The stackengine Package

Highly customized stacking of objects, insets, baseline changes, *etc.*

Steven B. Segletes
steven.b.segletes.civ@mail.mil

July 28, 2016
v3.26

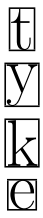
A sectional index for this package is found in §17.

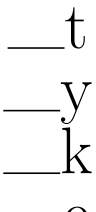
1 Definitions and Terms

Before I describe the various commands of this package, it is best for you to familiarize yourself with a few terms as used in this package.

1.1 Short Stacks and Long Stacks

In this package, "Short" refers to squeezing the extra space out of glyphs comprising the stack, so that the inter-item gap is solely what is set. Likewise, "Long" refers to a stack kept at constant inter-item baseline spacing. Let's look at the following two examples to see this clearly.

Notice the consistent inter-item gap on a short stack: 

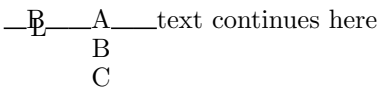
Notice the consistent inter-baseline spacing on a long stack: 

In this package, the default spacing (*i.e.*, stacklength) for the long

stack, saved in `\Lstackgap` is the length `\normalbaselineskip` (note that `\normalbaselineskip` has the same value as `\baselineskip`, but can be safely used inside a `tabular` environment). For short stacks, the default stacklength, stored in `\Sstackgap`, is 3pt, which corresponds to the same spacing found in TEX's `\shortstack` command. The default stack type constructed by this package is short, denoted by the definition of `\stacktype` as "S" (as opposed to "L" for long stacks). These defaults can be changed.

1.2 Baseline

The baseline is the horizontal line on which the current (unstacked) text is set. For non-descending letters, the bottom of the letters sit upon the baseline. In this document, we will graphically use the symbol —B— to denote the current baseline of the text, as in



1.3 (Over)Stacks and Understacks

In general terms, a stack is built up from the baseline, while an understack is built down from the baseline.

So, for example,

is a stack, while a is an understack.

With this package, both stacks and understacks can be long or short, depending on how the invocation is made. Sometimes, in this documentation, a stack will be called an "overstack" in order to avoid confusion with "understack."

1.4 Anchors

The term "anchor" is used to denote an item in the stack whose baseline does not change in the course of the stacking operation. In essence, stacking is done relative to the anchor.

For example, in

, the letter "c" is the anchor.

In the understack a, “a” is the anchor.
b
c

1.5 Alignment

Anyone familiar with LaTeX knows something about the horizontal alignment settings “l”, “c” and “r”. As pertains to a stack, we refer to the alignment of the various rows of the stack, as shown in the following examples.

	abcd		abcd		abcd
	abc		abc		abc
	ab		ab		ab
Left:	a	Center:	a	Right:	a

The default alignment within a stack may be set with the `\stackalignment` definition, to possible values of `l`, `c`, or `r`, denoting left, center, and right alignments, respectively. In addition, a number of the stacking macros provide the alignment as an argument or optional argument that can be specified at time of invocation.

The stackengine package defaults to center alignment, unless reset by way of the `\stackalignment` definition.

2 Parameter Defaults, Modes, and Macros Provided by stackengine

2.1 Parameter Defaults and How to Set Them

There are several macros that serve to store package default definitions. When a stacking macro does NOT explicitly ask for one of the quantities defined by these defaults, it will use the values of these parameters to serve as the default value. However, some stacking macros may ask the user to specify an explicit value for any number of these parameters in the calling arguments. In that case, an explicit value may be provided or else the macro holding the default value may itself be provided as the argument.

Let us first consider the stackgap defaults, associated with short and long stacks. Those stackgaps are actually stored in the `\def`'s

```
\Sstackgap
\Lstackgap
```

`\Sstackgap` The parameter `\Sstackgap` is the default inter-item gap for any stacking com-

\Lstackgap mand that builds a short stack. It defaults to 3pt. Correspondingly, \Lstackgap is the inter-item baselineskip for any stacking command that builds a long stack. It defaults to \normalbaselineskip. To change these defaults, the following \setstackgap macro is provided:

\setstackgap{S}{*inter-item stackgap*}
\setstackgap{L}{*inter-item baselineskip*}

\stackgap In addition, there is the \stackgap macro, which may be invoked by the user. This macro will look at the current value of \stacktype and will output either \Sstackgap or \Lstackgap, depending on whether \stacktype is defined as "S" (short) or "L" (long).

In addition to these lengths, there are several parameter definitions that define default behaviors for those stacking macros when explicit values of these settings are not requested. The parameters, each set with a \def (or a \renewcommand), are

\def\stackalignment{l *or* c *or* r}
\def\quietstack{T *or* F}
\def\useanchorwidth{T *or* F}
\def\stacktype{S *or* L}

\stackalignment The parameter \stackalignment defaults to "c" (center) and defines the default stacking alignment. Other options include "l" (left) and "r" (right). See §1.5 for more discussion of this parameter.

\quietstack The parameter \quietstack defaults to "F" (false) and determines whether the result of a stacking operation is suppressed from the output. If the output is suppressed by setting this parameter to "T", the most recent stacking operation may be recovered with a \usebox{\stackedbox}. See §3.1 for more discussion of this parameter.

\useanchorwidth The parameter \useanchorwidth defaults to "F" (false) and indicates that, upon creation, the stack width should be taken as the widest element of the stack. When set to "T", the stack width, upon creation, is instead taken as the width of the anchor. Note that this parameter applies at the time of stack creation and cannot be used to retroactively change the characteristics of an existing stack. See §8 for more discussion of this parameter.

\stacktype The parameter \stacktype defaults to "S" (short stack) and defines the default type of stack to build. When set to "L", newly created stacks are, by default, long stacks. Refer to §1.1 for a definition of these terms.

For macros that require explicit specification of some of these parameters, the parameter token itself (*e.g.*, \stackalignment, \quietstack, *etc.*) may be provided as the argument, if the default behavior is desired.

2.2 Stacking Modes

In addition to parameter definitions, there are several stackengine mode settings:

\stackMath
\stackText
\strutlongstacks{T *or* F}
\strutshortanchors{T *or* F}
\setstackEOL{*end-of-line character*}

\stackMath
\stackText

By default, this package will assume that the arguments of the stacking commands are text. However, an invocation of \stackMath will change the default so that stackengine arguments will be processed in inline math mode (so-called \textstyle math). Likewise, an invocation of \stackText will reset the default processing of arguments to text mode. Both of these modes are global in their scope, so that invocation, even within a macro, will continue to persist until later changed. If you would like to have a single application stacked in math mode, without changing the default mode setting, one could use the \ensurestackMath{} macro (see 2.3.8).

\strutlongstacks

By default, stacks made by stackengine are set in unframed boxes that are flush with the top and bottom of the stack. The \addstackgap macro (§2.3.11) is one way to add a buffer space above and below a stack. Such an approach makes more sense for short stacks, where a constant buffer size determines the gap between objects. But for long stacks, a method involving \strut's makes more sense. The mode \strutlongstacks, when set true (T), will automatically cause stackengine to condition each row of a long stack with a \strut during its construction (see §9 for more details).

\strutshortanchors

The \strutshortanchors macro (§6.1) is used to set the mode of how two items may be vertically combined into a split short-stack anchor. By default true (T), a split short-stack anchor is set so that the middle of the stack gap falls halfway up the height of the \strut. This will give the short-stack anchor the appearance of being split relative to text that sits atop the baseline. When set false, however, any short-stack anchor that is constructed has its gap split across the baseline itself.

\setstackEOL
[usestackEOL] package option

Until V3.1 of stackengine, the \Longstack, \Shortstack, \Longunderstack, and \Shortunderstack macros used, exclusively, space tokens as the end-of-line separators between rows of these stacks. Not only is such a syntax atypical of LaTeX, but it can be inconvenient when the items within a stack row also contain spaces. The \setstackEOL macro allows the end-of-line (EOL) character to be defined to a more convenient setting, such as \\, though any character will work. There is a package option [usestackEOL] which sets the end-of-line symbol to be used when parsing stack arguments to \\. When the stacking EOL is something other than a space token, leading and trailing spaces are

ignored in the row data for the four cited stacking commands, as well as their derivatives `\Centerstack` and `\Vectorstack`. (Note: other stacking commands in the `\stackon` / `\stackunder` family will not ignore, but retain spaces in their arguments, since there is no issue associated with EOL seperators.)

Note that these five mode-setting commands are not themselves parameters to be part of a `stackengine` argument, but are instead to be issued as separate commands prior to the invocation of a desired stacking macro.

