The \texttt{coolstr} package\footnote{This document corresponds to \texttt{cool} v2.2, dated 2009/09/09.}

\texttt{nsetzer}

September 10, 2009

The \texttt{coolstr} package is a “sub” package of the \texttt{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 \texttt{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

\texttt{\textbackslash substr}\{\texttt{\langle string\rangle}\}\{-\texttt{\langle start index\rangle}\}\{-\texttt{\langle num char\rangle}\}\} gives at most \texttt{\langle num char\rangle} characters from \texttt{\langle string\rangle}.

if \texttt{\langle start index\rangle} is greater than zero, and \texttt{\langle num char\rangle} is greater than zero, \texttt{\textbackslash substr} gives at most \texttt{\langle num char\rangle} starting with index \texttt{\langle start index\rangle} and going to the end of the string.

if \texttt{\langle start index\rangle} is greater than zero, and \texttt{\langle num char\rangle} is less than zero, \texttt{\textbackslash substr} gives at most \texttt{\langle num char\rangle} characters and going to the beginning of the string.

if \texttt{\langle start index\rangle} is less than zero, and \texttt{\langle num char\rangle} is greater than zero, \texttt{\textbackslash substr} gives at most \texttt{\langle num char\rangle} characters starting at the \texttt{\langle start index\rangle} character from the end of the string and going to the end of the string.

if \texttt{\langle start index\rangle} is less than zero, and \texttt{\langle num char\rangle} is less than zero, \texttt{\textbackslash substr} gives at most \texttt{\langle num char\rangle} characters starting at the \texttt{\langle start index\rangle} character from the end of the string and going to the beginning of the string.

There are two special, non-numeric values that \texttt{\langle char num\rangle} may take. They are \texttt{end} or \texttt{beg}, and they will always go to the end or beginning of the string, respectively.
3 Test Cases

3.1 \( \text{\textbackslash{substr}} \)

| \text{\textbackslash{substr}}{12345}\{1\}{2} | 12 |
| \text{\textbackslash{substr}}{12345}\{3\}{5} | 345 |
| \text{\textbackslash{substr}}{12345}\{3\}\{\text{\textend}\} | 345 |
| \text{\textbackslash{substr}}{12345}\{3\}\{\text{\textbegin}\} | 123 |
| \text{\textbackslash{substr}}{12345}\{-2\}\{1\} | 4 |
| \text{\textbackslash{substr}}{12345}\{3\}\{-2\} | 23 |
| \text{\textbackslash{substr}}{12345}\{-2\}\{-2\} | 34 |
| \text{\textbackslash{substr}}{12345}\{0\}\{5\} | (the null string) |
| \text{\textbackslash{substr}}{12345}\{2\}\{0\} | (the null string) |

3.2 \( \text{\textbackslash{isdecimal}} \)

| (null str) | not a decimal |
| _ | not a decimal |
| ~ | not a decimal |
| 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 |
| \text{\newcommand}{\text{\textbackslash{numberstore}}}{4.5} | |
| \text{\textbackslash{numberstore}} | is decimal |
3.3 \texttt{\textbackslash isnumeric}

<table>
<thead>
<tr>
<th></th>
<th>not numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>(null str)</td>
<td>not numeric</td>
</tr>
<tr>
<td>_</td>
<td>not numeric</td>
</tr>
<tr>
<td>__</td>
<td>not numeric</td>
</tr>
<tr>
<td>4.5</td>
<td>is numeric</td>
</tr>
<tr>
<td>4.5e5</td>
<td>is numeric</td>
</tr>
<tr>
<td>+4.5e5</td>
<td>is numeric</td>
</tr>
<tr>
<td>4.5e+5</td>
<td>is numeric</td>
</tr>
<tr>
<td>+4.5e+5</td>
<td>is numeric</td>
</tr>
<tr>
<td>4.5E5</td>
<td>is numeric</td>
</tr>
<tr>
<td>-4.5E5</td>
<td>is numeric</td>
</tr>
<tr>
<td>4.5E-5</td>
<td>is numeric</td>
</tr>
<tr>
<td>-4.5E-5</td>
<td>is numeric</td>
</tr>
<tr>
<td>4.5.E-5</td>
<td>not numeric</td>
</tr>
<tr>
<td>abcdefg</td>
<td>not numeric</td>
</tr>
<tr>
<td>abcE-5</td>
<td>not numeric</td>
</tr>
</tbody>
</table>

