer}%
28 \COOL@strgobble[#1]#3%
29 }%
30 }

```

`\strchar` `\strchar{⟨index⟩}` gives the *⟨index⟩* character of the string. Strings start indexing at 1.

```

31 \newcommand{\strchar}[2]{%
32 \setcounter{COOL@strpointer}{1}%
33 \COOL@strgobble[#2]#1\COOL@strEnd%
34 }

```

`\strlen` `\strlen{⟨string⟩}` gives the length of the string. It is better to use `\strlenstore` to record the length

```

\strlen{abc} 3

```

```

35 \newcommand{\strlen}[1]{%
36 \ifthenelse{\equal{#1}{}}{%
37 {%
38 0%
39 }%
40 % Else
41 {%
42 \strchar{#1}{0}%
43 \arabic{COOL@strpointer}%
44 }%
45 }

```

`\strlenstore` `\strlenstore{⟨string⟩}{⟨counter⟩}` stores the length of *⟨string⟩* in *⟨counter⟩*

```

46 \newcommand{\strlenstore}[2]{%
47 \ifthenelse{\equal{#1}{}}{%
48 {%
49 \setcounter{#2}{0}%
50 }%
51 % Else
52 {%

```

```

53 \strchar{#1}{0}%
54 \setcounter{#2}{\value{COOL@strpointer}}%
55 }%
56 }

```

`\substr` `\substr{<string>}{<index>}{<numchar>}`

a special value of `end` for `<numchar>` gives from `<index>` to the end of the string;
`beg` gives from `<index>` to the beginning of the string

```

57 \newcounter{COOL@str@index}
58 \newcounter{COOL@str@start}
59 \newcounter{COOL@str@end}
60 \newcommand{\substr}[3]{%
61 \strlenstore{#1}{COOL@strlen}%
62 \ifthenelse{#2 < 0 \AND \NOT #2 < -\value{COOL@strlen}}%
63 {%

```

The starting index is less than zero, so start that many characters back from the end. This means mapping the index to `<index> + <string length> + 1`

```

64 \setcounter{COOL@str@index}{\value{COOL@strlen}}%
65 \addtocounter{COOL@str@index}{#2}%
66 \addtocounter{COOL@str@index}{1}%
67 }%
68 % ElseIf
69 {\ifthenelse{#2 > 0 \AND \NOT #2 > \value{COOL@strlen}}%
70 {%

```

The starting index is greater than zero, and within the appropriate range; record it

```

71 \setcounter{COOL@str@index}{#2}%
72 }%
73 % Else
74 {%
75 %     \end{macrocode}
76 % The \meta{index} value is invalid. Set it to zero for returning the null string
77 %     \begin{macrocode}
78 \setcounter{COOL@str@index}{0}%
79 }%

```

Now deal with the `<numchar>` (which can also be negative)

```

80 \ifthenelse{\equal{#3}{beg}}%
81 {%
82 \setcounter{COOL@str@start}{1}%
83 \setcounter{COOL@str@end}{\value{COOL@str@index}}%
84 }%
85 % ElseIf
86 {\ifthenelse{\equal{#3}{end}}%
87 {%
88 \setcounter{COOL@str@start}{\value{COOL@str@index}}%
89 \setcounter{COOL@str@end}{\value{COOL@strlen}}%
90 }%
91 % ElseIf
92 {\ifthenelse{#3 < 0}%
93 {%

```

This means to take that many characters to the *left* of the starting index.

