The **getitems** package: gathering `\item`'s from a list-like environment[*]

Anders Hendrickson
St. Norbert College, De Pere, WI, USA
`anders.hendrickson@snc.edu`

January 11, 2016

1 Overview

The `enumerate` and `itemize` environments of LaTeX organize their contents through the use of the `\item` command. Each entry in these lists is prefaced with the command `\item`, making for very compact and easily readable source code. Package designers may find it useful to use the same syntax for their custom environments. The getitems package makes it easy to code such environments by parsing a string of tokens, separating them by the occurrence of `\item`'s, and saving the contents as macros. Nested environments are handled correctly.

Moreover, some typesetting tasks naturally consist of a "header" followed by several related items; one example would be a multiple-choice question on a school examination. This package saves any TeX tokens appearing before the first `\item` as the zeroth item for special handling.

2 Usage

`\gatheritems` To parse a string of text, such as the body of an environment, call

`\gatheritems{`⟨*text to parse*⟩`}`.

This will scan through the ⟨*text to parse*⟩, dividing it at each `\item` while respecting TeX groupings and nested environments, and store the divided portions of text into memory.

`numgathereditems` The total number of items in the parsed text is stored in the LaTeX counter `numgathereditems`.

`\gathereditem` To retrieve a stored item, you may call `\gathereditem{`⟨*item number*⟩`}`; the ⟨*item number*⟩ should expand to an arabic representation of a nonnegative

---

[*] This document corresponds to getitems.sty v1.0, dated 2016/01/11.

integer. Any tokens occurring before the first `\item` may be retrieved with `\gathereditem{0}`.

`\loopthroughitemswithcommand` Once the items are gathered, it will probably be necessary to loop through all of them. Of course a package author can do so manually, but `getitems` provides a built-in way to do so by calling `\loopthroughitemswithcommand{`⟨*macro*⟩`}`. The ⟨*macro*⟩ must be a control sequence taking exactly one argument; it will be called successively with the item text. For example,

```
\gatheritems{%
  Zero
  \item One
  \item Two
  \item Three
}
\loopthroughitemswithcommand{\fbox}
```

One	Two	Three

The result is the same as processing `\fbox{One}`, then `\fbox{Two}`, and finally `\fbox{Three}`. Note that `\loopthroughitemswithcommand` deliberately ignores the zeroth entry, which occurs before the first `\item`.

`currentitemnumber` Typically the package author will create a custom macro to process each item. This macro may make use of the index of the loop, which is stored in the LaTeX counter `currentitemnumber`. A conditional `\ifgatherbeginningofloop` is also available, which only evaluates as true when processing the first item; it is thus functionally equivalent to `\ifnum1=\c@currentitemnumber`. The custom macro may take advantage of this to run special code for the first item only.

`\ifgatherbeginningofloop`

3 Example

An example using `getitems` to create a custom environment may be informative. We use the `\NewEnviron` command from the `environ` package (automatically loaded by `getitems`) to define a `question` environment; the body between the `\begin{question}` and `\end{question}` is available as `\BODY`.

```
\def\doitem#1{\item #1\hfill $\Box$}%
\NewEnviron{question}{%
  \expandafter\gatheritems\expandafter{\BODY}%
  \gathereditem{0}%
  \begin{itemize}
    \loopthroughitemswithcommand{\doitem}
  \end{itemize}
}
\begin{question}
  Who proved the unsolvability of the quintic?
  \item Abel
  \item Galois
  \item Lie
\end{question}
```

Who proved the unsolvability of the quintic? Check the appropriate box.

