Sorted Lists (`sortlist.dtx`)

Charles Duan

May 2, 2023

This package provides for insertion sorting of lists.

```
\ProvidesFile{sortlist}[2005/02/12 Sorted list]
\RequirePackage{etoolbox,strings}
```

1 Data Structures

A list is defined by the following macros:

- `\sli@@`⟨*name*⟩: the list data itself
- `\sli@fn@`⟨*name*⟩: the comparator function for the list
- `\sli@k@`⟨*name*⟩: the key function for the list
- `\sli@inc@`⟨*name*⟩: the inclusion function for the list
- `\sli@uniq@`⟨*name*⟩`@`⟨*elt*⟩: each element in the list, for uniqueness testing

The list data is a series of groups, one for each element of the list. Each group contains two elements: (1) the list item itself, and (2) the key computed for that item. Each group is preceded by the macro `\sli@do`:

```
\sli@do{{item}{key}}\sli@do{{item}{key}}...
```

(The `\sli@do` is necessary to ensure that the elements don't get unbraced as they are traversed.)

2 Creating and Using Lists

`\NewSortedList` `\NewSortedList` creates a new list. It requires three arguments: a name for the list, a sorting function, and a function definition that determines what elements to sort:

```
\NewSortedList{list}{\SortAlpha}{\def#2{#1}}
```

which would create a list named “list” which is sorted alphabetically and uses the elements of the list as sorting keys.

The sort function should be a function that takes two parameters and sets either `\CompReversetrue` or `\CompReversefalse` depending on whether the order of the items should be reversed.

The key function should take two parameters, an item to be sorted and a macro name, and it should define the macro to be the desired key.

```
\def\NewSortedList#1#2{%
    \@namedef{sli@@#1}{}%
    \@namedef{sli@fn@#1}{#2}%
    \@namedef{sli@inc@#1}{\@secondoftwo}%
    \expandafter\def\csname sli@k@#1\endcsname##1##2%
}
```

Conditional for comparator macro.

```
\newif\ifCompReverse
```

2.1 Testing Lists

Determines if the list exists.

```
\def\IfKnownList#1{\@ifundefined{sli@fn@#1}\@secondoftwo\@firstoftwo}%
```

Tests whether list #1 contains an element #2. If so, runs #3; else runs #4. (#3 and #4 are absorbed later.)

```
\def\IfListContains#1#2{%
    \@ifundefined{sli@uniq@#1@\detokenize{#2}}{\@secondoftwo}{\@firstoftwo}%
}
```

Determines if the list is empty.

```
\def\IfEmptyList#1{%
    \@test \expandafter\ifx\csname sli@@#1\endcsname\@empty \fi
}
```

2.2 List Inclusion Conditions

`\ListInclusionMacro`

`\ListElementsMustBeUnique`

A list can have a condition for testing whether items ought to be included in it. That is set with `\ListInclusionMacro{`⟨*list*⟩`}`. A common test is that list elements must be unique; that is done with `\ListElementsMustBeUnique{`⟨*list*⟩`}`.

Sets the inclusion test for a list. #1 is the list name, and #2 (absorbed later) should be a definition for a two-argument macro. The macro will receive (1) an element to be added to the list and (2) a callback that should be run only if the element should be added.

```
\def\ListInclusionMacro#1{%
    \@namedef{sli@inc@#1}##1##2%
}
```

Requires all elements of the list to be unique. #1 is the list name.

```
\def\ListElementsMustBeUnique#1{%
    \ListInclusionMacro{#1}{%
        \IfListContains{#1}{##1}{}{##2}%
    }%
}
```

2.3 Adding to Lists

`\AddToList`

`\AddToList{`⟨*list*⟩`}`element adds an element to a list.

```
\def\AddToList#1#2{%
    \@nameuse{sli@inc@#1}{#2}{%
        \cslet{sli@uniq@#1@\detokenize{#2}}\@empty
        \def\sli@list{#1}%
        \def\sli@elt{#2}%
        \csname sli@k@#1\endcsname{#2}\sli@key
        \IfEmptyList{#1}{%
```

```
            \sli@insert{}{}%
        }{%
            \@expand{\@unbrace{\let\sli@sfun}}{\csname sli@fn@#1\endcsname}i%
            \@expand{\sli@add}{\csname sli@@#1\endcsname}{ii}%
        }%
    }%
}
```