2.3 Stacking Macros

2.3.1 `\stackengine`

The basic (and most general) stacking macro provided by this package is `\stackengine`. Nearly all other macros of this package merely provide an abbreviated invocation form of this macro, tailored for a particular application. The syntax is:

`\stackengine`{`\Sstackgap` *or* `\Lstackgap` *or* `\stackgap` *or stacklength*}
{*anchor*}
{*item*}
{`O` *or* `U`}
{`\stackalignment` *or* `l` *or* `c` *or* `r`}
{`\quietstack` *or* `T` *or* `F`}
{`\useanchorwidth` *or* `T` *or* `F`}
{`\stacktype` *or* `S` *or* `L`}

There are no optional arguments with `\stackengine`. The first item is a stacking length, which can mean different things depending on whether a short- or long-stack is being created, as discussed in §1.1. The second argument is the stack anchor, whose baseline does not change. The third argument is the item stacked relative to the anchor. The fourth argument is either an "O" (for a normal or overstack) or a "U" (for an understack). The fifth argument denotes a left, center, or right alignment of the stack items with the use of "l", "c", or "r". The sixth argument is either a "T" or "F" to denote whether the resulting stack is NOT printed ("T" denotes "do NOT print"). The seventh argument, also a "T" or an "F", denotes when "T" that the width of the whole stack is to be taken as the width of the anchor. Otherwise, the width of the stack conforms to the width of the widest item in the stack. The final argument is defined as either an "S" or an "L" to denote whether a short stack or a long stack is being requested.

While none of the arguments to `\stackengine` have default values, there are various definitions in the package that store default values. When one wishes

a stack to possess certain default properties, these default definitions may be passed as the values for the respective arguments to `\stackengine`. Arguments 1, 5, 6, 7, and 8 each possess a corresponding parameter containing the default variable. So, for example,

```
\stackengine{\stackgap}{A}{BC}{O}{\stackalignment}
            {\quietstack}{\useanchorwidth}{\stacktype}
```

will stack the letters "BC" over "A" using the default values of `\stackalignment`, `\quietstack`, `\useanchorwidth`, and `\stacktype`. In addition, the stackgap will be set to `\Sstackgap` if `\stacktype` is "S" or to `\Lstackgap` if `\stacktype` is "L". Alternately, the same stack could be forced to be a long stack with right-alignment and be typeset using

```
\stackengine{\Lstackgap}{A}{BC}{O}{r}{F}{\useanchorwidth}{L}
```

While the stacking macros in the following sections offer the convenience of a condensed syntax, which can be useful within your document, it may be preferable to use the `\stackengine` macro to create your own stacking macros, since one can achieve the stacking result without having to modify the package defaults.

2.3.2 `\stackon` and `\stackunder`

The `\stackon` and `\stackunder` macros are abbreviated forms of `\stackengine` in which the default values of parameters are taken. The only exception is the stacklength, which may be provided as an optional argument.

`\stackon[`*stacklength*`]{`*anchor*`}{`*item* `}`
`\stackunder[`*stacklength*`]{`*anchor*`}{`*item* `}`

In the case of `\stackon`, an (over)stack is created, whereas an understack is created with `\stackunder`. With both of these commands, the anchor is the first mandatory argument, and the item to be stacked above or below it is the second mandatory argument.

2.3.3 `\Shortstack` and `\Longstack`

The `\Shortstack` and `\Longstack` commands are stacking commands that allow more than two stacking rows to be specified at once. The items of a stack are provided, by default, in a space-separated list, as follows:

`\Shortstack[`*alignment*`]{`*item* `{`*it em*`}` `{\`*item*`}` `$`*item*`$` *... anchor*`}`
`\Longstack[`*alignment*`]{`*item* `{`*it em*`}` `{\`*item*`}` `$`*item*`$` *... anchor*`}`

The first item is at the top of the stack, farthest from the baseline, while the last item is the anchor of the stack. If an item (the contents of a single row) contains spaces, enclose the item in braces. If the item ends with a macro, enclose the item in braces. If used in a mixed text/math mode (see §13 for more details), math items may be enclosed within dollar-sign delimiters. As of V3.2, these commands have been made robust and their arguments need not be `\protect`ed.

The alignment may be specified as an optional argument, with other parameters taken by their defaults. The stacklength is taken as `\stackgap`, which will depend on the value of `\stacktype`.

To give an example of the braced syntax described above, the stack

a
b c
¶
β^2

`\Shortstack[r]{a {b c} {\P} $\beta^2$ ANCHOR}` will produce ANCHOR. Note that the use of the math delimiter (`$`) in the `\Shortstack` argument was needed only if the stack contains both inline math as well as text elements. If the stack is to contain only math-mode elements, then math arguments can be made the default:

```
\stackMath\Shortunderstack[c]{{\alpha} {\beta} {\gamma}}
```

yielding α
β
γ

As of V3.1 of stackengine, the user may define the the argument to `\Longstack`, `\Shortstack`, `\Longunderstack`, and `\Shortunderstack` with a row separator different than a space. The `[usestackEOL]` package option will make the EOL character within these stacking commands a `\\` character instead of a space. Here are an example of equivalent stack arguments for two different EOL characters:

Space EOL: `{a {b c} {\P} $\beta^2$ ANCHOR}`
`\\` EOL: `{a \\ b c \\ \P \\ $\beta^2$ \\ ANCHOR}`

Note that leading and trailing spaces surrounding the `\\` are ignored for rows in the second case. The `\setstackEOL` macro allows any arbitrary character to be set as the "EOL" character (for the purposes of parsing these stacking macros).

2.3.4 `\Shortunderstack` and `\Longunderstack`

The `\Shortunderstack` and `\Longunderstack` commands are the understacking equivalents of `\Shortstack` and `\Longstack`.

`\Shortunderstack[`*alignment*`]{`*anchor item* `{`*it em*`}` `{\`*item*`}` `$`*item*`$` *...*`}`
`\Longunderstack[`*alignment*`]{`*anchor item* `{`*it em*`}` `{\`*item*`}` `$`*item*`$` *...*`}`

The caveats to the specification of the space-separated list of items is the same as that mentioned above for `\Shortstack` and `\Longstack`.

Note that the items are still specified from the top downward. In the case of understacks; however, that means that the anchor is the first item specified, while the last item in the list is farthest below the baseline.

2.3.5 Top and bottom lapping macros

Top
and bottom lapping is achievable through the creation of various long stacks. The syntax is the same for all six of the macros provided. The six names differentiate whether the lap is to the top (t) or the bottom (b) of the calling point, and whether it should appear to the left (l), centered (c), or to the right (r) of the calling location. The long-stacklength may be provided as an optional argument.

`\tllap[`*stacklength*`]{`*item*`}`
`\tclap[`*stacklength*`]{`*item*`}`
`\trlap[`*stacklength*`]{`*item*`}`
`\bllap[`*stacklength*`]{`*item*`}`
`\bclap[`*stacklength*`]{`*item*`}`
`\brlap[`*stacklength*`]{`*item*`}`

See §5 for more information and examples of use for these lapping commands. If trying to remember the names of all these lapping commands is too tedious for you, two functionally identical macros have been introduced:

`\toplap[`*stacklength*`]{`*lap H-direction*`}{`*item*`}`
`\bottomlap[`*stacklength*`]{`*lap H-direction*`}{`*item*`}`

The names are simpler to remember, but they require, as their first mandatory argument, the horizontal direction of lapping (l, c, or r), while the material to be lapped is now in the second mandatory argument.

2.3.6 `\stackanchor`

The `\stackanchor` macro is one way to create a stack where the baseline is split between two items in the stack. The syntax is

`\stackanchor[`*stackgap*`]{`*top item*`}{`*bottom item*`}`

The stacking gap may be provided as an optional argument. All other parameters taken are the defaults. There are subtle differences in the way a short stack anchor versus a long stack anchor is split. One should consult §6.1 for a description of how the stack is constructed.

If a short stack is being created with a split anchor, the `\strutshortanchors{}` mode setting will determine whether the split occurs at the mid-height of the strutbox ("T") or whether it will occur about the baseline ("F").

If a long stack is to be created with a split anchor, it is best to formulate and `\savestack` both the upper portion (with a stack) and the lower portion (with an understack), and then `\stackanchor` the two parts together. Alternately, one could use the `\Centerstack` macro (§2.3.7).

2.3.7 `\Centerstack` and the math variants of `\Vectorstack`

The macro `\Centerstack` produces a type of `\Longstack` that is vertically centered about the mid-strut above the baseline. In essence, it is like a `\stackanchor` version of a `\Longstack`. Additionally, it is locally processed with the mode `\strutlongstacks` set as `T`, so as to ensure the proper vertical centering.

