The tabstackengine Package

Front-end to the stackengine package, allowing tabbed stacking

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

February 19, 2014
V1.10

Contents

1 Introduction

The tabstackengine package provides a front end to the stackengine package that allows for the use of tabbing characters within the stacking arguments. **Familiarity with the syntax of the stackengine package is assumed.** When invoked, tabstackengine loads the stackengine package with the `[usestackEOL]` option set, so that the end-of-line (EOL) character in certain stacking arguments will be taken, by default, as \\, rather than a space (which is the default EOL separator in stackengine).

With tabstackengine, command variations are introduced to allow several variants of tabbing within the macro arguments. The default tabbing character is

the ampersand (&); however, the tabbing character can be reset to other values.

In most cases (where it makes sense), a stackengine macro name may be prepended with the word tabbed, align, or tabular to create a new tabstackengine macro that allows for tabbed arguments.

2 Tabbing Variations within tabstackengine

The tabstackengine package syntax allows three types of tabbing variation denoted by the words tabbed, align, and tabular in the macro name itself. In the case of tabbed macros, the tabbed columns all share the same alignment, as dictated by the \stackalignment setting or perhaps provided as an optional argument in some macro forms.

In the case of align macros, the alignment in columns is alternately specified as right, then left, *etc.*, in the manner of the align environment of the amsmath package.

Finally, in the case of tabular macros, an extra argument is passed to the macro that specifies the left-center-right alignment for each individual column, in the manner of {lccr}.

3 Column Spacing within tabstackengine

\fixTABwidth

Intercolumn space can be introduced to tabstackengine output in one of two ways. First, there is a macro setting to force all columns to be the same width (namely, the width of the widest entry in the stack), using the syntax \fixTABwidth{T *or* F}. When set true, column space will be introduced to all but the widest column of a stack, so as to make all columns of a width equal to that of the widest column.

\setstacktabbedgap

Secondly, each of the tabbing variations has the means to introduce a fixed amount of space between columns. By default, the tabbed stacking macros add no space (0pt) between adjacent columns, but this value can be reset with the macro \setstacktabbedgap{*length*}.

\setstackalignngap

In the case of the align stacking macros, there is never any gap introduced after the right-aligned columns. However, the default gap introduced after the left-aligned columns is, by default, 1em (the same gap as \quad). It can be reset with the macro \setstackalignngap{*length*}.

\setstacktabulargap

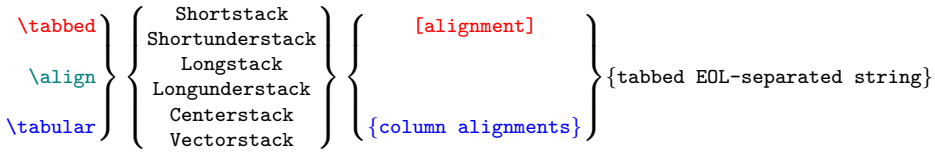
For the tabular stacks, the default intercolumn gap is the value of \tabcolsep.

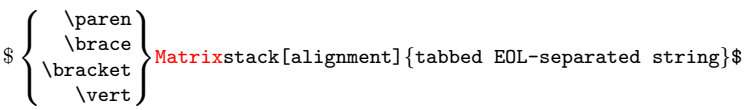
The default value may be reset with the macro \setstacktabulargap{*length*}.

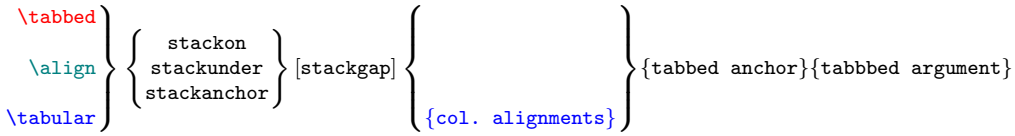
Note that these \setstack...gap macros are for setting horizontal gaps between columns of a stack. They should not be confused with the \setstackgap macro of stackengine that sets the vertical gap for long and short stacks.