3.4 \texttt{\textbackslash isint}

<table>
<thead>
<tr>
<th></th>
<th>not integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(null str)</td>
<td>not integer</td>
</tr>
<tr>
<td>_</td>
<td>not integer</td>
</tr>
<tr>
<td>__</td>
<td>not integer</td>
</tr>
<tr>
<td>4</td>
<td>is integer</td>
</tr>
<tr>
<td>+4</td>
<td>is integer</td>
</tr>
<tr>
<td>4.5</td>
<td>not integer</td>
</tr>
<tr>
<td>4.5e5</td>
<td>not integer</td>
</tr>
<tr>
<td>+4.5e5</td>
<td>not integer</td>
</tr>
<tr>
<td>4.5e+5</td>
<td>not integer</td>
</tr>
<tr>
<td>+4.5e+5</td>
<td>not integer</td>
</tr>
<tr>
<td>4.5E5</td>
<td>not integer</td>
</tr>
<tr>
<td>-4.5E5</td>
<td>not integer</td>
</tr>
<tr>
<td>4.5E-5</td>
<td>not integer</td>
</tr>
<tr>
<td>-4.5E-5</td>
<td>not integer</td>
</tr>
<tr>
<td>4.5.E-5</td>
<td>not integer</td>
</tr>
<tr>
<td>abcdefg</td>
<td>not integer</td>
</tr>
<tr>
<td>abcE-5</td>
<td>not integer</td>
</tr>
<tr>
<td>\renewcommand{\numberstore}{4}\numberstore</td>
<td>is integer</td>
</tr>
</tbody>
</table>

4 Acknowledgments

Thanks to J. J. Weimer for the comments and aid in coding. Thanks goes to Abraham Weishaus for pointing out a bug in \texttt{\textbackslash strlenstore}. Thanks to Daniel Kucerovsky for pointing the ‘blank-space’ bug of \texttt{\textbackslash isnumeric} (and consequently \texttt{\textbackslash isdecimal}).
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}{\%0\%0\%0}
4 \let\COOL@strStop=\relax

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

5 \renewcommand{\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 \ifthenelse{\value{COOL@strpointer}=#1}{{
10 \ifthenelse{\value{COOL@strpointer}=#1}{{
11 }%}
12 }%}
13 }%}
14 }%
\texttt{\textbackslash strchar} \texttt{\strchar{\langle index\rangle}} \textbf{gives the} \textit{\langle index\rangle} \textbf{character of the string. Strings start indexing at 1.}

\texttt{\textbackslash strlen} \texttt{\strlen{\langle string\rangle}} \textbf{gives the length of the string. It is better to use \texttt{\strlenstore} to record the length} \texttt{\strlen{abc} 3}
\strlenstore \strlenstore{\langle string\rangle}{\langle counter\rangle} stores the length of \langle string\rangle in \langle counter\rangle

\newcommand{\strlenstore}[2]{% 
  \ifthenelse{\equal{#1}{}}% 
  {% 
    \setcounter{#2}{0}% 
  }% 
  % Else 
  {% 
    \strchar{#1}{0}% 
    \setcounter{#2}{\value{COOL@strpointer}}% 
  }% 
}%