The `\Centerstack` macro has a math-mode equivalent called `\Vectorstack`, in which the stacking argument is automatically processed in math mode, regardless if whether the `\stackMath` mode is currently active.

The syntax is the same as `\Longstack`, except that the anchor is not specified, but results naturally from the centering operation:

`\Centerstack[`*alignment*`]{`*item* `{`*it em*`}` `{\`*item*`}` `$`*item*`$` *... item*`}`
`\Vectorstack[`*alignment*`]{`*item* `{`*it em*`}` `{\`*item*`}` *... item*`}`

One of the main purposes of providing this vertically centered stack is to be compatible with mathematical bracketing commands embodied in the use of the LaTeX syntax `\left(...\right)`, which require vertically centered material. To this end, there are several variants of `\Vectorstack` macro that are for specific use in mathmode. Here are examples of all of these macros demonstrated:

$$\begin{array}{ll} \mathrm{p} & p \\ \mathrm{q} & q \\ \mathrm{r} & r \\ \mathrm{s} & s \end{array} \quad \begin{pmatrix} p \\ q \\ r \\ s \end{pmatrix} \quad \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} \quad \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \quad \begin{vmatrix} p \\ q \\ r \end{vmatrix}.$$

which are, respectively,

```
\Centerstack{p q r s}
\Vectorstack{p q r s}
$\parenVectorstack{p q r s}$
$\bracketVectorstack{p q r s}$
$\braceVectorstack{p q r}$.
$\vertVectorstack{p q r}$.
```

2.3.8 `\ensurestackMath`

The macro `\ensurestackMath{}` macro is used to force its argument (a stacking command) to be processed in math mode, without setting the global default stacking mode to math mode (otherwise accomplished via `\stackMath`). Note that this macro does **not** ensure that all elements of its argument are processed in math mode (*i.e.*, it does not use, nor is it a replacement for `\ensuremath`), but only that stacking macros are processed in math mode.

2.3.9 `\abovebaseline` and `\belowbaseline`

The macros `\abovebaseline` and `\belowbaseline` also creates the means to shift an item's baseline. The syntax is

`\abovebaseline[`*stackgap*`]{`*item*`}`
`\belowbaseline[`*stackgap*`]{`*item*`}`

where the stackgap defines the stackgap distance. In a long stack, it will define the upward (for `\abovebaseline`) or downward shift (for `\belowbaseline`) of the baseline of the item. In a short stack, it will define the upward/downward vertical gap between the original baseline and the bottom/top of the shifted item.

For long stacks, a negative stackshift for `\belowbaseline` will produce the same result as the corresponding positive stackshift for `\abovebaseline`. But this will not be the case for short stacks. One should consult §6.2 for more details.

This command can also be employed to redefine the baseline of graphical objects, which can provide a useful technique to achieve horizontal alignment with other objects. Negative stack gaps are valid (see §6.2 for examples). While these macros will carry slightly more processing overhead than a simple `\raisebox`,

it may be the case that specifying the shift in terms of the baseline, rather than with respect to the current configuration, may make for more streamlined logic.

2.3.10 `\stackinset`

The macro `\stackinset` is for insetting smaller items (*e.g.*, images or text) inside of a larger background items (*e.g.*, images or text). The syntax is

`\stackinset{`*H-align*`}{`*H-offset*`}{`*V-align*`}{`*V-offset*`}{`*inset item*`}{`*anchor item*`}`

In addition to providing the inset and background items as the last two arguments, the inset may be vertically and horizontally offset relative to various references. The first argument *H-align*, is either an `l`, `c`, or `r`, indicating whether the horizontal offset is relative to the left, center, or right of the anchor. The second argument is a length corresponding to the actual horizontal offset, relative to the horizontal alignment. For `l` and `c` alignments, a positive horizontal offset is rightward, whereas it is leftward for a `r` alignment.

In the case of vertical measures, the third argument, *V-align*, can have values of `t`, `c`, or `b`, corresponding to top, center, and bottom, respectively. The fourth argument is a length corresponding to the vertical offset from the alignment location. For `b` and `c` alignments, a positive vertical offset is upward, whereas it is downward for a `t` alignment.

For horizontal and vertical offsets, negative lengths are allowed. However, this macro is always executed with the `\usestackanchor` parameter set as true (meaning that the resultant stack will be the horizontal size of the anchor, even if the inset spills outside of it).

While this command was designed with inlaid images in mind (see §10), there is nothing that prevents its use to inlay character glyphs upon each other, as in ⊛, accomplished with `\stackinset{c}{}{c}{-.25ex}{*}{O}`.

2.3.11 `\addstackgap`

The `\addstackgap` macro provides a user-requested feature to add a specified gap above and below an object. The syntax is

`\addstackgap[`*stackgap*`]{`*item*`}`

When a stack is otherwise created, it by default will vertically extend from the top of the top-most item in the stack to the bottom of the bottom-most item in the stack, but with no extra gap applied. Depending on how the user employs that stack, there may not be enough vertical gap between the stack and its

adjacent objects. This macro will apply a fixed gaplength (either specified or, by default, an amount given by `\Sstackgap`) above and below the argument of the macro. Negative gaps will have no effect on the vertical extent of the item.

Here's an example of a stack, before and after `\addstackgap` is applied, framed for clarity: `\stackon{g}{a}`: versus `\addstackgap{\stackon{g}{a}}` . This macro provides a slightly different result than the `\strutlongstacks` mode described in §2.2. For more details of the comparison, see §9.

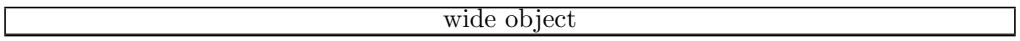
2.3.12 `\hsmash`

The macro `\hsmash` is a horizontal version of the popular `\smash` macro:

`\hsmash{`*item*`}`

In this case, the result is centered over its zero-width anchor position. If one wanted left or right aligned versions of this macro, the existing LaTeX macros `\rlap` and `\llap` would suffice.

The primary utility of this macro is in applying `\hsmash` to an object (often wider than the `\textwidth`), in order to extend the object into the margins in a controllable manner, such as in the case of these items below:



2.3.13 `\savestack`

The macro `\savestack` provides a means to save an intermediate result (in a box that can be recalled with the specified macro) without printing it. Its syntax is

`\savestack{`*macro token*`}{`*stacking operation*`}`

The macro token is a backslashed name that you can later use to recall the boxed intermediate result. The stacking operation is the intermediate result you wish to save in a box without printing. Reasons why this might be a useful macro are described in §3.2.

Note that while `\savestack` is designed to save an intermediate stacking operation in a box, the reality is that you can use this macro to save any sequence of boxable output, even if it is not a stacking operation. So, for example,

`\savestack{\mymacro}{\fbox{A}}` can later be recalled with an invocation of `\mymacro`, to yield A.

3 Making a Stack in Stages

There are times where one needs to make a stack in several consecutive steps. In order to do so, there are several ways, detailed below.

3.1 Suppressing Output

One can define `\quietstack` as `T` to suppress output. Then, the intermediate stack can be passed as an argument to the next stacking operation with a `\usebox{\stackedbox}`. That same `\usebox` may be used to output the final stack, as long as `\quietstack` remains defined as "T". Otherwise, `\quietstack` must again be defined "F" (false) to re-enable automatic output.

3.2 Saving a Stack without Printing It

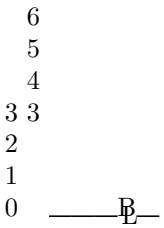
Perhaps a quicker and better way is to use the package's `\savestack` macro, of the form `\savestack{`*macro token*`}{`*stacking operation*`}` . This will perform the stacking operation and place it into the supplied token (without printing it), so that the intermediate stack can be passed to the subsequent stacking operation by executing the token macro.

For example,

```
\savestack{\pdq}{\Longstack[r]{6 5 4 {3 3}}}%
\Longstack[l]{{\pdq} 2 1 0}%
```

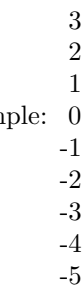
6
5
4

would first place "3 3" into `\pdq`, without printing it, in a right-aligned fashion, with the "3 3" on the baseline. Then, the second `\Longstack` would place the first stack atop of the "2" in a left-aligned fashion. The resulting stack would look as follows, with the "0" on the final baseline.



4 Choosing the Anchor Element

Nearly all stacking done with the stackengine package has a designated anchor element. The baseline of the anchor does not change. Instead, the other items are placed relative to the anchor. By adding items at first above and then below the anchor, any arbitrary stack can be built around the anchor.