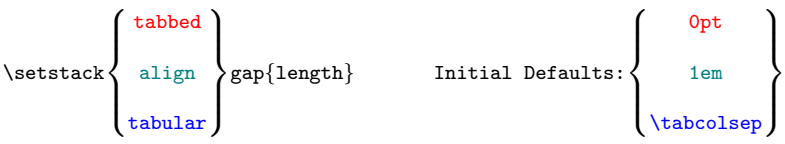
4 Command Summary

Below are the new commands introduced by this package. When there are multiple commands delimited by braces, any one of the commands within the brace may be selected.










```
\fixTABwidth{T or F}
\setstackTAB{tabbing character}
\TABunaryLeft    (\TABbinaryRight)
\TABunaryRight   (\TABbinaryLeft)
\TABbinary
```

The following macros are macros that can be used for parsing tabbed data outside of a TABstack.

```
\readTABrow{row ID}{tab-separated string}
\TABcell{row ID}{column number}
\TABcells{row ID}
```

```
\TABstrut{row ID}
\TABstackMath
\TABstackText
\ensureTABstackMath{}
```

4.1 Command Examples

Below we give examples of the various types of commands made available through the stackengine package.

Tabbed End-of-Line (EOL)-delimited Stacks

Here, the optional argument `[l]` defines the alignment of *all* the columns. The default alignment is `[c]`.

```
\tabbedShortunderstack[l]{A&B&CCC\\aaa&bbb&c\\1111&2&3}
```

A B CCC
aaa bbbc
11112 3

Align End-of-Line (EOL)-delimited Stacks

In an `align`-stack, the column alignments will always be `rlrl...` The gap following the left-aligned columns is set by `\setstackalignngap`.

```
\stackMath$Z:\left\{\alignCenterstack{%
  y=&mx+b,&0=&Ax+By+C\\y_1=&W_1,&y_2=&W_2}\right.$
```

$$Z:\left\{\begin{array}{ll} y=mx+b, & 0=Ax+By+C \\ y_1=W_1, & y_2=W_2 \end{array}\right.$$

Tabular End-of-Line (EOL)-delimited Stacks

In a `tabular`-stack, the alignment of each column is specified in a separate leading argument.

```
\stackText\tabularLongstack{rllc}{%
  9)& $y_1=mx+b$ &linear&*\\10)& $y_2=e^x$ &exponential&[23]}
```

9) $y_1=mx+b$ linear *
10) $y_2=e^x$ exponential [23]

Matrix Stack

The `Matrix`-stacks are a tabbed variant of stackengine's `Vector`-stacks.

```
\setstacktabbedgap{1.5ex}
$I = \bracketMatrixstack{1&0&0\\0&1&0\\0&0&1}$
```

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tabbed Stack

This variant of a `tabbed`-stack stacks exactly two items. Its arguments don't need to be protected. The optional argument is a stacking gap, as in the syntax of the stackengine package.

```
\setstacktabbedgap{1ex}
\tabbedstackon[4pt]{Jack&drove&the car&home.}{SN&V&DO&IO}
```

SN	V	DO	IO
Jack	drove	the car	home.

Align Stack

This is for stacking two items with `rlrl...` alignment pattern.

```
\stackMath\setstackalignngap{3em}
\alignstackunder[10pt]{y=&mx+b,&0=&Ax+By+C}{y_1=&W_1,&y_2=&W_2}
```

$$\begin{aligned} y &= mx + b, & 0 &= Ax + By + C \\ y_1 &= W_1, & y_2 &= W_2 \end{aligned}$$

Tabular Stack

This is for stacking two items with specifiable alignment pattern.

```
\stackMath\setstacktabulargap{1ex}\tabularstackanchor[4pt]{rcl}%
  {\bullet\bullet\bullet\bullet&\belowbaseline[-2pt]{\triangle}%
    &\bullet\bullet\bullet\bullet}%
  {1 + 3(4-3) & = & 7 - 6/2}
```

$$\begin{array}{rcl} \bullet\bullet\bullet & \triangle & \bullet\bullet\bullet \\ 1 + 2(4 - 3) & = & 6 - 6/2 \end{array}$$

Fixed Tab Width (equal width columns, based on largest)

With this mode set, the stack will have fixed-width columns, based on the overall widest entry.

```
\fixTABwidth{T}\setstacktabbedgap{1ex}%
$\left(\tabbedCenterstack[r]{1&34&544\\4324329&0&8\\89&123&1}\right)$
```

$$\left(\begin{array}{rrr} 1 & 34 & 544 \\ 4324329 & 0 & 8 \\ 89 & 123 & 1 \end{array}\right)$$

