dolines.tex

Meta-macros to separate arguments by newlines and by empty lines.
Version: 1.0 (15 April 1991) (doc: 15 Apr 1991)
Copyright © 1991 Marcel R. van der Goot

The macros in dolines.tex are intended to help you write your own macros, rather than to typeset text directly. Consequently, it is assumed that you have some experience writing macros; in other words, assume that this whole section is preceded by a "dangerous bend" symbol.

Dolines.tex* helps to write macros for which line breaks inserted by the user are important. When the definitions from dolines.tex are in effect, text consists of paragraphs separated by one or more empty lines (as usual). Each paragraph consist of one or more lines, where a line is terminated by a normal newline character, '`^^M`' (normally, TeX treats newlines as if they are spaces). Note that, as far as dolines.tex is concerned, a paragraph can only be terminated by an empty line, not by, e.g., `\par` or `\vskip`. To use dolines.tex, you have to define three macros, `\beforelines`, `\everyline#1`, and `\afterlines`. When dolines.tex detects the beginning of a paragraph, `\beforelines` is called. Next, for every line, `\everyline` is called with as argument a single macro. That macro has as its expansion the line that was read. (Usually, this is almost the same as saying that `\everyline` is called with the line as argument.) After the last line of the paragraph, `\afterlines` is called. No output is generated, other than what is produced by your own macros.

There are two ways of using dolines.tex. One is to read lines from a file, the other is to take the lines from the current input file. If you say

`\filedolines{`⟨file name⟩`}`

dolines.tex will attempt to read the file, calling `\beforelines`, `\everyline`, and `\afterlines` as described above. The file is read inside a group, so you may need to use `\global` if you do assignments within, e.g., `\everyline`.

As an example, we define a "poetry environment" which reads a poem from a file, preserving the poet's line breaks. We draw horizontal lines around the poem, and put every stanza in a vbox to prevent page breaks. (It is certainly possible to write better poetry environments, we are just illustrating dolines.tex.)

```
\input dolines
\def\readpoem#1{\medskip \hrule \filedolines{#1}\hrule \medskip}
\def\beforelines{\medskip \vbox\bgroup}
\def\everyline#1{\hbox{\kern\parindent \it #1}}
\def\afterlines{\egroup \medskip} % end of the vbox
```

If file "epic.tex" contains a poem, we can typeset it with `\readpoem{epic}`.

However, if you are typesetting several short rhymes, it is somewhat inconvenient to have each of them in a separate file. This is where the second method of using dolines.tex is useful. You have to define a macro `\enddolines`, the expansion of which should be the text you want to use to terminate dolines.tex's effect. You also have to define a macro `\finishdolines`. When you say `\begindolines`, dolines.tex starts reading lines, calling `\everyline` etc. This continues until dolines.tex reads a line equal to the (top level) expansion of `\enddolines`. Then your macro `\finishdolines` is called, and TeX resumes normal processing. `\begindolines` and the line equal to `\enddolines` form a group (as with `\filedolines`); `\finishdolines` is called *outside* this group.

Here is how we can extend the poetry environment with two control sequences `\beginpoem` and `\endpoem`. We keep the above definitions of `\beforelines` etc.

```
\def\beginpoem{\medskip \hrule \begindolines}
\def\enddolines{\endpoem}
\def\finishdolines{\hrule \medskip}
```

* To use dolines.tex you also need Midnight/loop.tex, which will be `\input` when you `\input dolines`.

Now we can typeset a poem as follows (the English stanza is from The TeXbook):

```
\beginpoem

Roses are red,
\quad Violets are blue;
Rhymes can be typeset
\quad With boxes and glue.

Blauw de viooltjes,
\quad Rood zijn de rozen;
Een rijm kan gezet
\quad Met plaksel en dozen.
\endpoem
```

Note that we never defined `\endpoem`, as it is never expanded. In fact, there is no reason to terminate the poem with a control sequence. If we define `\def\enddolines{***}`, three stars on a single line will play the rôle of `\endpoem` above.

There are a few things to be careful about. First, a line must have properly nested `{...}` groups. (Because a line is read as an argument delimited by a ⟨return⟩ character.) You should not use this to fold long lines, because that will insert an explicit ⟨return⟩ character with catcode 12 in the line. Second, spaces at the beginning of a line are skipped, unless you change the catcode of ‘␣’. This means that, for instance, there can be spaces in front of `\endpoem`. (Spaces are skipped at the beginning of a line because at that point TeX is in “state N” — see Chapter 8 of The TeXbook.) Third, when `\everyline` is called, the line that is its argument is already tokenized. This means that a catcode change within `\everyline` cannot effect this line. Furthermore, when `\beforelines` is called, the first line is already read and tokenized.