Actually inserts the current item into the list. #1 is the matter to go before the item; #2 is what goes after. (The parameters are absorbed by the helper.)

```
\def\sli@insert{%
    \@expand{\expandafter\sli@insert@\expandafter{\sli@elt}}{\sli@key}i%
}
\def\sli@insert@#1#2#3#4{%
    \@namedef{sli@@\sli@list}{#3\sli@do{{#1}{#2}}#4}%
}
```

Sets up the insertion run where there is at least one item already in the list.

```
\def\sli@add#1{%
    \sli@traverse\sli@mark#1\sli@stop
}
\def\sli@traverse#1\sli@mark\sli@do#2#3\sli@stop{%
    \@twodef\sli@cur@elt\sli@cur@key#2%
    \@expandtwo\sli@sfun\sli@key{\@secondoftwo#2}%
    \@test \ifCompReverse \fi{%
        \ifstrempty{#3}{%
            \sli@insert{#1\sli@do{#2}}{}%
        }{\sli@traverse#1\sli@do{#2}\sli@mark#3\sli@stop}%
    }{%
        \sli@insert{#1}{\sli@do{#2}#3}%
    }%
}
```

2.4 Using Lists

\ShowList

The `\ShowList{`⟨*list*⟩`}{`⟨*callback*⟩`}` macro executes ⟨*callback*⟩ on every item of the list, in order. ⟨*callback*⟩ should be in the form of a macro definition that accepts one argument, which will be set to the list item.

```
\def\ShowList#1{%
    \def\sli@list{\csname sli@@#1\endcsname}%
    \def\sli@do##1{\expandafter\sli@act\expandafter{\@firstoftwo##1}}%
    \afterassignment\sli@list\def\sli@act##1%
}
```

3 Sorting Functions

As explained above, sorted lists require a sorting function that takes two parameters and sets `\ifCompReverse` if the elements are in the wrong sorting order.

The following macros provide some standard sorting functions. Each will take two additional arguments besides the ones described, which will be the list elements for testing.

\ReverseSort

`\ReverseSort{`⟨*test*⟩`}` performs ⟨*test*⟩ on the list elements and gives the reverse result.

```
\def\ReverseSort#1#2#3{%
    #1{#2}{#3}\ifCompReverse\CompReversefalse\else\CompReversetrue\fi
}
```

\SortNum

`\SortNum` sorts elements numerically; all elements must be numbers.

```
\def\SortNum#1#2{%
    \@tempcnta#1\relax \@tempcntb#2\relax
    \ifnum\@tempcnta>\@tempcntb \CompReversetrue \else \CompReversefalse \fi
}
```

\NoSort

`\NoSort` performs no sorting, meaning that elements are listed in reverse order of addition. (Consider using `\ReverseSort` to get elements in original order of addition.)

```
\def\NoSort#1#2{\CompReversefalse}
```

\SortLen

`\SortLen` sorts elements by their length, counted by number of tokens.

```
\def\SortLen#1#2{\sli@sortlen#1\@nil#2\@nil}
\def\sli@sortlen#1#2\@nil#3#4\@nil{%
    \ifstrempty{#2}\CompReversefalse{%
        \ifstrempty{#4}\CompReversetrue{%
            \sli@sortlen#2\@nil#4\@nil
        }%
    }%
}
\def\@ifbothonechar#1#2{\@ifonechar{#1}{\@ifonechar{#2}}{\@secondoftwo}}
```

\SortEasyAlpha

`\SortEasyAlpha` sorts simple text elements alphabetically. Its input ought to have been converted by `\StripForAlpha` first.

```
\def\SortEasyAlpha#1#2{\sli@ea#1\@nil#2\@nil}
\def\sli@ea#1#2\@nil#3#4\@nil{%
    \@test \ifnum`#1=`#3\relax \fi{%
        \ifstrempty{#2}{\CompReversefalse}{%
            \ifstrempty{#4}{\CompReversetrue}{%
                \sli@ea#2\@nil#4\@nil
            }%
        }%
    }{%
        \ifnum`#1>`#3\relax \CompReversetrue \else \CompReversefalse \fi
    }%
}
```