Setting the Stack Tabbing Character

By default, for the parsing of columns within a row, this package employs the `&` character to delimit the columns. This value can be changed via `\setstackTAB{}`, where the argument is the newly desired tabbing character. It can be any of various characters, including a space token, if one wishes to use a space-separated list to parse the columns. Generally, it may need to be changed if you are building a TABstack inside of an environment that already uses the `&` character as a tab, such as the `tabular` environment for text or the various math environments in the amsmath package.

Unary versus Binary Operators/Relations at Cell Ends

There are two things to keep in mind regarding TABstacked content. First, a TABstack cell has no precise understanding up what content precedes it in the cell to the immediate left, nor what content follows it in the cell to the immediate right. It does, however know the overall height/depth of the content

across the whole row and creates a vertical "strut" of that height and depth, which must, in some way, be applied to every cell in the row.

This vertical strut can be applied to the cell immediately prior to or immediately following the cell content, as we shall see. However, such an action will have an effect on math operators and relations found at the leading or trailing ends of the cell content.

Math operators and relations can be categorized as unary or binary; some may be both, depending on their usage context, such as the minus sign. When used as $a - b$, the relation is binary, because it connects a and b in a mathematical relationship. Note how space appears both before and after the minus sign. Alternatively, when used as $-\pi$, the minus sign operates only upon what follows, in this case π, to produce a negative. Note how no space is introduced between the minus sign and π.

Because a TABstack cell has no intimate knowledge of the adjacent cell content, it is up to the user to employ his tabbing seperators in a way that produces the desired result. By default, tabstackengine will place the strut after the cell content. This means that any trailing math operator in a cell will present itself in its binary form (regardless of what comes in the cell to the right), because the strut will appear as trailing data against which the operator can be set. Similarly, any leading math operator will present itself as unary (regardless of what content appears in the cell to the left).

Thus, `\tabbedLongstack{y =&-mx +& b}` will present as $y = -mx + b$, by default, with the trailing equal and plus signs as binary, and the leading minus sign as unary. The package can reverse the default with the following declarative modes: `\TABunaryRight` (identical to `\TABbinaryLeft`); as well as `\TABbinary`, which will present both leading *and* trailing operators in their binary form. The default can be restored with `\TABunaryLeft` (identical to `\TABbinaryRight`).

Without changing any of the package strut modes, an operator, such as minus, can be forced into its unary mode by enclosing it in braces: `{-}`. Likewise, it can be forced into its binary mode by placing empty braces on both sides of it: `{}-{}`.

The Macros `\readTABrow`, `\TABcell(s)`, and `\TABstrut`

One of the nice features of tabstackengine is that it gives you access to its parser, which can be used for tasks unrelated to building stacks. It provides several macros, including `\readTABrow{`*ident*`}{`*tab-separated-string*`}` for digesting a list and assigning it a global ID, in this case, "*ident*". It also provides `\TABcells{`*ident*`}` to tell you how many items are in the list named "*ident*", as well as the macro `\TABcell{`*ident*`}{`*column number*`}` to provide the value of the

specified column from the ID'ed row. For example,

```
\setstackTAB{,}
\TABstackMath
\readTABrow{myident}{1,2,   3x , 4, 5,x^2,,g,\textbf{bold 9}}
\TABcells{myident} items in dataset ``myident''\\
The 6th data item is \TABcell{myident}{6}\\
The 9th data item is \TABcell{myident}{9}
```

will yield the following result:

9 items in dataset "myident"
The 6th data item is x^2
The 9th data item is **bold 9**

The result provided by `\TABcell{`*ident*`}{`*column number*`}` has been processed in the following way: 1) leading and trailing spaces have been removed from the column's data, and 2) the column will have been processed in math mode if `\TABstackMath` (see below) had been in force. Thus, placing the cell contents in a box, `\fbox{\TABcell{myident}{3}}` is $\boxed{3x}$.