`\filedolines` and `\begindolines` treat lines in pretty much the same way. There is however a significant difference in the way comments are treated. If a line has a comment (i.e., it ends with ‘`%...`’), `\begindolines` will miss the ⟨return⟩ at the end of the line. This is normal for TeX, it can be used to fold long lines. However, if you are using `\filedolines`, comments will be skipped, but lines are not concatenated: `\filedolines` will not miss the ⟨return⟩! (This is because `\filedolines` uses TeX’s `\read` primitive.) A rather obscure difference is that a line consisting of just ‘`\par`’ is read as an empty line by `\filedolines`, whereas it is read as a non-empty line by `\begindolines`. If you want to use `\begindolines` to read from a file, you can say `\expandafter\begindolines\input ....`

It is possible for `\finishdolines` to take arguments, but they have to appear on the line following the line with the expansion of `\enddolines`. Suppose we change the poem environment by defining `\finishdolines` as

```
\def\finishdolines#1%
   {\line{\strut \hss --- #1\hss\hss}\hrule \medskip}
```

Hence, `\finishdolines` will print the author’s name under the poem. The argument should be written on the line following `\endpoem`:

```
\beginpoem
   ...
\endpoem
{A.U. Thor}
```

Sometimes we want to use dolines.tex to read just one paragraph by making `\enddolines` equal to an empty line. For instance, a macro `\address` which reads an address terminated by an empty line. The problem with defining `\def\enddolines{}` is that if there is an empty line before the address (or even if the rest of the line after `\address` is empty), no lines will be read at all. To ensure that at least one paragraph is read, we use `\beforelines` to define `\enddolines`:

```
\def\beforelines{\normalbeforelines\def\enddolines{}}
\let\enddolines=\relax
```

Here, `\normalbeforelines` denotes whatever must be done at the beginning of the address. Initially `\enddolines` is equal to `\relax`, but as soon as the first non-empty line is found, `\enddolines` is redefined as an empty line. Following that, an empty line causes calls of `\afterlines` and `\finishdolines`. (A way to include an empty line in the address is to write a tie ('~') on a line of its own.)

As a more complicated example, we design a simple verbatim environment. The verbatim environment treats horizontal tabs as spaces, and considers multiple empty lines equal to a single empty line. We first write macros `\setupverbatim` and `\finishverbatim`. The first sets up the character codes for verbatim printing, the second resets those character codes.

```
\def\setupverbatim
   {\medskip \hrule \bgroup
    \def\do##1{\catcode`##1=12 }\dospecials
    \obeyspaces \tt
   }

\def\finishverbatim{\egroup \hrule \medskip}

{\obeyspaces\global\let␣=\␣}
```

Grouping is used to keep the catcode changes local. Plain TeX's `\dospecials` macro applies the macro `\do` to all special characters ('\\', '$', etc.), which, in this case, gives them all catcode 12 ("other"). But we don't want catcode 12 for spaces, because the `\tt` font has a visible space character '␣'. Therefore, spaces are made active with `\obeyspaces`, and in the last line we make an active space equal to `\␣`.

To start setting verbatim text, we use `\beginverbatim`; `\readverbatim` is used to typeset a whole file in verbatim mode. The macros for typesetting lines and paragraphs are very straightforward.

```
\def\beginverbatim{\setupverbatim\begindolines}
\let\finishdolines=\finishverbatim

\def\readverbatim#1{\setupverbatim\filedolines{#1}\finishverbatim}

\def\beforelines{\smallskip}
\def\everyline#1{\hbox{\strut #1}}
\def\afterlines{\smallskip}
```

Finally, we have to define `\enddolines`. We want to match `\beginverbatim` with `\endverbatim`, so we would like to do

```
\def\enddolines{\endverbatim}
```

The problem is that this makes `\enddolines` expand to the *control sequence* `\endverbatim`. But while we are in verbatim mode, no control sequences are recognized, because '\\' is not an escape character. What we really want is to make the expansion of `\enddolines` equal to the *string* "\endverbatim". This is done by temporarily making '\\' a normal character and '/' an escape character:

```
{\catcode`\/=0 /catcode`/\=12
 /gdef/enddolines{\endverbatim}
}
```

This completes the verbatim environment. Note that `\endverbatim` is only recognized if it appears on a line by its own and is not preceded by spaces. If you are not afraid of confusion, you can change the name `\finishverbatim` to `\endverbatim`.