```

94 \setcounter{COOL@str@start}{\value{COOL@str@index}}%
95 \addtocounter{COOL@str@start}{#3}%
96 \addtocounter{COOL@str@start}{1}%
97 \ifthenelse{\NOT \value{COOL@str@start} > 0}{\setcounter{COOL@str@start}{1}}{}%
98 \setcounter{COOL@str@end}{\value{COOL@str@index}}%
99 }%
100 % ElseIf
101 {\ifthenelse{#3 > 0}%
102 {%
103 \setcounter{COOL@str@start}{\value{COOL@str@index}}%
104 \setcounter{COOL@str@end}{\value{COOL@str@index}}%
105 \addtocounter{COOL@str@end}{#3}%
106 \addtocounter{COOL@str@end}{-1}%
107 \ifthenelse{\value{COOL@str@end} > \value{COOL@strlen}}{\setcounter{COOL@str@end}{\value{COOL@str
108 }}%
109 % Else
110 {%
111     nonsense submitted, so return the null string
112 \setcounter{COOL@str@index}{0}%
113 }}}}%
114
115 Now send back the appropriate thing
116 \ifthenelse{ \value{COOL@str@index} = 0 }%
117 {%
118 \setcounter{COOL@strpointer}{1}%
119 \COOL@substrgobbler#1\COOL@strStop\COOL@strEnd%
120 }%
121 }
122
123 Now define the “gobbler”
124 \def\COOL@substrgobbler#1#2\COOL@strEnd{%
125 \ifthenelse{\equal{#2}{\COOL@strStop}}%
126 {%
127 % Else
128 {%
129 #1%
130 }%
131 }%
132 % Else
133 {%
134 \ifthenelse{ \value{COOL@strpointer} < \value{COOL@str@start} \OR \value{COOL@strpointer} > \valu
135 }{}%
136 % Else
137 {%
138 #1%
139 }%
140 \stepcounter{COOL@strpointer}%
141 \COOL@substrgobbler#2\COOL@strEnd%
142 }%
143 }

```

Define a new boolean for comparing characters

```
144 \newboolean{COOL@charmatch}
```

\COOL@strcomparegobble This “gobbler” does character comparison

```
145 \def\COOL@strcomparegobble[#1]<#2>#3#4{%
146 \ifthenelse{\equal{#4}{\COOL@strEnd}}{%
147 {%
148 \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }{%
149 {%
150 \setboolean{COOL@charmatch}{true}%
151 }%
152 % Else
153 {%
154 }%
155 }%
156 % Else
157 {%
158 \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }{%
159 {%
160 \setboolean{COOL@charmatch}{true}%
161 }%
162 % Else
163 {%
164 }%
165 \stepcounter{COOL@strpointer}%
166 \COOL@strcomparegobble[#1]<#2>#4%
167 }%
168 }
```

\ifstrchareq \ifstrchareq{<string>}{<char index>}{<comparison char>}{<do if true>}{<do if false>}

```
169 \newcommand{\ifstrchareq}[5]{%
170 \setboolean{COOL@charmatch}{false}%
171 \setcounter{COOL@strpointer}{1}%
172 \COOL@strcomparegobble[#2]<#3>#1\COOL@strEnd\relax%
173 \ifthenelse{ \boolean{COOL@charmatch} }{%
174 {%
175 #4%
176 }%
177 % Else
178 {%
179 #5%
180 }%
181 }
```

\ifstrlneq \ifstrlneq{<string>}{<number>}{<do if true>}{<do if false>}
\ifstrlneq{abc}{3}{length is \$3\$}{length is not \$3\$} length is 3
\ifstrlneq{abcde}{3}{length is \$3\$}{length is not \$3\$} length is not 3

```
182 \newcommand{\ifstrlneq}[4]{%
183 \strlenstore{#1}{COOL@strlen}%
184 \ifthenelse{ \value{COOL@strlen} = #2 }{%
185 {%
186 #3%
```

```

187 }%
188 % Else
189 {%
190 #4%
191 }%
192 }

\COOL@decimalgobbler This “gobbler” is used to determine if the submitted string is a rational number
(satisfies  $d_n d_{n-1} \cdots d_1 d_0 . d_{-1} d_{-2} \cdots d_{-m}$ ). The idea behind the macro is that it
assumes the string is rational until it encounters a non-numeric object

193 \newboolean{COOL@decimalfound}
194 \newboolean{COOL@decimal}

COOL@decimalfound is a boolean indicating if the first decimal point is found
COOL@decimal is the flag that tells if the string contains numeric data

195 \def\COOL@decimalgobbler#1#2\COOL@strEnd{%
196 \ifthenelse{\equal{#2}{\COOL@strStop}}}%

this indicates we are at the end of the string. We only need to perform the check
to see if the digit is a number or the first decimal point

