Using boxes and glue in TEX and LATEX

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
Email: beebe@math.utah.edu, beebe@acm.org,
beebe@computer.org
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148

28 March 2009

Abstract

This document shows how to position text in boxes in TEX and LATEX, and provides a convenient set of commands for doing so simply.

1 Introduction

TEX and LATEX are powerful systems for typesetting and document markup, but that power often means considerable complexity. Both have lots of commands that help to hide the low-level details. However, a major challenge in typesetting most technical documents is to identify additional operations that are commonly needed for that particular document. One can then define private commands that strip the details entirely away, leaving just the name of the operation, and the text that it consumes. The input job is then much easier, and the input file is much more readable, especially if care is taken in its preparation to use consistent spacing and logical indentation, and to keep lines relatively short. It can otherwise be rather painful to find a missing or mismatched brace that TEX complains about in a line that is thousands of characters long.

It is important to remember that, as a document author, you are likely to spend much more time looking at the TEX *input* than at its nicely typeset output, so readable input should matter to *you*. If you do your typesetting job well, your readers will benefit, even it they might not be aware that

they have. They will certainly notice, however, if your output looks like the trashy regurgitation of a word processor on a typical desktop computer.

TEX and LATEX are capable of producing beautifully typeset and easy-to-read documents, but they cannot do so without the help of document authors and document style designers.

The next section of this document provides some of the low-level details of how TEX handles the typesetting process. If you find it complex, do not be alarmed. The whole point is to learn how to conceal the messy details inside easy-to-use commands, and once those commands have been worked out, everyone can use them without thought of their underpinnings. The section after that deals with the same problem, but at a logically-higher level, where we build similar new commands with less trouble.

2 Boxes and glue in TEX

The TEX typesetting system uses a model of *boxes* and *glue*. Boxes contain typeset objects, such as text, mathematical displays, and pictures, and glue is flexible space that can stretch and/or shrink by amounts that are under user control.

When TEX is typesetting, it is normally in *horizontal mode*, such as while it is working on this paragraph. Otherwise, TEX can be in *vertical mode*, or in *math mode*, or three others described in Chapter 13 of *The TEXbook* [2].

Two low-level TEX commands for boxes are `\hbox`, for a horizontal box, and `\vbox` for a box in vertical mode. In the latter, TEX is normally still collecting material for display from right to left: it is *not* building up a column of text, as in classical Chinese writing.

In both kinds of boxes, the result is an unbreakable object that acts much like a single character. TEX reads input as a string of characters, then breaks that string up in words, each of which forms a box. Word boxes are then collected into lines, lines into paragraphs, and paragraphs into a page galley. The space between the words can be normal interword space, or sentence-ending space, which is somewhat larger in English-language typesetting, and the space is normally glue, rather than of fixed size.

TEX has a sophisticated mathematical algorithm for figuring out the best way to stretch or shrink interbox glue to optimize the appearance of lines and paragraphs. Every so often, TEX checks to see whether it has enough material saved on the growing page galley to fill a complete output page, and it asynchronously (and effectively, unpredictably) calls the *output routine* whose job it is to figure out where the page break should happen, ship out a completed page to the DVI file, and replace the galley by whatever is left over.

TEX users can force a line break with the carriage-return command `\cr`, and a page break with the command `\eject`, but TEX is an expert system, and normally handles line and page breaking on its own.

2.1 Units of measurement in TEX

TEX allows you to specify sizes of typographical objects in any of nine different units:

bp	big point: 1 inch is exactly 72 bp; the PostScript page-description language uses these units, but just calls them points
cc	cicero: 1 cc is exactly 12 didôt points, and is thus the European analogue of the pica
cm	centimeter: 1 in is exactly 2.54 cm
dd	didôt point: 1 dd is (1238/1157) pt, and is a typographical unit common in some parts of Europe
in	inch: an archaic unit, roughly the width of a man's thumb; it has been discarded by most countries, but is still used in the USA
mm	millimeter: 1 in is exactly 25.4 mm
pc	pica: 1 pc is exactly 12 pt
pt	printer's point: 1 in is exactly 72.27 pt
sp	scaled point: 1 pt is exactly $2^{16} = 65536$ sp.

The units can be separated from their numeric value with optional space, so `3pc` and `3␣pc` are equivalent. The little half box in the latter is a convenient way to indicate explicit spaces in typewriter text.

