The coolstr package[*]

nsetzer

October 11, 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 begining of the string, respectively

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

3 Test Cases

3.1 \substr

\substr

`\substr{12345}{1}{2}`	12
`\substr{12345}{3}{5}`	345
`\substr{12345}{3}{end}`	345
`\substr{12345}{3}{beg}`	123
`\substr{12345}{-2}{1}`	4
`\substr{12345}{3}{-2}`	23
`\substr{12345}{-2}{-2}`	34
`\substr{12345}{0}{5}`	(the null string)
`\substr{12345}{2}{0}`	(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
	`\newcommand{\numberstore}{4.5}`
`\numberstore`	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

```
\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

```
\newcommand{\COOL@strEnd}{\%\%\%}
\newcommand{\COOL@intEnd}{\%@\%@\%@}
\let\COOL@strStop=\relax
```

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

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

This area defines the core technology behind the coolstr package: the string "gobbler".

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

Now we come to "the gobbler"—a recursive function that eats up a string. It must be written in TEX primatives.

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

```
\def\COOL@strgobble[#1]#2#3{%
\ifthenelse{\equal{#3}{\COOL@strEnd}}%
  {%
  \ifthenelse{\value{COOL@strpointer}=#1}%
    {%
    #2%
    }%
  % Else
    {%
    }%
  }%
% Else
  {%
  \ifthenelse{\value{COOL@strpointer}=#1}%
    {%
    #2%
    }%
  % Else
    {%
    }%
  \stepcounter{COOL@strpointer}%
  \COOL@strgobble[#1]#3%
  }%
}
```

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

```
\newcommand{\strchar}[2]{%
\setcounter{COOL@strpointer}{1}%
\COOL@strgobble[#2]#1\COOL@strEnd%
}
```

`\strlen` `\strlen{`⟨*string*⟩`}` gives the length of the string. It is better to use `\strlenstore` to record the length
`\strlen{abc}` 3

```
\newcommand{\strlen}[1]{%
\ifthenelse{\equal{#1}{}}%
  {%
  0%
  }%
% Else
  {%
  \strchar{#1}{0}%
  \arabic{COOL@strpointer}%
  }%
}
```

`\strlenstore`

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

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

`\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

```
\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}}%
  {%
```

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*⟩ $+ 1$

```
  \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{macroccode}
% The \meta{index} value is invalid.  Set it to zero for returning the null string
%    \begin{macrocode}
  \setcounter{COOL@str@index}{0}%
  }}%
```

Now deal with the ⟨*numchar*⟩ (which can also be negative)

```
\ifthenelse{\equal{#3}{beg}}%
  {%
  \setcounter{COOL@str@start}{1}%
  \setcounter{COOL@str@end}{\value{COOL@str@index}}%
  }%
% ElseIf
{\ifthenelse{\equal{#3}{end}}%
  {%
  \setcounter{COOL@str@start}{\value{COOL@str@index}}%
  \setcounter{COOL@str@end}{\value{COOL@strlen}}%
  }%
% ElseIf
{\ifthenelse{#3 < 0}%
  {%
```

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

```
  \setcounter{COOL@str@start}{\value{COOL@str@index}}%
  \addtocounter{COOL@str@start}{#3}%
  \addtocounter{COOL@str@start}{1}%
  \ifthenelse{\NOT \value{COOL@str@start} > 0}{\setcounter{COOL@str@start}{1}}{}%
  \setcounter{COOL@str@end}{\value{COOL@str@index}}%
  }%
% ElseIf
{\ifthenelse{#3 > 0}%
  {%
  \setcounter{COOL@str@start}{\value{COOL@str@index}}%
  \setcounter{COOL@str@end}{\value{COOL@str@index}}%
  \addtocounter{COOL@str@end}{#3}%
  \addtocounter{COOL@str@end}{-1}%
  \ifthenelse{\value{COOL@str@end} > \value{COOL@strlen}}{\setcounter{COOL@str@end}{\value{COOL@strlen}}}{}%
  }%
% Else
  {%
```

nonsense submitted, so return the null string

```
  \setcounter{COOL@str@index}{0}%
  }}}}%
```

Now send back the appropriate thing

```
\ifthenelse{ \value{COOL@str@index} = 0 }%
  {%
  }%
% Else
  {%
  \setcounter{COOL@strpointer}{1}%
  \COOL@substrgobbler#1\COOL@strStop\COOL@strEnd%
  }%
}
```

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
  {%
  \ifthenelse{ \value{COOL@strpointer} < \value{COOL@str@start} \OR \value{COOL@strpointer} > \value{COOL@str@end} }%
    {}%
  % Else
    {%
    #1%
    }%
  \stepcounter{COOL@strpointer}%
  \COOL@substrgobbler#2\COOL@strEnd%
  }%
}
```