If one, however, wished to access the raw data, with leading/trailing spaces still intact (and ignoring the `\TABstackMath` setting), one can compose the macro, `\csname TABX`*ident*`X\romannumeral` *column number*`\endcsname`, such that `\fbox{\csname TABXmyidentX\romannumeral 3\endcsname}`, using the above example, would give $\boxed{\text{3x }}$.

Finally, the macro `\TABstrut{myident}` provides a strut that spans the vertical height of the complete row content. Here is `\TABstrut{myident}` (that has been boxed with `\fboxsep=0pt`) compared to the boxed content of the row itself: | $\boxed{123x45x^2g\textbf{bold 9}}$. As you can see, the `\TABstrut{myident}` captures the vertical extent of the complete row data, including the descending g and the superscript of the x^2. The `\TABstrut{}`, when paired with the column content, is useful when one wishes to make all column entries of a row display with equal height and depth, even when their native height varies from column to column.

`\TABstackMath`, `\ensureTABstackMath`, and `\TABstackText`

These macros are *not* needed when building stacks in tabstackengine. When constructing TABstacks, their presence is superfluous because their companion macros, `\stackMath`, `\ensurestackMath`, and `\stackText`, will carry over from the stackengine package.

However, there is one application where their use is required. And that is

when you use the facilities of this package *not* to build TABstacks, but *instead* use it to parse data with `\readTABrow{}{}` and present it with `\TABcell{}{}`, respectively. Notably `\TABcell{}{}` does not access the stackengine setting for the status of `\stackMath` and therefore pays attention only to the current setting specified by `\TABstackMath`.

One should note, however, that TABstacks pay attention to both settings. Therefore, **if EITHER `\stackMath` or `\TABstackMath` have been invoked, the TABstack argument will be processed by `\TABcell` in math mode**. This is a good reason to avoid the "TAB" versions of these macros, unless employing them to parse data (and then, reset to `\TABstackText` when you are done with parsing).

5 Known Bugs/Missing Features

1. **No Horizontal Equivalent of \TABstrut**

   Currently, there is no macro that provides the width of a column's content. I hope to remedy this deficiency in a future release.

2. **Nothing Equivalent to `\hline`**

   This is not a bug, so much as a notation of a missing feature. Currently there is nothing equivalent to `\hline` available for use in tabstackengine arguments.

Acknowledgements

I would like to thank Prof. Enrico Gregorio of tex.stackexchange.com for his considerable assistance rendered. Many of the improvements he offered towards my stackengine package were carried over directly into this package, as well, resulting in a more robust implementation.