Internally, TEX stores dimensions as integral numbers of scaled points: 1 sp is tiny — smaller than the wavelength of visible light. It is sometimes useful to create objects that small so that they differ from empty objects, but are nevertheless invisible.

TEX deals only with 32-bit integer words, and does not take advantage of extra precision available on historical machines with larger words. The lower 16 bits of a dimension can be viewed as a fractional number of points, and the uppermost bit is needed for a sign (0 for plus, 1 for minus). That leaves 15 bits to hold an integral number of points, but TEX only expects 14 to be used, so that addition of two dimensions does not overflow. Thus, the largest dimension in TEX is exactly $2^{14} + (1 - 2^{-16})$ points, or about 5.758 meters or 18.89 feet.

TEX has several kinds of special storage locations, called *registers*, numbered from 0 to 255. For example, `\dimen0` can hold a fixed dimension, which can be specified in any of the nine units of measurement that are recognized by TEX.

Here is how you can assign a dimension to a register, and then have TEX display it back for you:

```
\dimen1 = 1in

\showthe \dimen1
> 72.26999pt.
```

```
\dimen2 = 10pc

\showthe \dimen2
> 120.0pt.

\dimen3 = 10cc
\showthe \dimen3
> 128.40103pt.
```

Notice that TEX's output is always in points, showing that it converts different input units to a common system of measurement.

You can convert a dimension to the much-smaller units of scaled points by assigning it to another kind of TEX register designed to hold signed integers, the `\count0` through `\count255` registers:

```
\dimen4 = 0.5pt
\count4 = \dimen4
\showthe \count4
> 32768.
```

You might have noticed that the conversion from inches to points was not quite what we claimed in the summary of TEX units. Here is how to see the differences:

```
\dimen1 = 1in
\dimen2 = 72.27pt
\count1 = \dimen1
\count2 = \dimen2

\showthe \count1
> 4736286.

\showthe \count2
> 4736287.
```

The two values differ by the tiny value 1 sp, so we can in practice ignore that difference. If we use higher-precision arithmetic, we find the *exact* decimal equivalents of the fractions as

$$\begin{aligned} 4\,736\,286/65\,536 &= 72.269\,989\,013\,671\,875, \\ 4\,736\,287/65\,536 &= 72.270\,004\,272\,460\,937\,5, \\ 4\,736\,286.72/65\,536 &= 72.27. \end{aligned}$$

TEX actually uses that last relation as the definition of the conversion of inches to scaled points, so that our assignment of 1 in to `\dimen1` has to be rounded to the nearest integral number of scaled points. That is why in the round-trip conversion from decimal to binary and back to decimal,

1 in became 72.26999 pt. TEX guarantees that its output decimal numbers are always converted on input back to the original binary numbers from whence they came. For more on the story of TEX's I/O conversions, see [3].

2.2 Limits of TEX arithmetic

While TEX detects overflow on assignment and multiplication, it does *not* do so on addition or subtraction, on the grounds that such overflows should be nonexistent, or at least rare, in the typesetting of practical documents. Here are experiments to show what happens:

```
\dimen1 = \maxdimen          % assign the largest possible dimension

\showthe \dimen1
16383.99998pt.

\count1 = \dimen1            % convert to scaled points

\showthe \count1
1073741823.

\dimen2 = 1073741823sp       % assign the largest possible dimension

\dimen3 = 1073741824sp       % assign 1 + largest possible dimension
! Dimension too large.

\showthe \dimen3             % TeX replaces it by \maxdimen
> 16383.99998pt.

\dimen4 = \dimen2
\advance \dimen4 by \dimen4  % form (largest + largest)

\showthe \dimen4             % it works!
32767.99997pt.

\dimen5 = \dimen4            % try to assign the overlarge value
! Dimension too large.

\showthe \dimen5             % TeX replaces it by \maxdimen
16383.99998pt.

\advance \dimen5 by 16383pt  % form (16383.99998pt + 16383pt)

\showthe \dimen5             % it works: value is almost 2 * \maxdimen
32766.99998pt.

\advance \dimen5 by 16383pt  % form (16383.99998pt + 16383pt + 16383pt)

\showthe \dimen5             % ERROR: uncaught overflow gave negative!
```

```
-16386.00002pt.

\dimen6 = 2\dimen2          % form 2 * (largest possible dimension)
! Dimension too large.

\showthe \dimen6            % TeX replaces it by \maxdimen
16383.99998pt.
```