- Abel ☐
- Galois ☐
- Lie ☐

This second example shows that nested environments are handled as expected.

```
\def\doitem#1{\item[$\Box$]
              \fbox{\parbox[t]{1.75in}{#1}}}%
\NewEnviron{question}{%
  \expandafter\gatheritems\expandafter{\BODY}%
  \gathereditem{0}%
  \begin{itemize}
    \loopthroughitemswithcommand{\doitem}
  \end{itemize}
}
\begin{question}
  Who proved the unsolvability of the quintic?
  Check the appropriate box.
  \item Abel
        \begin{itemize}
          \item Born August 5, 1802
          \item Died April 6, 1829
        \end{itemize}
  \item Galois
        \begin{itemize}
          \item Born October 25, 1811
          \item Died May 31, 1832
        \end{itemize}
  \item Lie
        \begin{itemize}
          \item Born December 17, 1842
          \item Died February 18, 1899
        \end{itemize}
\end{question}
```

Who proved the unsolvability of the quintic? Check the appropriate box.

- ☐ Abel
  - – Born August 5, 1802
  - – Died April 6, 1829
- ☐ Galois
  - – Born October 25, 1811
  - – Died May 31, 1832
- ☐ Lie
  - – Born December 17, 1842
  - – Died February 18, 1899

4 Implementation

We need the `trimspaces` package to remove excess spaces from the items we find. Although the `environ` package is not used by `getitems` itself, it will almost certainly be needed.

```
\RequirePackage{environ}
\RequirePackage{trimspaces}
\let\xa=\expandafter
```

`\gathereditem` The kth item found will be stored in the macro `\getitems@item@`$\langle k\rangle$; the user can access it through the `\gathereditem` macro.

```
\def\gathereditem#1{\csname getitems@item@#1\endcsname}
```

`numgathereditems` We define the LaTeX counter `numgathereditems`.

```
\newcounter{numgathereditems}
```

`\gatheritems` The main control sequence of this package is `\gatheritems`. The naïve strategy is to use the delimiter mechanism of TeX to split the text at the first occurrence of the token "`\item`." We add `\getitems@relax` before, and "`\item\getitems@terminalitem`" after, the text to help us detect empty items and prevent errors after we have found all the genuine `\item`'s.

```
\long\def\gatheritems#1{%
  \setcounter{getitems@begindepth}{0}%
  \setcounter{numgathereditems}{0}%
  \xa\long\xa\gdef\csname getitems@item@0\endcsname{}%
  \gatheritems@int\getitems@relax#1\item\getitems@terminalitem\getitems@endgatheritems
  \xa\let\xa\gatheredheader\xa=\csname getitems@item@0\endcsname
}
```

The trouble with the naïve strategy is that it won't handle nested environments correctly. To do that, we need to keep track of how deeply nested we are with the macro `\getitems@trackbegindepth`, defined below. That macro stores its results in the LaTeX counter `getitems@begindepth`; a value of 0 indicates the top-level within the argument of `\gatheritems`.

```
\def\@getitems@terminalitem{\getitems@terminalitem}%
\def\@dummy@relax{\getitems@relax}%
\long\def\gatheritems@int#1\item#2\getitems@endgatheritems{%
  \getitems@trackbegindepth{#1}%
  \ifnum\c@getitems@begindepth=0\relax
```

At this point we have gathered a complete `\item`; we have not stopped accidentally at a sub`\item`. The original `\item` might have had no content, in which case `#1` will be simply "`\getitems@relax`", and we do nothing; otherwise we strip off the `\getitems@relax` and store those tokens in `\getitems@item@`⟨*numgathereditems*⟩.

```
    \def\getitems@test@i{#1}%
    \ifx\getitems@test@i\@dummy@relax
      \relax
    \else
      \xa\xa\xa\g@addto@macro
         \xa\xa\csname getitems@item@\the\c@numgathereditems\endcsname
            \xa{\getitems@stripfirsttokenfrom#1\getitems@endstrip}%
    \fi
```