Here is a Longstack/understack example:  that was achieved as follows:

```
\savestack{\pdq}{\Longstack{3 2 1 0}}%
\Longunderstack[r]{{\pdq} -1 -2 -3 -4 -5}
```

In the first stack, "0" is the anchor of `\pdq`. In the second stack, `\pdq` is the anchor (which means that "0" remains the anchor of the composite stack).

5 Top and Bottom Lapping

Top and bottom lapping can be accomplished with a long stack, using a null item as the anchor, and `\useachorwidth` set to `T`. This technique is encoded in six `\`*xy*`lap` commands, where *x* can be `t` or `b` (for "top" and "bottom," respectively) and *y* can be `l`, `c`, or `r` (for "left," "center," and "right," respectively). Note that this *y* is not the same as the `\stackalignment`, but rather refers to the direction where the lapped item is placed relative to the lap invocation.

For illustration only, laps will be shown at a *visible* lapmark (given by `|`), defining

the point of invocation. In these examples, I lap a `\parbox` above or below the selected location.

this is a top-left lap `\tllap` this is a top-right lap `\trlap`

For example here or else here, or else I can put it here or here.

this is a bottom-left lap `\bllap` this is a bottom-right lap `\brlap`

this is a top-center lap `\tclap`

Of course, I can perform center lapping, as well. To see, for example, how the this is a bottom-center lap `\bclap` above was achieved, the last lap in the prior sentence was achieved by invoking

```
\tclap{\fbox{\parbox[b]{1.1in}{this is a top-center lap \tcl}}}
```

in the middle of the word "well." (Note that `\tcl` is a box containing the verbatim text "`\tclap`," created with the verbatimbox package.)

Note that `\toplap` and `\bottomlap` have also been introduced as macros that are functionally identical to these six lapping commands. While requiring an extra argument, their names may be easier for you to remember. See §2.3.5.

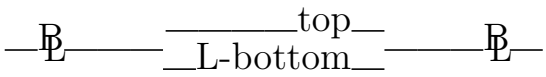
6 Shifting the Anchor's Baseline

By default, the baseline of the anchor becomes the baseline of the stack. If one wanted the stack baseline to **not** conform to the baseline of any of the individual stacked items, something needs to change. If one quantitatively knows the shift desired, a simple `\raisebox` could be used to vertically shift the anchor, after (or possibly prior to) stacking. There are also some other options provided by the package.

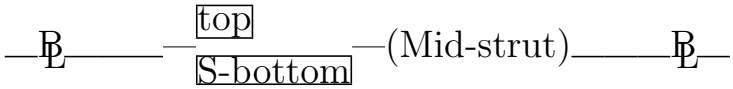
6.1 The `\stackanchor` Macro for "L" and "S" Stack Types

If one would like the anchor's baseline to be "midway" between two vertically arrayed items, the `\stackanchor` macro attempts to provide that capability for both "L" and "S" stack types, though the definitions of "midway" are different for each. The macro `\stackanchor` works for two-item stacks in both "L" and "S" stack types, as shown below.

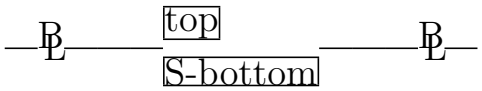
In the case on a long stack, what is satisfied is that the baseline of the anchor is made equidistant from the baselines of the two respective items comprising the stacked anchor.



In the case of a short stack, the middle of the inter-item gap is vertically located at the center of the font's `\strutbox`, so that the middle of the stackgap would be at the same vertical level as the middle of parentheses, were they located on the original baseline.

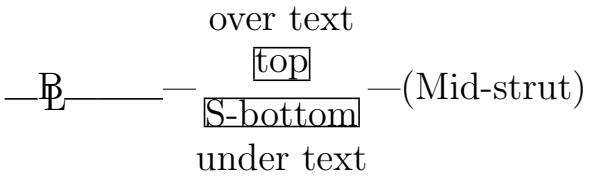


If one does not want to have the short-stack gap split at the mid-strut height, but rather at the baseline, the `\strutshortanchors{F}` mode may be set prior to the `\stackanchor` invocation.



The `\strutshortanchors` mode will only affect short stacks (`\stacktype` "S"), since long stacks do not use a strut as part of the `\stackanchor` definition.

For stackanchors with more than two rows, one should compose the top portion with a stack and the bottom portion with an understack, and then, finally, shift the anchor by applying `\stackanchor` to these two "substacks."



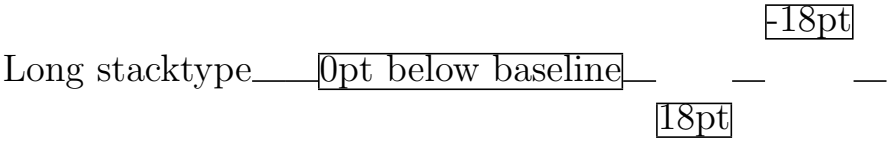
6.2 `\belowbaseline` for Moving the Anchor's Baseline

The command `\belowbaseline` is intended to be used to set an item, such as a stack anchor, below the baseline (this discussion is equally valid for the macro `\abovebaseline`, but in the opposite sense). The macro accomplishes this task by stacking the item below a null object. However, `\useanchorwidth` is temporarily set to `F` so that the stacked object takes on the width of the under-placed object.

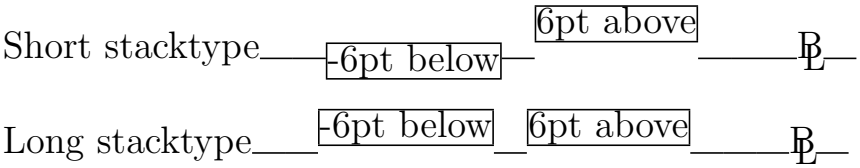
The command can be used with both stacktypes. With `\stacktype` "S",

the optional argument will denote the stackgap below the baseline (default `\Sstackgap`),

When used with `\stacktype` "L", the optional length denotes the distance to the baseline of the understacked object (default `\Lstackgap`).

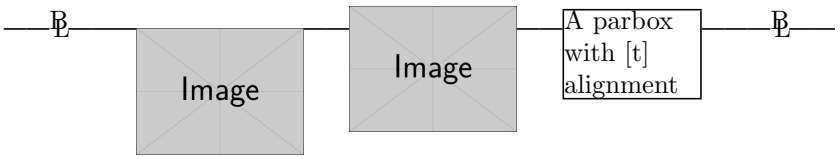


As the above images show, there is nothing to prevent the use of negative numbers being used to raise the the object above the baseline. But care must be taken, since a `\belowbaseline` with a negative shift will not necessarily equal the `\abovebaseline` with the corresponding positive shift (that equality only holds for long stacks). To better appreciate the meaning of negative baseline shifts, we compare, below, negative shifts for `\belowbaseline` to `\abovebaseline` with the corresponding positive argument.



The `\belowbaseline` command is also useful for forcing objects, such as images, to be top-aligned. This can be a useful feature when trying to achieve a desired horizontal alignment of disparate elements in tabular columns.

```
\def\stacktype{S}\belowbaseline[0pt]{\imgi}
\belowbaseline[-\ht\strutbox]{\imgi}:
```



7 Negative Stack Gap

A negative gap can be used to overlap things. For example, • plus "i" can make, when the stack gap is suitably negative, the following:

Note that the use of a negative stack gap for a two-item stack is actually very similar to the `\stackinset` macro described in §2.3.10 and §10. The issue of negative stackgaps is also discussed in §6.2.

8 The `\useanchorwidth` Option

Because • has width larger than the dot glyph itself (we box its extent here: ▭), stacking it could otherwise be a problem. For example, th i s uses the whole stackwidth to set the dotted-i in its place. On the other hand, when `\useanchorwidth` is defined as `T`, "this" uses only the anchor width to set the stack (where the "i" is the anchor).

Below, we show an `\fbox` around the dotted i, clearly demonstrating that the dot does not influence the width of the stack when `\useanchorwidth` is defined as "T".

9 Adding Gap Above and Below a Stack

By default, a stack is vertically clipped with no gap as in these two `\Longstack` examples: A a / G g

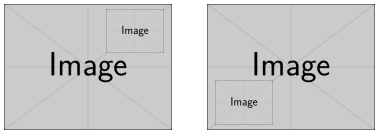
This package provides two ways to conveniently add gaps above and below a stack. For any type of stack (short or long), the `\addstackgap` macro (§2.3.11) adds a fixed amount of gap above and below a completed object. Alternately, during the construction of long stacks, the `\strutlongstacks` mode (§2.2) will automatically employ struts in each row of a long stack during its construction. However, these two approaches are not equivalent:

`\addstackgap[3pt]{}`: A a / G g `\strutlongstacks{T}`: A a / G g

Make sure you use the one that best applies to your situation.