A similar experiment carried out the 36-bit PDP-10 shows the same behavior up to the assignment to \dimen6, but since there are actually four more bits available for the integer part, we get this instead:

```
*\showthe \dimen5
> 81915.99998pt.
```

We then repeat the addition 31 more times, and finally we have a problem:

```
\advance \dimen5 by 16383pt \showthe \dimen5 % try 30
524256.99998pt.

\advance \dimen5 by 16383pt \showthe \dimen5 % try 31
?
? Integer overflow at user PC 1,,646437
[Type CONTINUE to proceed if possible,
      REENTER to close all files and exit.]
@continue
-507936.00002pt.
```

TEX continues the additions until *five* extra bits have been used ($36-32=4$ bits, plus one overflow bit), and then the arithmetic system used by the native Pascal compiler traps a real integer overflow, and pauses the program at the operating-system prompt. We then continue it, and find a negative value from the overflow of addition into the sign bit.

2.3 Type sizes in TEX

In order to keep common numbers of manageable size, humans often adopt specialized units of measurement. Astronomers use light years and megaparsecs, highway engineers use kilometers and miles, and many typographers use points and picas. Typical book text is set in 10 pt type on lines spaced 1 pc or 12 pt apart. This is called a 10/12, or 10 on 12, design. Footnotes might be set in 7 pt type, and headings in 12 pt, 14 pt, and 16 pt, with the book title on the cover set at 48 pt. Children's books use larger type, usually 12 pt or 14 pt, and large-print books for visually-impaired readers might use 18 pt or 24 pt type. Here are some examples:

This is 7 pt type.
This is 10 pt type.
This is 12 pt type.
This is 14 pt type.
This is 18 pt type.
This is 24 pt type.

Many type faces designed since the computer age have the same letter shapes in all type sizes, but readability is improved if the shapes are varied. Compare these examples:

New Century Schoolbook at 5 pt
New Century Schoolbook at 10 pt
New Century Schoolbook at 15 pt

Computer Modern Roman 5 at 5 pt
Computer Modern Roman 10 at 5 pt
Computer Modern Roman 17 at 5 pt
Computer Modern Roman 5 at 10 pt
Computer Modern Roman 10 at 10 pt
Computer Modern Roman 17 at 10 pt
Computer Modern Roman 5 at 15 pt
Computer Modern Roman 10 at 15 pt
Computer Modern Roman 17 at 15 pt

Notice that letters in the smaller design sizes of the Computer Modern family are somewhat wider, so when they are scaled up to a common size, their sample strings are longer.

2.4 Specifying glue amounts in TEX

TEX glue is specified as a fixed dimension, and optionally, with a plus and/or minus dimension. Along with `\dimen` registers, TEX has glue registers, called `\skip0` through `\skip255`. Here is how you can save glue settings in TEX registers, and ask TEX to display the contents of one of them:

```
\skip1 = 10pt
\skip2 = 10pt plus 3pt
\skip3 = 10pt minus 2pt
\skip4 = 10dd plus 3dd minus 2dd
\showthe \skip4
> 10.70007pt plus 3.21002pt minus 2.14001pt.
```

The four sample glue settings store, respectively, fixed glue, stretchable glue, shrinkable glue, and flexible glue that can both stretch and shrink, but only up to a specified amount. Interword and intersentence spaces are generally defined with glue like this, so that if more stretch or shrink of

spaces is needed than is available, TEX can warn about lines of text that are *underfull* (too little text to fill the line), or *overfull* (too much text in the line).

Although overfull lines are reported in the TEX log file, they can be hard to find in the typeset document if they only stick out a little. To make them highly visible while you are fine tuning your final document, assign the variable `\overfullrule` a nonzero dimension, such as 10 cm. TEX then displays a solid black box, called a *rule*, of that width in the right margin on each line that is overfull. To make the rules disappear, simply remove, or comment out, the assignment, or reset its value to 0 pt.

Just as you can assign dimension registers to count registers to convert from points to scaled points, you can assign skip registers to dimension and count registers to discard the flexible parts:

```
\skip1 = 10pt plus 3pt minus 2pt
\showthe \skip1
> 10.0pt plus 3.0pt minus 2.0pt.

\dimen1 = \skip1
\showthe \dimen1
> 10.0pt.

\count1 = \skip1
\showthe \count1
> 655360.
```

2.5 More on glue in boxes

Besides normal glue with fixed amounts of stretch and shrink, TEX also has two kinds of glue that are 'infinitely' stretchable and shrinkable: `\hfil` and `\hfill` in horizontal mode, and `\vfil` and `\vfill` in vertical mode. The two-ell forms are more flexible than the one-ell forms.