6 Code Listing

```
\def\tabstackengineversionnumber{V1.10}
%
% tabstackengine initial release
%
% THIS MATERIAL IS SUBJECT TO THE LaTeX Project Public License
%
% V1.00 -Adopted beta version 0.21 as initial release version 1.0
% V1.10 -Corrected unary/binary problem for left end of tabbed cell content;
%       -Added \TABunaryLeft (\TABbinaryRight) for "   cell{} ";
%        added \TABunaryRight (\TABbinaryLeft) for " {}cell   ";
%        added \TABbinary                      for " {}cell{} ";
%        The default is \TABunaryLeft (V1.00 wrongly equivalent to \TABbinary)
%        This removes need to brace unary negatives at lead of cell.
%       -Corrected bug of trailing \frac, noted in V1.00, by adding a
%        \relax to definition of \@postTAB in \readTABrow.
%
\ProvidesPackage{tabstackengine}
[2014/02/19 (\tabstackengineversionnumber) tabbed stacking]
\RequirePackage[usestackEOL]{stackengine}[2013-10-15]
\RequirePackage{calc}

\newtoggle{@doneTABreads}
\newcounter{TAB@stackindex}
\newcounter{TABcellindex@}
\newlength\maxTAB@width

\newcommand\setstackTAB[1]{%
  \ifstrempty{#1}{\def\TAB@char{ }}{\def\TAB@char{#1}}%
  \expandafter\define@processTABrow\expandafter{\TAB@char}%
}

\newcommand\define@processTABrow[1]{%
    \def\@processTABrow##1#1##2||{%
      \def\@preTAB{##1}%
      \def\@postTAB{##2}%
    }%
}
\setstackTAB{&}

\newcommand\readTABrow[2]{%
  \edef\row@ID{#1}%
  \togglefalse{@doneTABreads}%
  \edef\@postTAB{\unexpanded{#2\relax}\expandonce{\TAB@char}}%
  \setcounter{TABcellindex@}{0}%
  \whileboolexpr{ test {\nottoggle{@doneTABreads}}}{%
    \stepcounter{TABcellindex@}%
    \expandafter\@processTABrow\@postTAB||%
    \global\csedef{TABX\row@ID X\roman{TABcellindex@}}{\expandonce\@preTAB}%
    \setbox0=\hbox{%
      \stack@delim\TABcell{\row@ID}{\theTABcellindex@}\stack@delim}%
    \ifdim\wd0>\maxTAB@width\setlength{\maxTAB@width}{\the\wd0}\fi%
    \expandafter\ifstrempty\expandafter{\@postTAB}{%
      \toggletrue{@doneTABreads}%
    }{}%
```

```
  }%
  \expandafter\xdef\csname \row@ID TABcells\endcsname{\arabic{TABcellindex@}}%
  \find@TABstrut{#1}%
}

\newcommand\find@TABstrut[1]{%
  \def\@accumulatedTAB{}%
  \setcounter{TABcellindex@}{0}%
  \expandafter\protected@edef\csname @#1TABtextblob\endcsname{}%
  \whileboolexpr{test {\ifnumless{\theTABcellindex@}{\TABcells{#1}}}}{%
    \stepcounter{TABcellindex@}%
    \protected@edef\@accumulatedTAB{%
      \@accumulatedTAB\TABcell{#1}{\theTABcellindex@}}%
  }%
  \global\csedef{@#1TABtextblob}{\expandonce\@accumulatedTAB}%
}

\def\p@Tstrut#1{\protect\TABstrut{#1}}

\newcommand\TABunaryLeft{%
  \def\TAB@strutLeftA{}%
  \def\TAB@strutRightA{\p@Tstrut{i}}%
  \def\TAB@strutLeftB{}%
  \def\TAB@strutRightB{\p@Tstrut{ii}}%
  \def\TAB@strutLeftC{}%
  \def\TAB@strutRightC{\p@Tstrut{\TAB@prefix}}%
}
\let\TABbinaryRight\TABunaryLeft

\newcommand\TABunaryRight{%
  \def\TAB@strutLeftA{\p@Tstrut{i}}%
  \def\TAB@strutRightA{}%
  \def\TAB@strutLeftB{\p@Tstrut{ii}}%
  \def\TAB@strutRightB{}%
  \def\TAB@strutLeftC{\p@Tstrut{\TAB@prefix}}%
  \def\TAB@strutRightC{}%
}
\let\TABbinaryLeft\TABunaryRight

\newcommand\TABbinary{%
  \def\TAB@strutLeftA{{}}%
  \def\TAB@strutRightA{\p@Tstrut{i}}%
  \def\TAB@strutLeftB{{}}%
  \def\TAB@strutRightB{\p@Tstrut{ii}}%
  \def\TAB@strutLeftC{{}}%
  \def\TAB@strutRightC{\p@Tstrut{\TAB@prefix}}%
}
\TABunaryLeft

\newcommand\TAB@delim[1]{#1}%
\newcommand\TABstackMath{%
  \renewcommand\TAB@delim[1]{\ensuremath{##1}}%
}
\newcommand\TABstackText{%
  \renewcommand\TAB@delim[1]{##1}%
}
```

```
\newcommand\TABcell[2]{\TAB@delim{%
  \ignorespaces\csname TABX#1X\romannumeral#2\endcsname\unskip}}

\newcommand\TABcells[1]{\csname #1TABcells\endcsname}

% NOTE THAT THE STRUT IS NOT YET TYPESET, SINCE WE DO NOT YET KNOW