Now we test whether we have reached the end of the text to be parsed. This is the case if `#2` is simply `\getitems@terminalitem`, and we stop the recursion. Otherwise there is at least one more `\item` to process, so we increment `numgathereditems`, prepare `\getitems@item@`⟨*k+1*⟩, and prepare to recurse.

```
    \def\getitems@test@ii{#2}%
    \ifx\getitems@test@ii\@getitems@terminalitem
      \let\getitems@next=\relax
    \else
      \stepcounter{numgathereditems}%
      \xa\gdef\csname getitems@item@\the\c@numgathereditems\endcsname{}%
      \def\getitems@next{\gatheritems@int\getitems@relax#2\getitems@endgatheritems}%
    \fi
  \else
```

We are now in the case where `getitems@begindepth` $\neq 0$. This essentially means that the text in `#1` has more `\begin`'s than `\end`'s, so we have not read a complete `\item`; we stopped at an "`\item`" token within a sub-environment. We save the text gathered so far to `\getitems@item@`$\langle k\rangle$, including the `\item` we parsed by mistake, and then call `\gatheritems@int` again to sweep up more tokens.

```
    \xa\xa\xa\g@addto@macro
       \xa\xa\csname getitems@item@\the\c@numgathereditems\endcsname
         \xa{\getitems@stripfirsttokenfrom#1\getitems@endstrip}%
    \xa\g@addto@macro\csname getitems@item@\the\c@numgathereditems\endcsname{\item}%
    \def\getitems@next{\gatheritems@int\getitems@relax#2\getitems@endgatheritems}%
  \fi
  \getitems@next
}
```

This next macro is used by `\gatheritems@int` to strip off a dummy `\getitems@relax` token from the beginning of its first parameter.

```
\long\def\getitems@stripfirsttokenfrom#1#2\getitems@endstrip{#2}
```

Here is the code used to track the depth of nesting of `\begin`'s in a text.

```
\newcounter{getitems@begindepth}
\long\def\getitems@trackbegindepth#1{%
  \getitems@trackbegindepth@int#1\getitems@terminalbegindepth\getitems@endtrackbegindepth
}
\def\@getitems@begin{\begin}%
\def\@getitems@end{\end}%
\def\@getitems@terminalbegindepth{\getitems@terminalbegindepth}%
\long\def\getitems@trackbegindepth@int#1#2\getitems@endtrackbegindepth{%
  \def\getitems@test@i{#1}%
  \ifx\getitems@test@i\@getitems@begin
    \advance\c@getitems@begindepth by 1\relax
  \else
    \ifx\getitems@test@i\@getitems@end
      \advance\c@getitems@begindepth by -1\relax
    \fi
  \fi
  \def\getitems@test@ii{#2}%
  \trim@spaces@in\getitems@test@ii
  \ifx\getitems@test@ii\@getitems@terminalbegindepth
    \let\getitems@trackbegindepth@next=\relax
  \else
    \def\getitems@trackbegindepth@next{%
      \getitems@trackbegindepth@int#2\getitems@endtrackbegindepth}%
  \fi
  \getitems@trackbegindepth@next
}
```

`\loopthroughitemswithcommand`

Finally, we define the user-level command to loop through the gathered items from 1 through `numgathereditems`.

```
\newif\ifgatherbeginningofloop
\newcounter{currentitemnumber}
```

```
\def\loopthroughitemswithcommand#1{%
  \setcounter{currentitemnumber}{1}%
  \gatherbeginningofflooptrue
  \loopthroughitemswithcommand@int{#1}%
}

\def\loopthroughitemswithcommand@int#1{%
  \ifnum\c@currentitemnumber>\c@numgathereditems\relax
    \let\getitems@loop@next=\relax%
  \else
    \xa\xa\xa#1\xa\xa\xa{\csname getitems@item@\the\c@currentitemnumber\endcsname}%
    \def\getitems@loop@next{\loopthroughitemswithcommand@int{#1}}%
    \stepcounter{currentitemnumber}%
  \fi
  \gatherbeginningofloopfalse
  \getitems@loop@next
}
```