The boxes and glue model is powerful, and TEX's author, Donald Knuth, has written that he views it as the key idea that he discovered when he first sat down in 1977–1978 to design a computer program for typesetting. For example, to set something flush left, put infinitely-stretchable glue on its right. To set it flush right, put the glue on the left. For centered material, put the glue on both sides. Here are four examples, with vertical bars marking the ends of the horizontal box (boxes have no visible frames, although it is possible to write TEX commands to give them such outlines, and we use that feature shortly):

```
|\hbox to 40pt{word}|              |word    |
|\hbox to 40pt{word \hfil}|        |word    |
|\hbox to 40pt{\hfil word}|        |    word|
|\hbox to 40pt{\hfil word\hfil}|   |  word  |
```

The first two look identical, because TEX typesets the word, and then makes a 40pt-wide box from it. However, in the first case, there is not enough material to fill that box, so TEX complains that something is wrong:

```
Underfull \hbox (badness 10000) detected at line 169
```

This is not an error, but merely warns the user that the typeset output may not be optimal. The badness value reported is part of the mathematical algorithm that TEX uses for line and page breaking, but we do not consider it further in this document.

Supplying the trailing \hfil glue in the second example makes TEX happy, because the flexible glue allows the box to be filled exactly.

There is yet another kind of glue that can both stretch and shrink by an 'infinite' amount: \hss (horizontal infinitely shrinkable and stretchable glue), and its companion for vertical mode, \vss. Here is why this doubly-flexible glue is useful. Suppose you want to put an object in a box, but you do not know in advance whether the box is big enough, and you do not care whether or not it is. TEX complains about overfull boxes if there is too much text, but is silent if there is doubly-flexible glue involved:

```
|\hbox to 10pt{word}|            |word
|\hbox to 10pt{word \hss}|       |word
|\hbox to 10pt{\hss word}|     word|
|\hbox to 10pt{\hss word\hss}|  |word|
```

In the first case, TEX warns

```
Overfull \hbox (12.95982pt too wide) detected at line 201
```

but in the other three cases, no warning is issued.

The plain TEX macro package uses the horizontal shrink-or-stretch command in the definitions of two convenient macros for typesetting text to the left or right of the current point, but treating it as zero width, so that it can overlap surrounding text:

```
\def \rlap #1{\hbox to 0pt{#1\hss}}
\def \llap #1{\hbox to 0pt{\hss #1}}
```

The right-overlap macro makes its argument stick out to the right of the current point, whereas its left-overlap companion makes the argument protrude to the left. Here is an example:

stuff|comes from \llap{stuff}|, while |\rlap{~extra} is typeset away from this text as |.extra

Notice that the sentence-ending period overlaps the space preceding the last word, and that the first word sticks out in the left margin of this indented display.

If no width is given for the horizontal box, then TEX creates a box of exactly the right size to hold its contents. Fill glue then disappears entirely:

```
\def \W {word}
|\W \W|                        |wordword|
|\W \hfil \W|                  |wordword|
|\hfil \W \hfil \W|            |wordword|
|\hfil \W \hfil \W \hfil|      |wordword|
|\hfil \W \hfil \W \hfil|      |wordword|
```

There is no space between the words, even though we used a space after the `\W` shorthands, because of TEX's rule that spaces are ignored after macros whose names are one or more letters. TEX's official name for these is *control words*.

TEX also has macros that consist of a backslash and a single special character, such as `\%`. They are called *control sequences*, and TEX does not ignore spaces after them.

If we add fixed-width boxes to our last example, the fill glue takes effect again:

```
|\hbox to 60pt{\W \W}|                      |wordword    |
|\hbox to 60pt{\W \hfil \W}|                |word   word|
|\hbox to 60pt{\hfil \W \hfil \W}|          | word  word|
|\hbox to 60pt{\hfil \W \hfil \W \hfil}|    | word word |
|\hbox to 60pt{\hfil \W \hfil \W \hfil}|    | word word |
```

We can make a few experiments with the two-ell forms of glue like this:

```
|\hbox to 60pt{\hfil \W \hfill}|    |word        |
|\hbox to 60pt{\hfill \W \hfil}|    |        word|
|\hbox to 60pt{\hfill \W \hfill}|   |    word    |
```

Clearly, two-ell glue overwhelms one-ell glue. This can be useful in the design of complex macros for positioning of text in boxes.