10 Image Insets

The command `\stackinset` allows images to be stacked atop one another, to produce the effect of an inset image. In addition to specifying the two images, the effect can be tailored by specifying the vertical and horizontal offset of the inset image.



The last figure was obtained with `\stackinset{l}{3pt}{b}{2pt}{\imgii}{\imgi}`, showing that `\imgii` was offset 2pt from the bottom and 3pt from the left edge of `\imgi`.

The syntax of `\stackinset` allows for a convenient nesting of figure text labels. For example,

```
\stackinset{l}{ .1in}{b}{.2in}{Note 1}{%
\stackinset{r}{ .1in}{t}{.2in}{Note 2}{%
\stackinset{c}{-.5in}{c}{.4in}{Note 3}{%
\stackinset{r}{ .8in}{b}{.6in}{Note 4}{%
\stackinset{r}{ .1in}{c}{.0in}{\imgi}{%
\includegraphics[width=3in]{example-image}%
}}}}}
```

gives the following result:

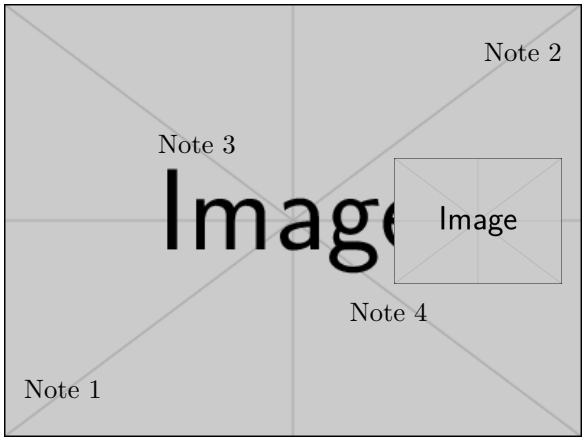


Figure 1. An example of `\stackinset`.

11 Stacking for Subfiguring and Baseline Manipulation

The stacking commands provide a convenient way to underset a subfigure notation under the corresponding image. Furthermore, because of the flexibility of the stacking commands, the baselines of the subfigures can be manipulated in many ways, which may come in handy depending on how you are choosing to lay out your subfigures.

Examples of this are shown in figure 2.

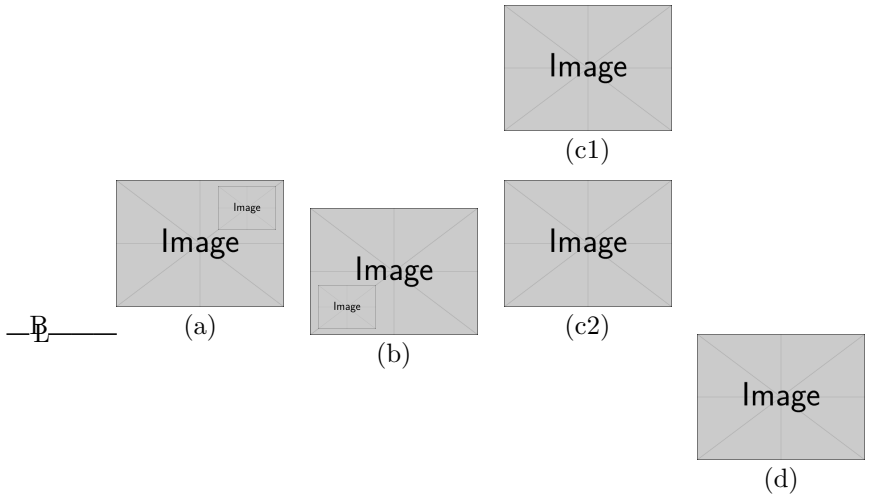


Figure 2. Image stacking with stacktype "S", stack gap set to 3pt and using:
(a) `\stackon{(a)}{\imga}`, (b) `\stackunder{\imgb}{(b)}`,
(c) `\Shortstack{{\imgi} (c1) {} {\imgi} (c2)}`, and
(d) `\belowbaseline[0pt]{\stackunder{\imgi}{(d)}}`.

12 Nested Stacking Commands

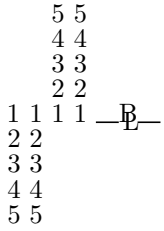
The macros `\Shortstack`, `\Longstack`, and the corresponding understack macros are set up to conveniently stack multiple objects using a space separated argument list. Because of parsing constraints on the space-separated list, however (see §13), there may be times that you would prefer building stacks by way of nesting the more primitive stacking macros.

Nesting can be accomplished, but some care must be taken, in the case of long stacks. But first, let us see the ways in which nesting may be applied for short

stacks:

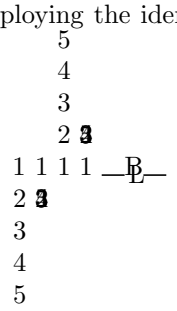
```
\stackunder{1}{\stackunder{2}{\stackunder{3}{\stackunder{4}{5}}}}
\stackunder{\stackunder{\stackunder{\stackunder{1}{2}}{3}}{4}}{5}
\stackon{1}{\stackon{2}{\stackon{3}{\stackon{4}{5}}}}
\stackon{\stackon{\stackon{\stackon{1}{2}}{3}}{4}}{5}
```

will produce the following:



In these cases, the nesting could be applied on either the first or the second argument of the `\stackunder` or `\stackon` commands, with the same end result.

In the case of long stacks however, employing the identical methodology gives a different, at first unexpected, result:



In this case, nesting the second argument works as hoped, but nesting the first argument does not. The reason for this "failure" stems not from a failure of logic, but an enforcement of it. It stems from the very definition of a long stack, in which the baseline of the second argument is placed `\LStackgap` below or above the baseline of the first argument. By nesting the first argument, all the second arguments are placed `\Lstackgap` from the unchanging stack baseline (thus overlapping), and not relative to the current top or bottom of the nested stack.

Therefore, to nest long stacks, only the second argument of the stackengine may be nested.

13 Math Mode and Mixed Mode Usage

As of V3.0, the stackengine package was extended to allow its arguments to be processed, by default, using inline math-mode (`\textstyle` math). The math-mode default is brought about with an invocation of `\stackMath`, which can be later reversed via `\stackText`.

Except in circumstances that require the particular features of this package, stackengine may not be the best package for math mode since there are many packages that already cater directly to the need to stack and align mathematical objects. However, if there is a need to use stackengine there, because of its particular facility with stacking gaps and alignment, there are several hints to remember.

Hints for use in math mode: (operating under `\stackMath`)

1. Arguments to stacking commands are only taken in math mode, by default, following the invocation of the `\stackMath` macro.
2. The `\stackMath` and `\stackText` macros are not to be used within stacking arguments, but are invoked prior to stacking, to set the default manner in which stacking arguments are subsequently processed.
3. As described in §2.3.3, if the math argument has spaces in a command which uses space-separated lists, or if the argument ends in a macro, the argument must be enclosed in braces.

   `\(y = \Shortstack{{a + b} {\odot} b} x\)`: $y = \Shortstack{{a + b} {\odot} b} x$
4. In math mode, the `\stackanchor` command seems to preserve the math axis better with an "S" `\stacktype`, rather than type "L".

Hints for using math in text mode: (operating under `\stackText`)

1. Stacking commands from this package can be used in math mode, but will set their arguments as text,

   `\( y = \stackunder{a}{+} x\)`: $y = \stackunder{a}{+} x$
2. To set an argument in math style, it must be set between math delimiters.

   `\( y = \stackunder{$a$}{+} x\)`: $y = \stackunder{$a$}{+} x$
3. To set a particular stack completely in math style, without globally setting the mode with a `\stackMath`, one can use `\ensurestackMath{}`.

4. If not using `\ensurestackMath`, each space-separated item of the argument to `\Shortstack`, `\Longstack`, and the corresponding understacks must each be enclosed in their own set of math delimiters (either `$...$` or `\(...\)`).

   `\(y = \Shortstack{$a$ . $b$} x\)`: $y = \overset{a}{\underset{.}{b}}x$

5. For more complex items, the `\savestack` command can be used to save an intermediate math expression (even if it isn't actually a stack).

