

The `coolstr` package*

nsetzer

December 30, 2006

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 Test Cases

2.1 `\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

*This document corresponds to `cool` v2.0a, dated 2006/12/30.

2.2 `\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

2.3 `\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
	<code>\renewcommand{\numberstore}{4}</code>
<code>\numberstore</code>	is integer

3 Acknowledgments

Thanks to J. J. Weimer for the comments and aid in coding.

4 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 $\langle index \rangle$ 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 \strchar{#1}{0}
37 \arabic{COOL@strpointer}%
38 }
```

`\strlenstore` `\strlenstore{<string>}{<counter>}` stores the length of `<string>` in `<counter>`

```
39 \newcommand{\strlenstore}[2]{%
40 \strchar{#1}{0}%
41 \setcounter{#2}{COOL@strpointer}%
42 }
```

Define a new boolean for comparing characters

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

`\COOL@strcomparegobble` This “gobbler” does character comparison

```
44 \def\COOL@strcomparegobble[#1]<#2>#3#4{%
45 \ifthenelse{\equal{#4}{\COOL@strEnd}}{%
46 {%
47 \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }{%
48 {%
49 \setboolean{COOL@charmatch}{true}%
50 }%
51 % Else
52 {%
53 }%
54 }%
55 % Else
56 {%
57 \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }{%
58 {%
59 \setboolean{COOL@charmatch}{true}%
60 }%
61 % Else
62 {%
63 }%
64 \stepcounter{COOL@strpointer}%
65 \COOL@strcomparegobble[#1]<#2>#4%
66 }%
67 }
```

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

```
68 \newcommand{\ifstrchareq}[5]{%
69 \setboolean{COOL@charmatch}{false}%
70 \setcounter{COOL@strpointer}{1}%
71 \COOL@strcomparegobble[#2]<#3>#1\COOL@strEnd\relax%
72 \ifthenelse{ \boolean{COOL@charmatch} }{%
73 {%
74 #4%
75 }%
76 % Else
77 {%
78 #5%
79 }%
80 }
```

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

\ifstrleneq{abcde}{3}{length is \$3\$}{length is not \$3\$} length is not 3

```

81 \newcommand{\ifstrleneq}[4]{%
82 \strchar{#1}{0}%
83 \ifthenelse{ \value{COOL@strpointer} = #2 }{%
84 {%
85 #3%
86 }%
87 % Else
88 {%
89 #4%
90 }%
91 }

```

\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

```

92 \newboolean{COOL@decimalfound}
93 \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

```

94 \def\COOL@decimalgobbler#1#2\COOL@strEnd{%
95 \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

```

96 {%
97 \ifthenelse{‘#1 < ‘0 \OR ‘#1 > ‘9}%
98 {%
99 \ifthenelse{ ‘#1 = ‘. \AND \NOT \value{COOL@strpointer} = 1 \AND \NOT \boolean{COOL@decimalfound}
100 {%
101 }%
102 % Else
103 {%
104 \setboolean{COOL@decimal}{false}%
105 }%
106 }%
107 % Else
108 {%
109 }%
110 }%
111 % Else
112 {%
113 \ifthenelse{ ‘#1 < ‘0 \OR ‘#1 > ‘9 }%
114 {%

```

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

```

115 \ifthenelse{ ‘#1 = ‘. \AND \NOT \boolean{COOL@decimalfound} }%
116 {%
117 \setboolean{COOL@decimalfound}{true}%
118 }%
119 {\ifthenelse{ \(< ‘#1 = ‘+ \OR ‘#1 = ‘- \) \AND \value{COOL@strpointer} = 1 }%

```

```

120 {%
121 }%
122 % Else
123 {%
124 \setboolean{COOL@decimal}{false}%
125 }}%
126 }%
127 % Else
128 {%
129 \stepcounter{COOL@strpointer}%
130 \COOL@decimalgobbler#2\COOL@strEnd%
131 }%
132 }

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

133 \newcommand{\isdecimal}[2]{%
134 \setcounter{COOL@strpointer}{1}%
135 \setboolean{COOL@decimalfound}{false}%
136 \setboolean{COOL@decimal}{true}%
137 \expandafter\COOL@decimalgobbler#1\COOL@strStop\COOL@strEnd%
138 \ifthenelse{ \boolean{COOL@decimal} }{%
139 {%
140 \setboolean{#2}{true}%
141 }%
142 % Else
143 {%
144 \setboolean{#2}{false}%
145 }%
146 }%