197 {%
198 \ifthenelse{‘#1 < ‘0 \OR ‘#1 > ‘9}%
199 {%
200 \ifthenelse{ ‘#1 = ‘. \AND \NOT \value{COOL@strpointer} = 1 \AND \NOT \boolean{COOL@decimalfound}
201 {%
202 }%
203 % Else
204 {%
205 \setboolean{COOL@decimal}{false}%
206 }%
207 }%
208 % Else
209 {%
210 }%
211 }%
212 % Else
213 {%
214 \ifthenelse{ ‘#1 < ‘0 \OR ‘#1 > ‘9 }%
215 {%

not at the end of a string, and have encountered a non-digit. If it is a number,
then this non digit must be the first decimal point or it may be the first character
and a + or – sign

216 \ifthenelse{ ‘#1 = ‘. \AND \NOT \boolean{COOL@decimalfound} }%
217 {%
218 \setboolean{COOL@decimalfound}{true}%
219 }%
220 {\ifthenelse{ \('#1 = ‘+ \OR ‘#1 = ‘- \) \AND \value{COOL@strpointer} = 1 }%
221 {%
222 }%
223 % Else
224 {%
225 \setboolean{COOL@decimal}{false}%
226 }%

```



```

227 }%
228 % Else
229 {%
230 \stepcounter{COOL@strpointer}%
231 \COOL@decimalgobbler#2\COOL@strEnd%
232 }%
233 }

\isdecimal  isdecimal{<string>}{<boolean>}

234 \newcommand{\isdecimal}[2]{%
235 \setcounter{COOL@strpointer}{1}%
236 \setboolean{COOL@decimalfound}{false}%
237 \setboolean{COOL@decimal}{true}%
238 \expandafter\COOL@decimalgobbler#1\COOL@strStop\COOL@strEnd%
239 \ifthenelse{ \boolean{COOL@decimal} }%
240 {%
241 \setboolean{#2}{true}%
242 }%
243 % Else
244 {%
245 \setboolean{#2}{false}%
246 }%
247 }%

\isnumeric  \isnumeric{<string>}{<boolean>} stores true in <boolean> if <string> is numeric

248 \newboolean{COOL@numeric}%
249 \def\COOL@eparser#1e#2\COOL@strEnd{%
250 \xdef\COOL@num@magnitude{#1}%
251 \xdef\COOL@num@exponent{#2}%
252 }
253 \def\COOL@ecorrector#1e\COOL@strStop{%
254 \xdef\COOL@num@exponent{#1}%
255 }
256 \def\COOL@Eparser#1E#2\COOL@strEnd{%
257 \xdef\COOL@num@magnitude{#1}%
258 \xdef\COOL@num@exponent{#2}%
259 }
260 \def\COOL@Ecorrector#1E\COOL@strStop{%
261 \xdef\COOL@num@exponent{#1}%
262 }
263 \newcommand{\isnumeric}[2]{%
264 \COOL@eparser#1e\COOL@strStop\COOL@strEnd%
265 \ifthenelse{ \equal{\COOL@num@exponent}{\COOL@strStop} }%
266 {%
267 \COOL@Eparser#1E\COOL@strStop\COOL@strEnd%
268 \ifthenelse{ \equal{\COOL@num@exponent}{\COOL@strStop} }%
269 {%
270 \gdef\COOL@num@exponent{0}%
271 }%
272 % Else
273 {%
274 \expandafter\COOL@Ecorrector\COOL@num@exponent%
275 }%
276 }

```

```

277 % Else
278 {%
279 \expandafter\COOL@ecorrector\COOL@num@exponent%
280 }%
281 \isdecimal{\COOL@num@magnitude}{\COOL@numeric}%
282 \ifthenelse{ \boolean{\COOL@numeric} }%
283 {%
284 \isdecimal{\COOL@num@exponent}{\COOL@numeric}%
285 \ifthenelse{ \boolean{\COOL@numeric} }%
286 {%
287 \setboolean{#2}{true}%
288 }%
289 % Else
290 {%
291 \setboolean{#2}{false}%
292 }%
293 }%
294 % Else
295 {%
296 \setboolean{#2}{false}%
297 }%
298 }

```

In addition to identifying numeric data, it is useful to know if integers are present, thus another “gobbler” is needed