% WHETHER \@#1textblob IS TO BE INVOKED IN MATH MODE OR NOT
\newcommand\TABstrut[1]{\vphantom{\csname @#1TABtextblob\endcsname}}

\newcommand\tabbedShortstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Shortstack}}

\newcommand\alignShortstack[1]{%
  \@TAB@stack{}{#1}{A}{\Shortstack}}

\newcommand\tabularShortstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Shortstack}}

\newcommand\tabbedShortunderstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Shortunderstack}}

\newcommand\alignShortunderstack[1]{%
  \@TAB@stack{}{#1}{A}{\Shortunderstack}}

\newcommand\tabularShortunderstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Shortunderstack}}

\newcommand\tabbedLongstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Longstack}}

\newcommand\alignLongstack[1]{%
  \@TAB@stack{}{#1}{A}{\Longstack}}

\newcommand\tabularLongstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Longstack}}

\newcommand\tabbedLongunderstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Longunderstack}}

\newcommand\alignLongunderstack[1]{%
  \@TAB@stack{}{#1}{A}{\Longunderstack}}

\newcommand\tabularLongunderstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Longunderstack}}

\newcommand\tabbedCenterstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Centerstack}}

\newcommand\alignCenterstack[1]{%
  \@TAB@stack{}{#1}{A}{\Centerstack}}

\newcommand\tabularCenterstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Centerstack}}

\newcommand\tabbedVectorstack[2][\stackalignment]{%
  \ensureTABstackMath{\@TAB@stack{#1}{#2}{D}{\Vectorstack}}}
```

```
\newcommand\alignVectorstack[1]{%
  \ensureTABstackMath{\@TAB@stack{}{#1}{A}{\Vectorstack}}}

\newcommand\tabularVectorstack[2]{%
  \ensureTABstackMath{\@TAB@stack{}{#2}{#1}{\Vectorstack}}}

\newcommand\parenMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left(\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right)}}

\newcommand\braceMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left\{\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right\}}}

\newcommand\bracketMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left[\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right]}}

\newcommand\vertMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left|\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right|}}

\newcommand\@TAB@stack[4]{%
  \setlength{\maxTAB@width}{0pt}%
  \let\sv@stackalignment\stackalignment%
  \edef\stackalignment{#1}%
  \@readMANYrows{#2}%
  \edef\TAB@narg{\narg}%
  \setcounter{TAB@stackindex}{0}%
  \whileboolexpr{test {\ifnumless{\theTAB@stackindex}{\TAB@narg}}}{%
    \stepcounter{TAB@stackindex}%
    \edef\TAB@prefix{\roman{TAB@stackindex}}%
    \protected@edef\TABrow@data{\csname arg\TAB@prefix\endcsname}%
    \def\@tmp{\readTABrow{\TAB@prefix}}%
    \expandafter\@tmp\expandafter{\TABrow@data}%
  }%
%
  \setcounter{TABcellindex@}{0}%
  \whileboolexpr{test {\ifnumless{\theTABcellindex@}{\TABcells{i}}}}{%
    \def\col@stack{%
        \TAB@strutLeftA%
        \TABcell{i}{\theTABcellindex@}%
        \TAB@strutRightA%
}%
    \stepcounter{TABcellindex@}%
    \@getTABalignment{#3}{\theTABcellindex@}%
    \ifboolexpr{%
      test {\ifnumgreater{\theTABcellindex@}{1}}%
    }{\add@TAB@gap{#3}{\theTABcellindex@}}{}%
    \setcounter{TAB@stackindex}{1}%
    \whileboolexpr{test {\ifnumless{\theTAB@stackindex}{\TAB@narg}}}{%
      \stepcounter{TAB@stackindex}%
      \edef\TAB@prefix{\roman{TAB@stackindex}}%
      \protected@edef\col@stack{\col@stack\SEP@char%
        \TAB@strutLeftC%
        \TABcell{\TAB@prefix}{\theTABcellindex@}%
        \TAB@strutRightC%
      }%
    }%
    \iftoggle{fixed@TABwidth}%
      {\makebox[\the\maxTAB@width][\stackalignment]{%