Define a new boolean for comparing characters

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

`\COOL@strcomparegobble` This "gobbler" does character comparison

```
\def\COOL@strcomparegobble[#1]<#2>#3#4{%
\ifthenelse{\equal{#4}{\COOL@strEnd}}%
  {%
  \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }%
    {%
    \setboolean{COOL@charmatch}{true}%
    }%
  % Else
    {%
    }%
  }%
% Else
  {%
  \ifthenelse{\value{COOL@strpointer}=#1 \AND \equal{#2}{#3} }%
    {%
    \setboolean{COOL@charmatch}{true}%
    }%
  % Else
    {%
    }%
  \stepcounter{COOL@strpointer}%
  \COOL@strcomparegobble[#1]<#2>#4%
  }%
}
```

\ifstrchareq `\ifstrchareq{`⟨*string*⟩`}{`⟨*char index*⟩`}{`⟨*comparison char*⟩`}{`⟨*do if true*⟩`}{`⟨*do if false*⟩`}`

```
\newcommand{\ifstrchareq}[5]{%
\setboolean{COOL@charmatch}{false}%
\setcounter{COOL@strpointer}{1}%
\COOL@strcomparegobble[#2]<#3>#1\COOL@strEnd\relax%
\ifthenelse{ \boolean{COOL@charmatch} }%
  {%
  #4%
  }%
% Else
```

```
 {%
 #5%
 }%
}
```

\ifstrleneq

`\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}[4]{%
\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}
```

`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#1#2\COOL@strEnd{%
\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

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

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{ ‘#1 = ‘.  \AND \NOT \boolean{COOL@decimalfound} }%
   {%
   \setboolean{COOL@decimalfound}{true}%
   }%
  {\ifthenelse{ \(‘#1 = ‘+ \OR ‘#1 = ‘-\) \AND \value{COOL@strpointer} = 1 }%
   {%
   }%
  % Else
   {%
   \setboolean{COOL@decimal}{false}%
```

```
      }}%
    }%
  % Else
    {}%
  \stepcounter{COOL@strpointer}%
  \COOL@decimalgobbler#2\COOL@strEnd%
  }%
}
```

`\isdecimal` isdecimal{⟨*string*⟩}{⟨*boolean*⟩}

```
\newcommand{\isdecimal}[2]{%
\setcounter{COOL@strpointer}{1}%
\setboolean{COOL@decimalfound}{false}%
\setboolean{COOL@decimal}{true}%
\expandafter\COOL@decimalgobbler#1\COOL@strStop\COOL@strEnd%
\ifthenelse{ \boolean{COOL@decimal} }%
  {%
  \setboolean{#2}{true}%
  }%
% Else
  {%
  \setboolean{#2}{false}%
  }%
}%
```

`\isnumeric` \isnumeric{⟨*string*⟩}{⟨*boolean*⟩} stores `true` in ⟨*boolean*⟩ if ⟨*string*⟩ is numeric

```
\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%
    }%
  }
% Else
  {%
  \expandafter\COOL@ecorrector\COOL@num@exponent%
  }%
\isdecimal{\COOL@num@magnitude}{COOL@numeric}%
\ifthenelse{ \boolean{COOL@numeric} }%
  {%
  \isdecimal{\COOL@num@exponent}{COOL@numeric}%
  \ifthenelse{ \boolean{COOL@numeric} }%
  {%
  \setboolean{#2}{true}%
  }%
 % Else
  {%
  \setboolean{#2}{false}%
  }%
 }%
% Else
 {%
 \setboolean{#2}{false}%
 }%
}
```

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

```
\newboolean{COOL@isint}
\def\COOL@intgobbler#1#2\COOL@strEnd{%
\ifcat#11%
\ifthenelse{\equal{#2}{\COOL@strStop}}%
 {%
 \ifthenelse{`#1 < `0 \OR `#1 > `9}%
  {%
  \setboolean{COOL@isint}{false}%
  }%
 % Else
  {%
  }%
 }%
% Else
 {%
 \ifthenelse{ `#1 < `0 \OR `#1 > `9 }%
  {%
  \ifthenelse{ `#1 = `+ \OR `#1 = `- \AND \value{COOL@strpointer} = 1 }%
   {}%
  % Else
   {%
   \setboolean{COOL@isint}{false}%
   }%
  }%
 % Else
  {%
  }%
 \stepcounter{COOL@strpointer}%
 \COOL@intgobbler#2\COOL@strEnd%
 }%
\else%
 \setboolean{COOL@isint}{false}%
\fi%
}
```

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

```
\newcommand{\isint}[2]{%
\setcounter{COOL@strpointer}{1}%
\setboolean{COOL@isint}{true}%
\expandafter\COOL@intgobbler#1\COOL@strStop\COOL@strEnd%
\ifthenelse{ \boolean{COOL@isint} }%
 {%
 \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@strcomparegobble`: 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 13

`\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

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.