```

299 \newboolean{COOL@isint}
300 \def\COOL@intgobbler#1#2\COOL@strEnd{%
301 \ifcat#11%
302 \ifthenelse{\equal{#2}{\COOL@strStop}}%
303 {%
304 \ifthenelse{‘#1 < ‘0 \OR ‘#1 > ‘9}%
305 {%
306 \setboolean{COOL@isint}{false}%
307 }%
308 % Else
309 {%
310 }%
311 }%
312 % Else
313 {%
314 \ifthenelse{ ‘#1 < ‘0 \OR ‘#1 > ‘9 }%
315 {%
316 \ifthenelse{ ‘#1 = ‘+ \OR ‘#1 = ‘- \AND \value{COOL@strpointer} = 1 }%
317 }%
318 % Else
319 {%
320 \setboolean{COOL@isint}{false}%
321 }%
322 }%
323 % Else
324 {%
325 }%
326 \stepcounter{COOL@strpointer}%
327 \COOL@intgobbler#2\COOL@strEnd%

```

```

328 }%
329 \else%
330 \setboolean{COOL@isint}{false}%
331 \fi%
332 }

\isint \isint{<string>}{<boolean>} sets the <boolean> to true if <string> is an integer or
false otherwise

333 \newcommand{\isint}[2]{%
334 \setcounter{COOL@strpointer}{1}%
335 \setboolean{COOL@isint}{true}%
336 \COOL@intgobbler#1\COOL@strStop\COOL@strEnd%
337 \ifthenelse{ \boolean{COOL@isint} }{%
338 {%
339 \setboolean{#2}{true}%
340 }%
341 % Else
342 {%
343 \setboolean{#2}{false}%
344 }%
345 }

```

Change History

v1.0	boolean	11
General: Initial Release	1	\isnumeric: added extra mandatory argument for storing return
v2.0	boolean	9
General: Added three new commands: ifstrchareq, ifstrlen, strlen	1	\strlen: added to package 4
\COOL@decimalgobbler: added this “gobbler” to complete isnumeric	8	\strlenstore: added to package 4
\COOL@strcomparegobble: added to package for single character comparisons	7	v2.0a
\ifstrchareq: added to package to do character comparing	7	\isint: modified internals slightly to work with cool package 11
\ifstrlen: added to package to do length comparison	7	v2.1
\isdecimal: added	9	\ifstrlen: altered function to use strlenstore 7
\isint: added extra mandatory argument for storing return		\strlen: added ifthenelse to return 0 for empty string 4
		\strlenstore: added ifthenelse to return 0 for empty string 4
		corrected error in setting counter 4
		\substr: added to package 5

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

\% 2, 3

C		260, 264, 265, 267, 268, 302, 336
\COOL@decimalgobbler	<u>193</u> , 238
\COOL@Ecorrector	260, 274
\COOL@ecorrector	253, 279
\COOL@Eparser	256, 267
\COOL@eparser	249, 264
\COOL@intEnd	3
\COOL@intgobbler	300, 327, 336
\COOL@num@exponent	.	251, 254, 258,
		261, 265, 268, 270, 274, 279, 284
\COOL@num@magnitude	...	250, 257, 281
\COOL@strcomparegobble	<u>145</u> , 172
\COOL@strEnd	..	2, 5, 8, 33, 119, 122,
		141, 146, 172, 195, 231, 238,
		249, 256, 264, 267, 300, 327, 336
\COOL@strgobble	7, 28, 33
\COOL@strStop	
	...	4, 119, 123, 196, 238, 253,
I		
\ifstrchareq	<u>169</u>
\ifstrleneq	<u>182</u>
\isdecimal	<u>234</u> , 281, 284
\isint	<u>333</u>
\isnumeric	<u>248</u>
M		
\meta	76
S		
\setstrEnd	<u>2</u>
\strchar	<u>31</u> , 42, 53
\strlen	<u>35</u>
\strlenstore	<u>46</u> , 61, 183
\substr	<u>1</u> , <u>57</u>