   ```
   \savestack{\a}{\(\sqrt{a + b}\)}
   \savestack{\dash}{- - - - -}
   \(y = \stackunder{\stackon{\dash}{\a}}{$b$} x\):
   ```
   $y = \overset{\sqrt{a+b}}{\underset{b}{- - - - -}}x$

14 Deprecated Features

There are several definitions and macros from earlier versions of the stackengine package that are still fully functional, but are discouraged because of more preferable alternatives. They have been omitted from the current documentation, except for mention here. They are:

- `\def\usestackstrut{T/F}`. This definition is fully replaced by the mode setting given by the `\strutshortanchors{T/F}` macro. The new name is less ambiguous and the macro form is chosen vis-à-vis the `\def` form, because the macro represents a **mode** of stack construction, rather than an argument parameter.
- `\bottominset` and `\topinset`. These macros are subsumed by the much more versatile `\stackinset` macro. The replacement offers insets relative to not only the top and bottom, but also the mid-height of the anchor. It also provides the capability to horizontally place the inset relative to the center of the anchor, in addition to the left or the right. Most importantly, the placement of the anchor as the last argument of `\stackinset` means that the syntax for nested insets is much more readable vis-à-vis the nested syntax for `\bottominset` and `\topinset`.

15 Backward Compatibility

The author of this package profoundly apologizes for the fact that the syntax of this package changed between versions 1.0 and 2.0 of this package.

`\setstackgap` In version 1.0 of this package, stackgaps could be directly defined by setting the

LaTeX lengths `\Sstackgap` and `\Lstackgap`. The problem with this approach is that, under fontsize changes, these stackgaps were not automatically updated to reflect the new fontsize. As of V2.0, that deficiency was remedied. The syntax that replaces these length specifications is

`\setstackgap{S}{`*inter-item stackgap*`}`
`\setstackgap{L}{`*inter-item baselineskip*`}` .

Thus, `\Sstackgap` and `\Lstackgap` are no longer lengths (but instead `\def`s), and cannot be set via the `\Sstackgap=...` or `\setlength{\Sstackgap}{...}` syntax. It is these settings which will need to be revised to modernize older code.

`[oldsyntax]` While not encouraged, the V1.0 syntax may be recovered via

```
\usepackage[oldsyntax]{stackengine}
```

If you retain the "oldsyntax" setting, you must remember that stackengine will honor your specified gap sizes (`\Sstackgap` and `\Lstackgap`) at the time of their **definition**, rather than at the time of their **invocation**. Thus, if `\Sstackgap` is specified as 0.7ex when the text is `\normalsize`, the gap length is set in a `\normalsize` font, even if the fontsize is later changed. To overcome this behavior, the stackgap must be redefined at the new fontsize, or better still, `[oldsyntax]` usage needs to be relegated.

16 Acknowledgements

I would like to thank Prof. Enrico Gregorio at tex.stackexchange for assisting in numerous ways to make this package better:

- by providing three lines of code to strip a leading backslash from an argument:
  `http://tex.stackexchange.com/questions/42318/removing-a-backslash-from-a-character-sequence`
- in making the transition from the deficient V1.0 syntax as painless as possible:
  `http://tex.stackexchange.com/questions/123443/defining-a-length-that-scales-with-fontsize-changes/123470#123470`
- and in helping to make the parsing routines of this package robust:
  `http://tex.stackexchange.com/questions/137298/centering-breaks-fragile-command-is-there-a-fix/137314#137314`

Thanks also to Dr. David Carlisle for explaining the need for argument expansion in certain situations:
`http://tex.stackexchange.com/questions/131581/parsing-rows-and-tab-characters/131594#131594`

Finally, I extend my thanks to Per Starbäck, for his unsolicited but gratefully accepted edit of the V3.21 package documentation, which corrected numerous typographical errors.

17 Sectional index of **stackengine** definitions, modes, & macros

Definition, Mode, or Macro	§ Defined	§ Used
\abovebaseline	2.3.9	6.2
\addstackgap	2.3.11	2.3.2, 9
\bclap	2.3.5	5
\belowbaseline	2.3.9	6.2
\bllap	2.3.5	5
\bottomlap	2.3.5	5
\braceVectorstack	2.3.7	
\bracketVectorstack	2.3.7	
\brlap	2.3.5	5
\Centerstack	2.3.7	2.3.6
\ensurestackMath	2.3.8	2.2, 13
\hsmash	2.3.12	
\Longstack	2.3.3	1.1, 1.3, 2.2, 2.3.4, 2.3.7, 3.2, 4, 9, 12, 13
\Longunderstack	2.3.4	1.3, 2.2, 4, 12, 13
\Lstackgap	2.1	1.1, 2.3.1, 6.2, 12, 15
\parenVectorstack	2.3.7	
\quietstack	2.1	2.3.1, 3.1
\savestack	2.3.13	2.3.6, 3.2, 4, 13
\setstackEOL	2.2	2.3.3
\Shortstack	2.3.3	1.1, 1.3, 1.4, 1.5, 2.2, 2.3.4, 6.1, 11, 12, 13
\Shortunderstack	2.3.4	1.1, 1.3, 1.4, 2.2, 2.3.3, 12, 13
\Sstackgap	2.1	1.1, 2.3.1, 2.3.11, 6.2, 15
\stackalignment	2.1	1.5, 2.3.1, 4, 5
\stackanchor	2.3.6	2.3.7, 2.3.10, 6.1, 13
\stackengine	2.3.1	2.3.2
\stackgap	2.1	2.3.1, 2.3.3
\stackinset	2.3.10	7, 10, 14
\stackMath	2.2	2.3.3, 2.3.8, 13
\stackon	2.3.2	2.3.11, 7, 8, 11, 12, 13
\stackText	2.2	13
\stacktype	2.1	1.1, 2.3.1, 2.3.3, 6.1, 6.2, 11, 13
\stackunder	2.3.2	11, 12, 13
\strutlongstacks	2.2	2.3.7, 2.3.11, 9
\strutshortanchors	2.2	2.3.6, 6.1, 14
\tclap	2.3.5	5
\tllap	2.3.5	5
\toplap	2.3.5	5
\trlap	2.3.5	5
\useanchorwidth	2.1	2.3.1, 6.2, 8
\Vectorstack	2.3.7	
\vertVectorstack	2.3.7	