2.6 More features of horizontal boxes

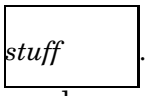
Characters in the Latin alphabet have different shapes, and in most typefaces, different widths. The letters *d f h k l t* have *ascenders*, making them higher than the vowels *a e o u*, while the letters *f g j p q y* have *descenders*, giving them added depth below the vowels. Similarly, an *m* is wider than an *i*. When TEX makes a normal horizontal box, the box width is the sum of the widths of the characters, and the fixed parts of any glue, contained in it. Shrink and stretch components of glue are discarded for the width calculation. The box also has both a *height* above the *baseline*, the invisible line on which the characters rest, and a *depth* below the baseline. The depth is zero if there are no objects with descenders. The height and depth are chosen from the largest vertical extents of the contained objects.

If you look carefully at typeset material, you will observe that, in most typefaces, parentheses, brackets, and braces have both descenders and ascenders, and the typeface designer usually makes their extents the maximum among all of the characters in the design. This sample text shows

that design choice in the New Century Schoolbook typeface used in this document: *(h g) [k j] { l p }*.

You can force TEX to choose a larger height and depth than normal when you write a command for a horizontal box by ensuring that it has suitable contents, such as an invisible vertical rule of zero width. The command

```
\hbox to 50pt {\vrule height 20pt depth 10pt width 0pt \it stuff}
```

produces a box whose (invisible) outline looks like this: 

. The three extents of the vertical rule can appear in any order, and any convenient units.

In order to see the otherwise-invisible box edges in that example, we used the LATEX built-in command `\fbox` to create a frame, and we eliminated the default margin inside the frame by setting `\fboxsep = 0pt`. Plain TEX does not have the `\fbox` command, but *The TEXbook* shows how to make something like it on pp. 223 and 321.

One particular zero-width vertical rule is convenient for ensuring that separate boxes all get the same height and depth. It has the height and depth of parentheses in the normal prose font, and is given the macro name `\strut`. Its definition in the `plain.tex` file of macro definitions is roughly equivalent to this:

```
\def \strut {\vrule height 8.5pt depth 3.5pt width 0pt}
```

Compare these two experiments with outlined boxes, first without struts, and then with struts:

```
\def \Fbox #1{\fbox{\strut #1}}
\fbox{aeou} \fbox{dhkl} \fbox{gpq}   aeou dhkl gpq
\Fbox{aeou} \Fbox{dhkl} \Fbox{gpq}   aeou dhkl gpq
```

Notice the different vertical extents of the boxes in the first case, and how they have identical extents in the second case.

2.7 Horizontal alignment of boxes in TEX

When horizontal boxes are set together, they are treated as separate words, and therefore spaced accordingly. The input

```
\fbox{one} \fbox{two} \fbox{three}\fbox{four}
```

produces one two threefour. As the example shows, we can put spaces between them, or run them together so that they fit tightly.

2.8 Vertical boxes in TEX

Now let us investigate vertical boxes, and see how we can position text inside them using glue to push material left, right, up, and down. Further, to simplify the typesetting of this document, we show first the TEX input and then its output, without trying to place them side by side, as we did earlier. Typesetting has now become significantly more complex, and there are several important points to note:

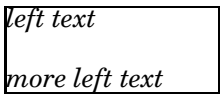
- The `\fbox` and its contents are a separate paragraph in this document, so we turn off indentation with `\noindent` to get the frame aligned with the left text margin.
- The `\fbox` macro always leaves a margin of width `\fboxsep` around its argument, and the margin default width is small, just 3 pt. We set it to zero above so that, in this document, the frame is always tight against its contents.
- The `\fbox` expects an argument in horizontal mode, so we wrap that argument in an `\hbox` of a specified width, after switching to an italic font for the box contents.
- Entering vertical mode is something like starting a new paragraph, so in order to control precisely the positioning of text inside the `\vbox`, we reset the paragraph indentation to 0 pt. Because the assignment to `\parindent` is inside the horizontal box, the change is lost once the box is complete, so it does not affect paragraph indentation in the rest of this document.
- Although many TEX programmers run TEX commands together in horrid unreadable messes, we prefer to use indentation and vertical alignment of matching braces to clarify the logical nesting. In particular, this means that immediately after an open or close brace at end of line, we put a comment-starting percent that causes TEX to ignore the rest of the line, and *all following horizontal space on the next line*. Without that comment, a newline following the open brace, and all leading horizontal space on the next line, would be treated as a single space by TEX, and that would affect the text positioning in the box.