\StripForAlpha

`\StripForAlpha{`⟨*text*⟩`}{`⟨*callback*⟩`}` is a helpful function to precede alphabetical sorting. It takes ⟨*text*⟩, uppercases it, and removes all non-alphabetical characters from it. It also pads numbers of fewer than four digits, so that "Volume 3" is sorted before "Volume 14".

```
%
\def\StripForAlpha#1#2{%
    \def\sli@strip{#1}%
    \edef\sli@strip{%
        \expandafter\sli@strip@angle\meaning\sli@strip. \relax
    }%
    \@expand{#2}\sli@strip i%
}
\def\sli@strip@angle#1>{\sli@strip@spaces}
\def\sli@strip@spaces#1 #2\relax{%
    \ifstrempty{#2}{\sli@strip@next#1.\relax}{%
        \sli@strip@spaces#1.#2\relax
    }%
}
\edef\reserved@a{\noexpand\let\noexpand\sli@bs\expandafter\@gobble\string\\}
\reserved@a
\def\sli@strip@next#1{%
    \@test \ifx#1\sli@bs \fi{\sli@strip@cs}{%
        \@test \ifx#1\relax \fi{}{%
            \@ifdigit{#1}{\sli@strip@num{#1}.....\sli@stop}{%
                \@test \ifnum\uccode`#1>\z@ \fi{#1}{}%
                \sli@strip@next
            }%
        }%
    }%
}
```

Numbers should be read fully and sorted numerically rather than digit-by-digit. The easiest way to do this is to pad numbers to a fixed width. But unnecessarily large numbers are generally identifiers rather than quantities. Accordingly, only numbers of four or fewer digits are padded.

```
\def\sli@strip@num#1.#2\sli@stop#3{%
    \@ifdigit{#3}{%
        \ifstrempty{#2}{%
            \sli@strip@num{#1#3}.\sli@stop
        }{%
            \sli@strip@num{#1#3}#2\sli@stop
        }%
    }{#2#1\sli@strip@next#3}%
}
```

```
\def\sli@strip@cs#1{%
    \@test \ifnum\catcode`#1=11 \fi{\sli@strip@csend}{%
        \@test \ifnum`#1=`@ \fi{\sli@strip@csend}{%
            \sli@strip@next
        }%
    }%
}
\def\sli@strip@csend#1.{\sli@strip@next}
```

\SortAlpha

`\SortAlpha` sorts elements alphabetically, after they have been stripped of non-alphabetical characters. (This macro is probably outdated in view of `\SortEasyAlpha` and `\StripForAlpha`.)

```
\def\SortAlpha#1#2{\sli@alpha#1\@nil#2\@nil}
\def\sli@alpha#1#2\@nil#3#4\@nil{%
    \@ifbothonechar{#1}{#3}{%
\sli@alpha@bothonechar#1{#2}#3{#4}%
    }{%
% Group found inside argument. Expand it.
\sli@alpha#1#2\@nil#3#4\@nil
    }%
}
\def\sli@alpha@bothonechar#1#2#3#4{%
    \ifcat\noexpand#3\relax
% #3 is a control sequence, but not \sli@stop
\expandafter\sli@alpha@chkthree
    \else
\expandafter\sli@alpha@onecs
    \fi
    % Args to whichever command is chosen
    #1{#2}#3{#4}%
}
\def\sli@alpha@onecs#1{%
    \ifcat\noexpand#1\relax
\expandafter\sli@alpha@chkone
    \else
\expandafter\sli@alpha@chars
    \fi
    #1% Put it back onto token list
}
\def\sli@alpha@chkone#1#2#3#4{%
    \@test\ifx#1\sli@stop\fi{%
\CompReversefalse
    }{%
\sli@alpha#2\@nil#3#4\@nil
    }%
}
\def\sli@alpha@chkthree#1#2#3#4{%
    \@test\ifx#3\sli@stop\fi{%
\CompReversetrue
    }{%
\sli@alpha#1#2\@nil#4\@nil
    }%
}
\def\sli@alpha@chars#1#2#3#4{%
    % Both #1 and #3 are characters.
    \@test\ifnum\uccode`#1=\z@ \fi{%
        \sli@alpha@chars@nocase
    }{%
        \sli@alpha@chars@ucase
    }%
    #1{#2}#3{#4}%
}
\def\sli@alpha@chars@nocase#1#2#3#4{%
    \@test\ifnum`#1>`#3 \fi{%
% First char is greater than second
\CompReversetrue
    }{%
\@test\ifnum`#1=`#3 \fi{%
    % Chars are equal.
    \sli@alpha#2\@nil#4\@nil
}{%
    % Second char is greater than the first.
    \CompReversefalse
}%
    }%
}
\def\sli@alpha@chars@ucase#1#2#3#4{%
    \@test\ifnum\uccode`#1>\uccode`#3 \fi{%
% First char is greater than second
\CompReversetrue
```

```
    }{%
\@test\ifnum\uccode`#1=\uccode`#3 \fi{%
    % Chars are equal.
    \sli@alpha#2\@nil#4\@nil
}{%
    % Second char is greater than the first.
    \CompReversefalse
}%
    }%
}
```