18 Code Listing

```
\def\stackengineversionnumber{v3.26}
\ProvidesPackage{stackengine}
[2016/07/28 \stackengineversionnumber\
 Stacking text and objects in convenient ways]
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Steven B. Segletes.
%
% V1.0  -Initial release.
% V2.0  -Revised syntax, making \Sstackgap=len, \Lstackgap=len syntax
%        obsolete.  New syntax is \setstackgap{S}{len} and
%        \setstackgap{L}{len}
%       -Condensed several routines
%       -Revised documentation for clarity and to eliminate typos.
%       -Added \toplap and \bottomlap.
%       -Disallowed \quietstack for lapping.
%       -added \abovebaseline
% V3.0  -Added \stackMath and \stackText macros to treat arguments,
%        by default, in math mode or not.
% V3.1  -Added \hsmash, for zero-width, center-aligned argument (note:
%        other alignments can be achieved with \llap and \rlap)
%       -Eliminated need for \usepackage{readarray} through the
%        implementation of \@readMANYrows and \@processROW
%       -Added [usestackEOL] package option to use \\ separator, rather
%        than space separator in \Shortstack, \Longstack,
%        \Shortunderstack, and \Longunderstack
%       -Implemented \setstackEOL to define new separator in \Shortstack,
%        \Longstack, \Shortunderstack, and \Longunderstack
%       -Renamed some package internal variables with @...
%       -Added \addstackgap to place vertical gap above and below an
%        existing stack.  Negative gaps have no effect.
%       -Added \stackinset, deprecates \topinset & \bottominset.
%       -allow c \stackalignment on \stackinset, \topinset, and
%        \bottominset to have meaning (rightward offset relative to
%        centerline, can be positive or negative).
%       -Forced \useanchorwidth{T} for insets, since otherwise the
%        stackwidth could vary for reasons not obvious to user.
%       -Added \strutlongstacks{} mode for auto-adding a strut to
%        each row of a long stack
%       -Recommend \strutshortanchors{} mode in preference to
%        \def\usestackstrut{} definition (now deprecated).
% V3.11 -Corrected bug in \stackinset which did not condition the
%        anchor when using vertical centering in \stackMath mode
% V3.2  -Incorporated changes suggested by E. Gregorio to make robust
%        the supporting macro \@readMANYrows
%       -Introduced routines: \Centerstack, \Vectorstack, \parenVectorstack
```

```
%        \bracketVectorstack, \braceVectorstack, \vertvectorstack,
%        and \ensurestackMath
% V3.21 -Corrected a bug in \stackinset introduced in V3.11
% V3.22 -Removed \usepackage{calc}
%       -Corrected \@STRT placement to not inadvertantly force a \mathop
%        on rows beginning with a minus sign.
% V3.23 -Made EOL-separated stacks (i.e., \Longstack, \Shortstack,
%        \Centerstack, \Vectorstack, etc.) work properly when a single
%        (unstacked) row is input.
%       -Noted in documentation that while EOL-separated stacks
%        ignore leading/trailing spaces in their arguments (unless the
%        EOL itself is a space token), the \stackon / \stackunder
%        family of macros do not ignore spaces in their arguments.
%       -Reinstated \usepackage{calc} (Oops!)
% V3.24 -Replaced references to \roman with \romannumeral, since \roman
%        is redefined by packages like spanish babel.
% V3.25 -Default longstack gap is now \normalbaselineskip, so as to
%        automatically work in tabular environments.
% V3.26 -V3.25 on partially fixed the problem of \normalbaselineskip.
\RequirePackage{etoolbox}
\usepackage{calc}

\newtoggle{@doneROWreads}
\newtoggle{stackloop@done}
\newcounter{@stackindex}
\newlength\@boxshift
\newlength\stack@tmplength
\newlength\temp@stkl
\global\newlength\@stackedboxwidth
\newsavebox\@addedbox
\newsavebox\@anchorbox
\newsavebox\stackedbox
\newsavebox{\@centerbox}
\newcounter{ROWcellindex@}

%% Following 3 lines thanks to Prof. Enrico Gregorio, from:
%% http://tex.stackexchange.com/questions/42318/
\begingroup\lccode`\|=`\\
\lowercase{\endgroup\def\removebs#1{\if#1|\else#1\fi}}
\newcommand{\stack@macro@name}[1]{\expandafter\removebs\string#1}

%% Following 3 macros were, in V3.2, made robust with the help of
%% Prof. Enrico Gregorio.  See:
%% http://tex.stackexchange.com/questions/137298/
\newcommand\setstackEOL[1]{%
  \ifstrempty{#1}{\def\SEP@char{ }}{\def\SEP@char{#1}}%
  \expandafter\define@processROW\expandafter{\SEP@char}%
}
\newcommand\define@processROW[1]{%
    \def\@processROW##1#1##2||{%
      \def\@preSEP{##1}%
      \def\@postSEP{##2}%
    }%
}
% FOR PROCESSING A GROUP OF STACK ROWS SEPARATED BY \SEP@char
\newcommand\@readMANYrows[1]{%
  \togglefalse{@doneROWreads}%
```

```
  \edef\@postSEP{\unexpanded{#1}\expandonce{\SEP@char}}%
  \setcounter{ROWcellindex@}{0}%
  \whileboolexpr{test {\nottoggle{@doneROWreads}}}{%
    \stepcounter{ROWcellindex@}%
      \expandafter\@processROW\@postSEP||%
    \expandafter\ifstrempty\expandafter{\@postSEP}{%
      \toggletrue{@doneROWreads}%
    }{}%
    \csedef{arg\romannumeral\theROWcellindex@}{\expandonce{\@preSEP}}%
  }%
% \narg GIVES HOW MANY ROWS WERE PROCESSED
  \xdef\narg{\arabic{ROWcellindex@}}%
}
%%%%%

% PROCESS PACKAGE OPTIONS
\newif\ifstackengine@oldsyntax
\newif\ifstackengine@usestackEOL
\DeclareOption{oldsyntax}{\stackengine@oldsyntaxtrue}
\DeclareOption{usestackEOL}{\stackengine@usestackEOLtrue}
\ProcessOptions\relax
\ifstackengine@oldsyntax%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% THIS VERSION 1.0 SYNTAX WAS PHASED OUT %%%%%%%%%%%%%%%%%%%
%%%%% (but can invoke with \usepackage[oldsyntax]{stackengine} )
\newlength\Sstackgap\newlength\Lstackgap
\newcommand\stackgap{\if S\stacktype\the\Sstackgap\else\the\Lstackgap\fi}
\global\setlength{\Sstackgap}{3pt}
\global\setlength{\Lstackgap}{\normalbaselineskip}
\else%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% AND REPLACED WITH THE FOLLOWING %%%%%%%%%%%%%%%%%%%%%%%%%
% Again, thanks to Prof. Gregorio for his lucid explanation that helped
% improve this package.  See:
% http://tex.stackexchange.com/questions/123443/
% defining-a-length-that-scales-with-fontsize-changes/123470#123470
\newcommand{\setstackgap}[2]{\@namedef{#1stackgap}{#2}}
\newcommand\stackgap{\@nameuse{\if S\stacktype S\else L\fi stackgap}\relax}
\setstackgap{S}{3pt}% SHORTSTACKING GAP BETWEEN ITEMS (SAME AS \shortstack)
\setstackgap{L}{\normalbaselineskip}% LONGSTACKING GAP BETWEEN ITEMS (def. \normalbaselineskip)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\fi
\ifstackengine@usestackEOL%
% USE \\ AS SEPERATOR IN \Shortstack, \Longstack,
% \Shortunderstack, and \Longunderstack
  \setstackEOL{\\}
\else
% USE SPACE TOKEN AS SEPERATOR IN \Shortstack, \Longstack,
% \Shortunderstack, and \Longunderstack
  \setstackEOL{ }
\fi

%%%%% PACKAGE DEFAULT DEFINITIONS
\def\useanchorwidth{F}% T= FORCE BOX WIDTH TO ANCHOR-WIDTH
\def\quietstack{F}% T = SUPPRESS OUTPUT OF ALL COMMANDS
%                   (USE \usebox{\stackedbox} TO RETRIEVE)
\def\stackalignment{c}% l, c, or r for left, center or right alignment
```

```
\def\stacktype{S}% S for constant gap shortstack;
                 % L for constant baselineskip longstack

% DEPRECATED USAGE \def\usestackstrut{}; INSTEAD USE \strutshortanchors{}
%\def\usestackstrut{T}

%%%%% MODES
\newcommand\stackMath{\global\def\stack@delim{$}}
\newcommand\stackText{\global\def\stack@delim{}}
\stackText

\newcommand\strutlongstacks[1]{\def\@strutlongstacks{#1}}
\strutlongstacks{F}

\newcommand\strutshortanchors[1]{\def\usestackstrut{#1}}
\strutshortanchors{T}

%%%%% MACROS