  Failure to supply such comments in macro definitions is a common source of mistakes of unwanted space in typeset output, and TEX has no easy provision for warning you about such problems; it just assumes that you are a responsible and reliable co-worker.
- When TEX finishes a paragraph, or when in horizontal mode it meets a vertical-mode command like `\vfil`, TEX ends the line, and adds an implicit one-ell `\hfil` that supplies padding glue after the text on that line. That is why TEX does not complain about an underfull box at the end of most paragraphs. In our examples, this means that a leading

one-ell \hfil on the lines of text inside the vertical box would actually be set *centered*, rather than set flush left. The two-ell \hfill overwhelms the one-ell form, and forces text flush left. We show examples of both of these.

Text *flush left* at top and bottom

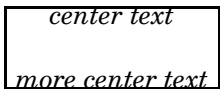
```
\noindent
\fbox{%
      \it
      \hbox to 80pt{%
                    \parindent = 0pt
                    \vbox to 30pt {%
                                     left text
                                     \vfil
                                     more left text%
                                   }%
                  }%
}%
```



The central one-ell \vfil inside the vertical box forced the two lines to the top and bottom of the box.

Text *centered* at top and bottom

```
\noindent
\fbox{%
      \it
      \hbox to 80pt{%
                    \parindent = 0pt
                    \hsize = 80pt
                    \vbox to 30pt {\hfil center text
                                   \vfil
                                   \hfil more center text}
                  }%
}%
```



As we observed in the last list item, the one-ell horizontal fill here results in centered text.

Text *flush right* at top and bottom

```
\noindent
\fbox{%
      \it
      \hbox to 80pt{%
                    \parindent = 0pt
                    \hsize = 80pt
                    \vbox to 30pt {\hfill right text
                                   \vfil
                                   \hfill more right text}
                   }%
}%
```

right text

more right text

Using the leading two-ell horizontal fill pushes the text flush against the right edge of the vertical box.

Text *flush left* at top and bottom

```
\noindent
\fbox{%
      \it
      \hbox to 80pt{%
                    \parindent = 0pt
                    \hsize = 80pt
                    \vbox to 30pt {left text \hfill
                                   \vfil
                                   more left text \hfill}
                   }%
}%
```

left text

more left text

This one is similar to the first vertical box example, but supplies explicit two-ell horizontal fill on the right, forcing text to the left edge of the vertical box.

2.9 Vertical alignment of boxes in TEX

When vertical boxes are set together, they are treated as separate *lines* starting new paragraphs, and therefore spaced accordingly. The input

```
\vbox{\fbox{one}}
\vbox{\fbox{two}}
\vbox{\fbox{three}}
\vbox{\fbox{four}}\vbox{\fbox{five}}\vbox{\fbox{six}}
```

produces one
two
three
four five

The first four boxes are indented according to the current value of `\parindent`. The sixth box disappeared off the edge of the page, because it was given no preceding space at which the line could be broken.

If we put those boxes inside a *vertical* box, inside of which we set the horizontal size, then we can control the line width:

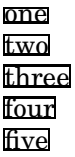
```
\vbox
{%
    \hsize = 40pt
    \parindent = 0pt
    \vbox{one}
    \vbox{two}
    \vbox{three}
    \vbox{four}\vbox{five}\vbox{six}%
}
```

one
two
three
four
five

produces six . The input spacing between the vertical boxes has no visible effect.

There is some additional vertical spacing in that last display that we can reveal by framing the contents of the inner horizontal boxes, so that the input

```
\vbox
{%
    \hsize = 40pt
    \parindent = 0pt
    \vbox{\fbox{one}}
    \vbox{\fbox{two}}
    \vbox{\fbox{three}}
    \vbox{\fbox{four}}\vbox{\fbox{five}}\vbox{\fbox{six}}%
}
```



produces six , and again, the input spacing does not matter.

We can eliminate the extra vertical spacing with \offinterlineskip to get tight vertical stacking with the input

```
\vbox
{%
    \offinterlineskip
    \hsize = 40pt
    \parindent = 0pt
    \vbox{\fbox{one}}
    \vbox{\fbox{two}}
    \vbox{\fbox{three}}
    \vbox{\fbox{four}}\vbox{\fbox{five}}\vbox{\fbox{six}}%
}
```

to produce six .