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

147 \newboolean{COOL@numeric}%
148 \def\COOL@eparser#1e#2\COOL@strEnd{%
149 \xdef\COOL@num@magnitude{#1}%
150 \xdef\COOL@num@exponent{#2}%
151 }
152 \def\COOL@ecorrector#1e\COOL@strStop{%
153 \xdef\COOL@num@exponent{#1}%
154 }
155 \def\COOL@Eparser#1E#2\COOL@strEnd{%
156 \xdef\COOL@num@magnitude{#1}%
157 \xdef\COOL@num@exponent{#2}%
158 }
159 \def\COOL@Ecorrector#1E\COOL@strStop{%
160 \xdef\COOL@num@exponent{#1}%
161 }
162 \newcommand{\isnumeric}[2]{%
163 \COOL@eparser#1e\COOL@strStop\COOL@strEnd%
164 \ifthenelse{ \equal{\COOL@num@exponent}{\COOL@strStop} }{%
165 {%
166 \COOL@Eparser#1E\COOL@strStop\COOL@strEnd%
167 \ifthenelse{ \equal{\COOL@num@exponent}{\COOL@strStop} }{%
168 {%
169 \gdef\COOL@num@exponent{0}%

```

```

170 }%
171 % Else
172 {%
173 \expandafter\COOL@Ecorrector\COOL@num@exponent%
174 }%
175 }
176 % Else
177 {%
178 \expandafter\COOL@ecorrector\COOL@num@exponent%
179 }%
180 \isdecimal{\COOL@num@magnitude}{\COOL@numeric}%
181 \ifthenelse{ \boolean{\COOL@numeric} }%
182 {%
183 \isdecimal{\COOL@num@exponent}{\COOL@numeric}%
184 \ifthenelse{ \boolean{\COOL@numeric} }%
185 {%
186 \setboolean{#2}{true}%
187 }%
188 % Else
189 {%
190 \setboolean{#2}{false}%
191 }%
192 }%
193 % Else
194 {%
195 \setboolean{#2}{false}%
196 }%
197 }

```

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

```

198 \newboolean{COOL@isint}
199 \def\COOL@intgobbler#1#2\COOL@strEnd{%
200 \ifcat#1%
201 \ifthenelse{\equal{#2}{\COOL@strStop}}%
202 {%
203 \ifthenelse{‘#1 < ‘0 \OR ‘#1 > ‘9}%
204 {%
205 \setboolean{COOL@isint}{false}%
206 }%
207 % Else
208 {%
209 }%
210 }%
211 % Else
212 {%
213 \ifthenelse{ ‘#1 < ‘0 \OR ‘#1 > ‘9 }%
214 {%
215 \ifthenelse{ ‘#1 = ‘+ \OR ‘#1 = ‘- \AND \value{COOL@strpointer} = 1 }%
216 }%
217 % Else
218 {%
219 \setboolean{COOL@isint}{false}%
220 }%

```

```

221 }%
222 % Else
223 {%
224 }%
225 \stepcounter{COOL@strpointer}%
226 \COOL@intgobbler#2\COOL@strEnd%
227 }%
228 \else%
229 \setboolean{COOL@isint}{false}%
230 \fi%
231 }

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

232 \newcommand{\isint}[2]{%
233 \setcounter{COOL@strpointer}{1}%
234 \setboolean{COOL@isint}{true}%
235 \COOL@intgobbler#1\COOL@strStop\COOL@strEnd%
236 \ifthenelse{ \boolean{COOL@isint} }{%
237 {%
238 \setboolean{#2}{true}%
239 }%
240 % Else
241 {%
242 \setboolean{#2}{false}%
243 }%
244 }

```

Change History

v1.0		
General: Initial Release 1	
v2.0		
General: Added three new commands: <code>ifstrchareq</code> , <code>ifstrleneq</code> , <code>strlen</code> 1	
<code>\COOL@decimalgobbler</code> : added this “gobbler” to complete <code>isnumeric</code> 5	
<code>\COOL@strcomparegobble</code> : added to package for single character comparisons 4	v2.0a
<code>\ifstrchareq</code> : added to package to do character comparing 4	
		<code>\ifstrleneq</code> : added to package to do length comparison 5
		<code>\isdecimal</code> : added 6
		<code>\isint</code> : added extra mandatory argument for storing return boolean 8
		<code>\isnumeric</code> : added extra mandatory argument for storing return boolean 6
		<code>\strlen</code> : added to package 3
		<code>\strlenstore</code> : added to package . 4
		<code>\isint</code> : modified internals slightly to work with <code>cool</code> package 8

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 8

C	
\COOL@decimalgobbler	92, 137
\COOL@Ecorrector	159, 173
\COOL@ecorrector	152, 178
\COOL@Eparser	155, 166
\COOL@eparser	148, 163
\COOL@intEnd	3
\COOL@intgobbler	199, 226, 235
\COOL@num@exponent	150, 153, 157, 160, 164, 167, 169, 173, 178, 183
\COOL@num@magnitude	149, 156, 180
\COOL@strcomparegobble	44, 71
\COOL@strEnd	2, 5, 8, 33, 45, 71, 94, 130, 137, 148, 155, 163, 166, 199, 226, 235
\COOL@strgobble	7, 28, 33
\COOL@strStop	4, 95, 137, 152, 159, 163, 164, 166, 167, 201, 235
I	
\ifstrchareq	68
\ifstrleneq	81
\isdecimal	133, 180, 183
\isint	232
\isnumeric	147
S	
\setstrEnd	2
\strchar	31, 36, 40, 82
\strlen	35
\strlenstore	39
X	
\xdef	149, 150, 153, 156, 157, 160