\substr \substr{\langle string\rangle}{\langle index\rangle}{\langle numchar\rangle} 
  a special value of end for \langle numchar\rangle gives from \langle index\rangle to the end of the string; beg gives from \langle index\rangle to the beginning of the string
\newcounter{COOL@str@index} 
\newcounter{COOL@str@start} 
\newcounter{COOL@str@end} 
\newcommand{\substr}[3]{% 
  \strlenstore{#1}{COOL@strlen}% 
  \ifthenelse{#2 < 0 \AND \NOT #2 < -\value{COOL@strlen}}% 
  {% 
    \setcounter{COOL@str@index}{\value{COOL@strlen}}% 
    \addtocounter{COOL@str@index}{#2}% 
  }% 
  % The starting index is less than zero, so start that many characters back from the end. This means mapping the index to \langle index\rangle + \langle string length\rangle + 1 
  \setcounter{COOL@str@index}{\value{COOL@strlen}}% 
  \addtocounter{COOL@str@index}{#2}% 
}
\addtocounter{COOL@str@index}{1}\%
\%
\% ElseIf
{\ifthenelse{#2 > 0 \AND \NOT #2 > \value{COOL@strlen}}%
 {%
The starting index is greater than zero, and within the appropriate range; record it
\setcounter{COOL@str@index}{#2}%
}%
% Else
{% % \end{macrocode}%
% The \meta{index} value is invalid. Set it to zero for returning the null string% \begin{macrocode}%
\setcounter{COOL@str@index}{0}%
\}%
}
\begin{macrocode}
Now deal with the \langle numchar \rangle (which can also be negative)
{\ifthenelse{\equal{#3}{beg}}%
 {% \setcounter{COOL@str@start}{1}\%
 \setcounter{COOL@str@end}{\value{COOL@str@index}}\%
}%
% Elself
{\ifthenelse{\equal{#3}{end}}%
 {% \setcounter{COOL@str@end}{\value{COOL@strlen}}%
}%
% Elself
{\ifthenelse{#3 < 0}%
 {% \setcounter{COOL@str@start}{\value{COOL@str@index}}\%
 \setcounter{COOL@str@end}{\value{COOL@strlen}}\%
}%
This means to take that many characters to the left of the starting index.

\( \text{setcounter{COOL@str@start}}{\text{value{COOL@str@index}}} \)
\( \text{addtocounter{COOL@str@start}}{#3} \)
\( \text{addtocounter{COOL@str@start}}{1} \)
\( \text{ifthenelse{NOT value{COOL@str@start} > 0}{setcounter{COOL@str@start}{1}}{}} \)
\( \text{setcounter{COOL@str@end}}{\text{value{COOL@str@index}}} \)
\( \text{setcounter{COOL@str@start}}{1} \)
\( \text{ifthenelse{#3 > 0}{%} \}
\( \text{setcounter{COOL@str@start}}{\text{value{COOL@str@index}}} \)
\( \text{setcounter{COOL@str@end}}{\text{value{COOL@str@index}}} \)
\( \text{addtocounter{COOL@str@end}}{#3} \)
\( \text{addtocounter{COOL@str@end}}{-1} \)
\( \text{ifthenelse{value{COOL@str@end} > value{COOL@strlen}}{setcounter{COOL@str@end}{value{COOL@strlen}}}{} \}
\)
\( \text{setcounter{COOL@str@index}}{0} \)
\( \text{setcounter{COOL@str@index}}{0} \)
\( \text{setcounter{COOL@str@index}}{0} \)

