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
	\newcommand{\numberstore}{4.5}
\numberstore	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
	`\renewcommand{\numberstore}{4}`
`\numberstore`	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

```
\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]{%
\strchar{#1}{0}
\arabic{COOL@strpointer}%
}
```

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

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

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]{%
\strchar{#1}{0}%
\ifthenelse{ \value{COOL@strpointer} = #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}%
\COOL@intgobbler#1\COOL@strStop\COOL@strEnd%
\ifthenelse{ \boolean{COOL@isint} }%
{%
\setboolean{#2}{true}%
}%
% Else
{%
\setboolean{#2}{false}%
}%
}
```

Change History

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