%\stackengine{\Sstackgap or \Lstackgap or \stackgap or stacklength}
%            {anchor}
%            {item}
%            {O or U}
%            {\stackalignment or l or c or r}
%            {\quietstack or T or F}
%            {\useanchorwidth or T or F}
%            {\stacktype or S or L}
\newcommand*\stackengine[8]{%
  \def\@STRT{}%
  \if T\@strutlongstacks\if L#8\def\@STRT{\strut}\fi\fi%
  \sbox{\stackedbox}{%
    \sbox{\@anchorbox}{\@STRT\stack@delim#2\stack@delim}%
    \sbox{\@addedbox}{\@STRT\stack@delim#3\stack@delim}%
    \settowidth{\@stackedboxwidth}{\@STRT\stack@delim#2\stack@delim}%
    \if F#7%
      \ifdim\wd\@addedbox>\@stackedboxwidth%
        \settowidth{\@stackedboxwidth}{\stack@delim\@STRT#3\stack@delim}%
      \fi%
    \fi%
    \setlength{\@boxshift}{#1}%
    \if L#8%
      \if U#4%
        \setlength{\@boxshift}{-\@boxshift}%
      \fi%
    \else%
      \if U#4%
        \setlength{\@boxshift}{-\dp\@anchorbox -\ht\@addedbox -\@boxshift}%
      \else%
        \setlength{\@boxshift}{\ht\@anchorbox +\dp\@addedbox +\@boxshift}%
      \fi%
    \fi%
    \if c#5%
      \hspace{.5\@stackedboxwidth}%
      \hspace{-.5\wd\@anchorbox}%
      \usebox{\@anchorbox}%
      \hspace{-.5\wd\@anchorbox}%
      \hspace{-.5\wd\@addedbox}%
```

```
      \raisebox{\@boxshift}{\@STRT\stack@delim#3\stack@delim}%
      \hspace{-.5\wd\@addedbox}%
      \hspace{.5\@stackedboxwidth}%
    \else%
      \if l#5%
        \usebox{\@anchorbox}%
        \hspace{-\wd\@anchorbox}%
        \raisebox{\@boxshift}{\@STRT\stack@delim#3\stack@delim}%
        \hspace{-\wd\@addedbox}%
        \hspace{\@stackedboxwidth}%
      \else%
        \if r#5%
          \hspace{\@stackedboxwidth}%
          \hspace{-\wd\@anchorbox}%
          \usebox{\@anchorbox}%
          \hspace{-\wd\@addedbox}%
          \raisebox{\@boxshift}{\@STRT\stack@delim#3\stack@delim}%
        \fi%
      \fi%
    \fi%
  }%
  \if F#6\usebox{\stackedbox}\fi%
}

\newcommand*\stackunder[3][\stackgap]{\stackengine%
  {#1}{#2}{#3}{U}{\stackalignment}{\quietstack}{\useanchorwidth}{\stacktype}}

\newcommand*\stackon[3][\stackgap]{\stackengine%
  {#1}{#2}{#3}{O}{\stackalignment}{\quietstack}{\useanchorwidth}{\stacktype}}

\newcommand*\Shortstack[2][\stackalignment]{\@stack{#1}{#2}{O}{S}}

\newcommand*\Longstack[2][\stackalignment]{\@stack{#1}{#2}{O}{L}}

\newcommand*\Shortunderstack[2][\stackalignment]{\@stack{#1}{#2}{U}{S}}

\newcommand*\Longunderstack[2][\stackalignment]{\@stack{#1}{#2}{U}{L}}

\newcommand*\@stack[4]{%
  \@readMANYrows{#2}%
  \if U#3%
    \setcounter{@stackindex}{\narg}%
    \def\@stacksign{-}%
    \def\@stackcondition{\the@stackindex > 1}%
  \else%
    \setcounter{@stackindex}{1}%
    \def\@stacksign{}%
    \def\@stackcondition{\the@stackindex < \narg}%
  \fi%
  \def\@STRT{}%
  \if T\@strutlongstacks\if L#4\def\@STRT{\strut}\fi\fi%
  \sbox{\stackedbox}{%
    \@STRT\stack@delim\ignorespaces%
      \csname arg\romannumeral\the@stackindex\endcsname%
    \unskip\stack@delim}%
  \ifnum\narg>1%
    \togglefalse{stackloop@done}%
```

```
  \else%
    \toggletrue{stackloop@done}%
  \fi%
  \whileboolexpr{test {\nottoggle{stackloop@done}}}{%
    \addtocounter{@stackindex}{\@stacksign 1}%
    \stackengine{\csname#4stackgap\endcsname}%
      {\ignorespaces\csname arg\romannumeral\the@stackindex\endcsname\unskip}%
      {\usebox{\stackedbox}}{#3}{#1}{T}{\useanchorwidth}{#4}%
    \ifnum\@stackcondition\else\toggletrue{stackloop@done}\fi%
  }%
  \if F\quietstack\usebox{\stackedbox}\fi%
}

% PERHAPS THESE SIX MACROS CAN BE REPLACED BY \toplap AND \bottomlap
\newcommand*\tllap[2][\Lstackgap]{\@stacklap{#1}{l}{#2}{O}}
\newcommand*\tclap[2][\Lstackgap]{\@stacklap{#1}{c}{#2}{O}}
\newcommand*\trlap[2][\Lstackgap]{\@stacklap{#1}{r}{#2}{O}}
\newcommand*\bllap[2][\Lstackgap]{\@stacklap{#1}{l}{#2}{U}}
\newcommand*\bclap[2][\Lstackgap]{\@stacklap{#1}{c}{#2}{U}}
\newcommand*\brlap[2][\Lstackgap]{\@stacklap{#1}{r}{#2}{U}}

\newcommand*\toplap[3][\Lstackgap]{\@stacklap{#1}{#2}{#3}{O}}
\newcommand*\bottomlap[3][\Lstackgap]{\@stacklap{#1}{#2}{#3}{U}}

\newcommand*\@stacklap[4]{%
  \if l#2\stackengine{#1}{}{#3}{#4}{r}{F}{T}{L}\else%
    \if r#2\stackengine{#1}{}{#3}{#4}{l}{F}{T}{L}\else%
      \stackengine{#1}{}{#3}{#4}{c}{F}{T}{L}%
    \fi%
  \fi
}

\newcommand*\hsmash[1]{\stackengine{0pt}{}{#1}{O}{c}{F}{T}{L}}

\newcommand*\stackanchor[3][\stackgap]{%
  \setlength{\stack@tmplength}{#1}%
  \setlength{\stack@tmplength}{.5\stack@tmplength}%
  \if T\usestackstrut%
    \if S\stacktype\addtolength{\stack@tmplength}%
      {.5\ht\strutbox-.5\dp\strutbox}\fi%
  \fi%
  \stackengine{\stack@tmplength}{}{#2}{O}{\stackalignment}{T}{F}{\stacktype}%
  \if T\usestackstrut%
    \if S\stacktype\addtolength{\stack@tmplength}%
      {-\ht\strutbox+\dp\strutbox}\fi%
  \fi%
  \stackengine{\stack@tmplength}{\usebox{\stackedbox}}{#3}{U}%
              {\stackalignment}{\quietstack}{F}{\stacktype}%
}

\newcommand*\abovebaseline[2][\stackgap]{%
  \stackengine{#1}{}{#2}{O}{\stackalignment}{\quietstack}{F}{\stacktype}}

\newcommand*\belowbaseline[2][\stackgap]{%
  \stackengine{#1}{}{#2}{U}{\stackalignment}{\quietstack}{F}{\stacktype}}

\newcommand*\savestack[2]{%
```

```
  \global\edef\sv@name{\stack@macro@name{#1}}%
  \@ifundefined{\sv@name content}{%
    \expandafter\newsavebox\expandafter{\csname\sv@name content\endcsname}%
   }{}%
  \expandafter\sbox\csname\sv@name content\endcsname{#2}%
  \protected@edef#1{\usebox{\csname\sv@name content\endcsname}}%
}

\newcommand*\addstackgap[2][\Sstackgap]{\stackengine{#1}{%
  \stackengine{#1}{#2}{}{O}{c}{\quietstack}{T}{S}}{}{U}{c}{\quietstack}{T}{S}}

\newcommand*\topinset[4]{\stackinset{\stackalignment}{#4}{t}{#3}{#1}{#2}}

\newcommand*\bottominset[4]{\stackinset{\stackalignment}{#4}{b}{#3}{#1}{#2}}

%\stackinset{l/c/r}{x}{b/c/t}{y}{inset}{anchor}
\newcommand*\stackinset[6]{%
  \def\conditioned@inset{\stack@delim#5\stack@delim}%
  \def\conditioned@anchor{\stack@delim#6\stack@delim}%
  \ifstrequal{#4}{}%
    {\setlength{\stack@tmplength}{0pt}}%
    {\setlength{\stack@tmplength}{#4}}%
  \if c#3%
    \setlength{\temp@stkl}{%
      \heightof{\conditioned@anchor}+\depthof{\conditioned@anchor}%
      -\heightof{\conditioned@inset}-\depthof{\conditioned@inset}}%
    \addtolength{\stack@tmplength}{.5\temp@stkl}%
  \fi%
  \ifstrequal{#2}{}{\def\stack@tmp{0pt}}{\def\stack@tmp{#2}}%
  \def\stack@lroffset{\rule{\stack@tmp}{0pt}}%
  \addtolength{\stack@tmplength}{%
    \heightof{\conditioned@inset}+\depthof{\conditioned@inset}}%
  \if c#1%
    \def\conditioned@inset{\stack@lroffset\stack@lroffset#5}%
  \else%
    \def\conditioned@inset{\stack@lroffset#5\stack@lroffset}%
  \fi%
  \stackengine{-\stack@tmplength}{#6}{\conditioned@inset}%
              {\inset@valign{#3}}{#1}{\quietstack}{T}{S}%
}

\def\inset@valign#1{\if t#1O\else U\fi}

\newcommand\Centerstack[2][\stackalignment]{%
  \sbox{\@centerbox}{\strutlongstacks{T}\Longstack[#1]{#2}}%
  \raisebox{-.5\ht\@centerbox+.5\ht\strutbox}{\usebox{\@centerbox}}}

\newcommand\Vectorstack[2][\stackalignment]{%
  \ensurestackMath{\Centerstack[#1]{#2}}}

\newcommand\parenVectorstack[2][\stackalignment]{%
  \ensurestackMath{\left(\Centerstack[#1]{#2}\right)}}

\newcommand\bracketVectorstack[2][\stackalignment]{%
  \ensurestackMath{\left[\Centerstack[#1]{#2}\right]}}

\newcommand\braceVectorstack[2][\stackalignment]{%
```

```
  \ensurestackMath{\left\{\Centerstack[#1]{#2}\right\}}}

\newcommand\vertVectorstack[2][\stackalignment]{%
  \ensurestackMath{\left|\Centerstack[#1]{#2}\right|}}

\newcommand\ensurestackMath[1]{%
  \let\sv@stackmode\stack@delim\stackMath%
  #1\global\let\stack@delim\sv@stackmode}

\endinput
```