Now send back the appropriate thing
\( \text{ifthenelse{value{COOL@str@index} = 0}{%} \}
\( \text{setcounter{COOL@strpointer}}{1} \)
\( \text{COOL@substrgobbler#1\textbackslash COOL@strStop\textbackslash COOL@strEnd} \)
\( \text{}} \)
Now define the “gobbler”
\def\COOL@substrgobbler#1#2\COOL@strEnd{%
  \ifthenelse{\equal{#2}{\COOL@strStop}}%
    {%
      \ifthenelse{ \value{COOL@strpointer} < \value{COOL@str@start} \OR \value{COOL@strpointer} > \value{COOL@str@end} }%
        {}%
      % Else
      #1%
    %}%
  % Else
  {%
    \stepcounter{COOL@strpointer}%
    \COOL@substrgobbler#2\COOL@strEnd%
  %}
%
\def\COOL@strcomparegobble[#1]<#2>#3#4{%
  \ifthenelse{\equal{#4}{\COOL@strEnd}}%
    {%}
  % Else
  {%
    \ifthenelse{ \value{COOL@strpointer} < \value{COOL@str@start} \OR \value{COOL@strpointer} > \value{COOL@str@end} }%
      {}%
    % Else
    #1%
  %}
  %}
%
\newboolean{COOL@charmatch}
\COOL@strcomparegobble
This “gobbler” does character comparison
\def\COOL@strcomparegobble[#1]<#2>#3#4{%
  \ifthenelse{\equal{#4}{\COOL@strEnd}}%
    {%}
\ifstrleneq{string}{number}{do if true}{do if false}
\ifstrleneq{abc}{3}{length is $3$}{length is not $3$} length is 3
\ifstrleneq{abcde}{3}{length is $3$}{length is not $3$} length is not 3

\newcommand{\ifstrleneq}{
\strlenstore{#1}{COOL@strlen}\%
\ifthenelse{ \value{COOL@strlen} = #2 }{% 
    #3\%
}% Else
    #4\%
}\)

\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.

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

\newcommand{\COOL@decimalgobbler}{#1#2\COOL@strEnd}{%
\ifthenelse{\equal{#1}{\COOL@strStop}}{% 
\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
\def{\COOL@decimalgobbler}{% 
\ifthenelse{\equal{#1}{\COOL@strStop}}{% 
}{% 
}
user submitted a null string, which can not be numeric

\setboolean{COOL@decimal}{false}%
\{\ifthenelse{\equal{\texttt{#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

\{\%  
\ifthenelse{\texttt{#1} < '0 \text{ OR } \texttt{#1} > '9} %
  {\%  
   \ifthenelse{ \texttt{#1} = '.' \text{ AND} \NOT \value{COOL@strpointer} = 1 \text{ AND} \NOT \boolean{COOL@decimalfound}} %
    {\%  
     \setboolean{COOL@decimal}{false} %
    }%  
   }%  
  }%  
\% \Else 
\{\%  
   \setboolean{COOL@decimal}{false} %
\}%  
\}%  \Else  
\{\%
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

\ifthenelse{ \texttt{#1} = '.' \text{ AND} \NOT \boolean{COOL@decimalfound} } %
  {\%  
   \setboolean{COOL@decimal}{true} %
\}%
\isdecimal\{⟨string⟩\}\{⟨boolean⟩\}\\

\newcommand{\isdecimal}[2]{\\
\setcounter{COOL@strpointer}{0}\\
\setboolean{COOL@decimalfound}{false}\\
\setboolean{COOL@decimal}{true}\\
\expandafter\COOL@decimalgobbler#1\COOL@strStop\COOL@strEnd%\\
\ifthenelse{ \boolean{COOL@decimal} }{\setboolean{#2}{true}}{\setboolean{#2}{false}}\\
}
\isnumeric \isnumeric{\texttt{string}}\{\texttt{boolean}\} stores \texttt{true} in \texttt{boolean} if \texttt{string} is numeric