```

```
        \expandafter#4\expandafter{\col@stack}}}%
      {\expandafter#4\expandafter{\col@stack}}%
  }%
  \let\stackalignment\sv@stackalignment%
}

\newcommand\tabbedstackon[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackon}}

\newcommand\alignstackon[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackon}}

\newcommand\tabularstackon[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackon}}

\newcommand\tabbedstackunder[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackunder}}

\newcommand\alignstackunder[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackunder}}

\newcommand\tabularstackunder[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackunder}}

\newcommand\tabbedstackanchor[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackanchor}}

\newcommand\alignstackanchor[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackanchor}}

\newcommand\tabularstackanchor[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackanchor}}

\newcommand\@TABstackonunder[5]{%
  \setlength{\maxTAB@width}{0pt}%
  \let\sv@stackalignment\stackalignment%
  \readTABrow{i}{#2}%
  \readTABrow{ii}{#3}%
  \setcounter{TABcellindex@}{0}%
  \whileboolexpr{test {\ifnumless{\theTABcellindex@}{\TABcells{i}}}}{%
    \stepcounter{TABcellindex@}%
    \@getTABalignment{#4}{\theTABcellindex@}%
    \ifboolexpr{%
      test {\ifnumgreater{\theTABcellindex@}{1}}%
    }{\add@TAB@gap{#4}{\theTABcellindex@}}{}%
    \iftoggle{fixed@TABwidth}%
      {\makebox[\the\maxTAB@width][\stackalignment]{%
       #5[#1]{\TAB@strutLeftA\TABcell{i}{\theTABcellindex@}\TAB@strutRightA}%
              {\TAB@strutLeftB\TABcell{ii}{\theTABcellindex@}\TAB@strutRightB}}}%
      {#5[#1]{\TAB@strutLeftA\TABcell{i}{\theTABcellindex@}\TAB@strutRightA}%
              {\TAB@strutLeftB\TABcell{ii}{\theTABcellindex@}\TAB@strutRightB}}%
  }%
  \let\stackalignment\sv@stackalignment%
}

\newcommand\@getTABalignment[2]{%
  \ifstrequal{#1}{D}{}{%                    T, DO NOTHING (USE \stackalignment)
```

```
    \ifstrequal{#1}{A}{%
      \ifnumequal{1}{#2}{%
        \def\stackalignment{r}}{%          A, 1st ELEMENT, SET TO r
        \if l\stackalignment%
          \def\stackalignment{r}\else%     A, SWITCH l TO r
          \def\stackalignment{l}\fi}}{%    A, SWITCH r TO l
          \set@tabularcellalignment{#1}{#2}%   tabular, READ #2 location
      }%
  }%
}

\def\tabbed@gap{0pt}
\def\align@gap{1em}
\def\tabular@gap{\tabcolsep}

\newcommand\setstacktabbedgap[1]{\def\tabbed@gap{#1}}
\newcommand\setstackalignngap[1]{\def\align@gap{#1}}
\newcommand\setstacktabulargap[1]{\def\tabular@gap{#1}}

\newcommand\add@TAB@gap[2]{%
  \ifstrequal{#1}{D}{\hspace{\tabbed@gap}}{%
    \ifstrequal{#1}{A}{%
      \if r\stackalignment\hspace{\align@gap}\fi%
    }{%
      \hspace{\tabular@gap}%
    }%
  }%
}

\newcounter{TABalignmentindex@}

\newcommand\set@tabularcellalignment[2]{%
  \setcounter{TABalignmentindex@}{1}%
  \edef\tabular@settings{#1.}%
  \whileboolexpr{test {\ifnumless{\theTABalignmentindex@}{#2}}}{%
    \stepcounter{TABalignmentindex@}%
    \edef\tabular@settings{\expandafter\@gobble\tabular@settings.}%
  }%
  \expandafter\@getnextTABchar\tabular@settings\\% GET NEXT TAB ALIGNMENT
  \if l\@nextTABchar\edef\stackalignment{l}\else%
    \if r\@nextTABchar\edef\stackalignment{r}\else%
      \if c\@nextTABchar\edef\stackalignment{c}\fi%
    \fi%                                          IGNORE IF NOT l, c, OR r
  \fi%
}

\def\@getnextTABchar#1#2\\{\gdef\@nextTABchar{#1}}


\newtoggle{fixed@TABwidth}
\newcommand\fixTABwidth[1]{%
  \if T#1\toggletrue{fixed@TABwidth}\else\togglefalse{fixed@TABwidth}\fi%
}
\fixTABwidth{F}

\newcommand\ensureTABstackMath[1]{\TABstackMath#1\TABstackText}
```

```
\endinput
```