If we instead fit the vertical boxes inside a *horizontal* box, with the input

```
\hbox
{%
    \hsize = 30pt
    \parindent = 0pt
    \vbox{\fbox{one}}
    \vbox{\fbox{two}}
    \vbox{\fbox{three}}
    \vbox{\fbox{four}}\vbox{\fbox{five}}\vbox{\fbox{six}}%
}
```

then we get output like this: one two three four five six , where the vertical boxes are each 30 pt wide.

As a final example, we stack some text inside the vertical boxes using TEX's \cr command to force line breaks. Since this document is written in LATEX markup, we have to restore the definition of that macro inside the outer box. The input

```
\hbox
{%
    \let \cr = \\
    \hsize = 30pt
    \parindent = 0pt
```

```
    \vbox{one \cr two}
    \vbox{three \cr four}
    \vbox{five \cr six}
}
```

then produces

one three five
two four six .

From our experiments, we conclude that vertical boxes get their natural heights from their contents, although the user can force a particular height with `\vbox to 30pt{...}`. Their width is the current horizontal size, `\hsize`. If TEX is in vertical mode, they stack vertically with spacing dependent on whether interline glue is being used or not. Otherwise, if TEX is forced to horizontal mode, and there is sufficient space, they stack horizontally.

2.10 General text positioning in TEX

The examples that we have presented give us the experimental evidence that we need to write some useful macros for general positioning of text in boxes, so that the result is a rectangular object that acts just like a letter in horizontal mode. Each is named with an initial letter that selects the horizontal text positioning (*left*, *center*, *right*), and takes four arguments: *box width*, *box height*, *top text*, and *bottom text*. Either of the last two may be empty when the macros are actually used.

```
%% Typeset left-adjusted text in box.
%% Usage: \Lbox{width}{height}{top text}{bottom text}
\def \Lbox #1#2#3#4%
{%                      left-adjusted text in box
    \hbox to #1
    {%
        \parindent = 0pt
        \hsize = #1
        \vbox to #2
        {%
            \strut #3%
            \vfill
            \strut #4%
        }%
    }%
}

%% Typeset centered text in box.
%% Usage: \Cbox{width}{height}{top text}{bottom text}
\def \Cbox #1#2#3#4%
{%                      centered text in box
    \hbox to #1
```

```
    {%
        \parindent = 0pt
        \hsize = #1
        \vbox to #2
        {%
            \hfil \strut #3%
            \vfill
            \hfil \strut #4%
        }%
    }%
}

%% Typeset right-adjusted text in box.
%% Usage: \Rbox{width}{height}{top text}{bottom text}
\def \Rbox #1#2#3#4%
{%                      right-adjusted text in box
    \hbox to #1
    {%
        \parindent = 0pt
        \hsize = #1
        \vbox to #2
        {%
            \hfill \strut #3%
            \vfill
            \hfill \strut #4%
        }%
    }%
}
```

TEX ignores spaces after box dimensions and numerical assignments, so we do not need final percent characters on those lines. The text arguments `#3` and `#4` are each preceded by TEX's `\strut` command. Here, we use it to ensure that *something* appears in both top and bottom positions, even if the user provides empty arguments, and also that both lines have the same height and depth (unless the argument text is unusually high or deep).

Without logical indentation and most of the horizontal spaces, here is how the last one could also be written:

```
%% Typeset right-adjusted text in box.
%% Usage: \Rbox{width}{height}{top text}{bottom text}
\def \Rbox #1#2#3#4{\hbox to #1{\parindent = 0pt \hsize = #1
                    \vbox to #2{\hfill \strut #3\vfill
                    \hfill \strut #4}}}
```

Youcanmakeitevenlessreadablebydroppingdocumentationandspacing, like this:

```
\def\Rbox#1#2#3#4{\hbox to#1{\parindent=0pt\hsize=#1\vbox to#2{%
```

```
\hfill\strut#3\vfill\hfill\strut#4}}}
```

Sadly, far too many TEX macro-definition files, and LATEX style files, look like that already. The low-level details of typesetting are hard, and it is simply foolish to make them even harder with inscrutable code. Humans are unlikely to notice the microseconds of computer time saved by eliminating spaces in macro files.