\begin{verbatim}
\newboolean{\COOL@numeric}%
\def\COOL@eparser#1e#2\COOL@strEnd{%
  \xdef\COOL@num@magnitude{#1}%
  \xdef\COOL@num@exponent{#2}%
}
\def\COOL@ecorrector#1e\COOL@strStop{%
  \xdef\COOL@num@exponent{#1}%
}
\def\COOL@Eparser#1E#2\COOL@strEnd{%
  \xdef\COOL@num@magnitude{#1}%
  \xdef\COOL@num@exponent{#2}%
}
\def\COOL@Ecorrector#1E\COOL@strStop{%
  \xdef\COOL@num@exponent{#1}%
}
\newcommand{\isnumeric}[2]{%
  \COOL@eparser#1e\COOL@strStop\COOL@strEnd%
  \ifthenelse{\equal{\COOL@num@exponent}{\COOL@strStop}}%
    {%
      \COOL@Eparser#1E\COOL@strStop\COOL@strEnd%
      \ifthenelse{\equal{\COOL@num@exponent}{\COOL@strStop}}%
        {%
          \gdef\COOL@num@exponent{0}%
        }%
    }%
  \else%
    \expandafter\COOL@Ecorrector\COOL@num@exponent%
  \fi%
}
\end{verbatim}
In addition to identifying numeric data, it is useful to know if integers are present, thus another “gobbler” is needed.
\isint \isint{\textit{string}}{\textit{boolean}} sets the \textit{boolean} to \texttt{true} if \textit{string} is an integer or \texttt{false} otherwise.
\setboolean{#2}{true} %
}\%
% Else
{%
\setboolean{#2}{false} %
}%
}%
Change History

v1.0
General: Initial Release ........ 1

v2.0
General: Added three new commands: ifstrchareq, ifstrleneq, strlen .......... 1
\COOL@decimalgobbler: added this “gobbler” to complete isnumeric ............... 11
\COOL@strcomparegobbler: added to package for single character comparisons .......... 9
ifstrchareq: added to package to do character comparing ....... 10
ifstrleneq: added to package to do length comparison ....... 11
isdecimal: added .......... 13
isint: added extra mandatory argument for storing return boolean ............. 16
isnumeric: added extra mandatory argument for storing return boolean ............. 14

\strlen: added to package ...... 5
\strlenstore: added to package ... 6

v2.0a
isint: modified internals slightly to work with cool package ...... 16

v2.1
ifstrleneq: altered function to use strlenstore ............... 11
\strlen: added ifthenelse to return 0 for empty string ............... 5
strlenstore: added ifthenelse to return 0 for empty string ...... 6
corrected error in setting counter 6
\substr: added to package ...... 6

v2.1b
isint: added expandafter before COOL@intgobbler to expand macros before evaluating ...... 16

v2.2
\COOL@decimalgobbler: fixed blank space bug (blank space causes code to ‘crash’) ...... 11

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 ........................................ 193, 242
\COOL@decimalgobbler ........... 264, 278
\COOL@Ecorrector ................. 257, 283
\COOL@Ecorrector ................. 260, 271
\COOL@Epsparser ................. 253, 268
\COOL@EpintEnd ................. 3
\COOL@Epgobbler ................. 304, 331, 340
\COOL@EnumExponent ............ 255, 258, 262, 265, 269, 272, 274, 278, 283, 288
\COOL@EnumMagnitude ........... 254, 261, 285
\COOL@Estrcomparegobbler ....... 145, 172

\COOL@End ......................... 2, 5, 8, 33, 119, 122, 141, 146, 172, 195, 235, 242,
\COOL@Estrgobble .................. 7, 28, 33
\COOL@EstrStop .................... 4, 119, 123, 196, 200, 242, 257, 264, 268, 269, 271, 272, 306, 340
\COOL@Esubstrgobbler ............ 119, 122, 141

I ...................................... 169
ifstrchareq ....................... 182
ifstrleneq ....................... 238, 285, 288
isdecimal ....................... 337
isint ....................... 252
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Assigned Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>\M</td>
<td>\strchar</td>
<td>31, 42, 53</td>
</tr>
<tr>
<td>\meta</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>\S</td>
<td>\strlen</td>
<td>35</td>
</tr>
<tr>
<td>\setstrEnd</td>
<td></td>
<td>46, 61, 183</td>
</tr>
<tr>
<td>\strlenstore</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>\substr</td>
<td></td>
<td>1, 2, 57</td>
</tr>
</tbody>
</table>