The input for the rest of this paragraph looks like this:

```
Here is how the boxes can be used inside a paragraph as if
they were single words:
\fbox{\Lbox{20pt}{20pt}{TL}{BL}}
\fbox{\Cbox{20pt}{20pt}{TC}{BC}}
\fbox{\Rbox{20pt}{20pt}{TR}{BR}}
\fbox{\Lbox{20pt}{20pt}{TL}{}}
\fbox{\Cbox{20pt}{20pt}{TC}{}}
\fbox{\Rbox{20pt}{20pt}{TR}{}}
\fbox{\Lbox{20pt}{20pt}{}{BL}}
\fbox{\Cbox{20pt}{20pt}{}{BC}}
\fbox{\Rbox{20pt}{20pt}{}{BR}}.
```

Here is how the boxes can be used inside a paragraph as if they were single words: TL BL TC BC TR BR TL TC TR BL BC BR. These boxes have a depth below the baseline because we defined them with a `\strut`.

2.11 Alignment control in TEX text positioning

So far, we have not discussed the vertical alignment of vertical boxes. TEX actually has three kinds of such boxes. The most common is the `\vbox` that we have been using. It creates an object whose *bottom edge* aligns with the baseline of the surrounding text. The second kind is the `\vcenter` box, which, for curious historical reasons, is restricted to math mode. It aligns a horizontal line passing through its center with the current baseline. The third kind is the `\vtop` box, which aligns the bottom edge of its first line with the baseline. We show shortly how one of these can be used to make a fourth kind of vertical box, with its top edge aligned with the baseline.

Here are two of the six extensions to our box commands that show how we can make the alternate vertical box alignments equally easy to use:

```
\def \LCbox #1#2#3#4%
{%                      left-adjusted text in \vcenter box
    \hbox to #1
    {%
        \parindent = 0pt
        \hsize = #1
        $
```

```
            \vcenter to #2
            {%
                \strut #3%
                \vfil
                \strut #4%
            }%
        $%
    }%
}
\def \LTbox #1#2#3#4%
{%                      left-adjusted text in \vtop box
    \hbox to #1
    {%
        \parindent = 0pt
        \hsize = #1
        \vtop to #2
        {%
            \strut #3%
            \vfil
            \strut #4%
        }%
    }%
}
```

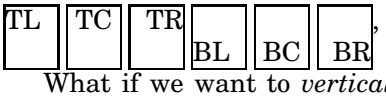
Definitions of the others should be obvious.

The input for the rest of this paragraph looks like this:

```
Here is how they can be used inside a paragraph as if
they were single words:
\fbox{\LCbox{20pt}{20pt}{TL}{BL}}
\fbox{\CCbox{20pt}{20pt}{TC}{BC}}
\fbox{\RCbox{20pt}{20pt}{TR}{BR}}
\fbox{\LCbox{20pt}{20pt}{TL}{}}
\fbox{\CCbox{20pt}{20pt}{TC}{}}
\fbox{\RCbox{20pt}{20pt}{TR}{}}
\fbox{\LCbox{20pt}{20pt}{}{BL}}
\fbox{\CCbox{20pt}{20pt}{}{BC}}
\fbox{\RCbox{20pt}{20pt}{}{BR}},
with more text following.
```

Here is how they can be used inside a paragraph as if they were single words: TL BL TC BC TR BR TL TC TR BL BC BR, with more text following.

Now change `Cbox` to `Tbox` and repeat that example. Here is how they can be used inside a paragraph as if they were single words: TL BL TC BC TR BR



, with more text following.

What if we want to *vertically center* a line of text inside one of these boxes? One solution is to make some new definitions of three-argument macros like this one:

```
\def \LBCbox #1#2#3%
{%      left-adjusted text in \vbox box
    \hbox to #1
    {%
        \parindent = 0pt
        \hsize = #1
        \vbox to #2
        {%
            \vfil
            \strut #3%
            \vfil
        }%
    }%
}
```

We can then use the \vbox-based macros in text like this:

\LBCbox{30pt}{20pt}{LC} 

,

\CBCbox{30pt}{20pt}{CC} 

, and

\RBCbox{30pt}{20pt}{RC} 

.

The \vtop-based macros produce text like this:

\LTCbox{30pt}{20pt}{LTC} 

,

\CTCbox{30pt}{20pt}{CTC} 

, and

\RTCbox{30pt}{20pt}{RTC} 

.

Notice that they hang *below* the baseline: the reason is that the contents of the vertical box start with fill glue, so TEX pretends that there is an invisible object of zero depth and height above the glue, and that object's baseline determines the top alignment.

The \vcenter-based macros produce text like this:

`\LCCbox{30pt}{20pt}{LCC}` 

,

`\CCCbox{30pt}{20pt}{CCC}` 

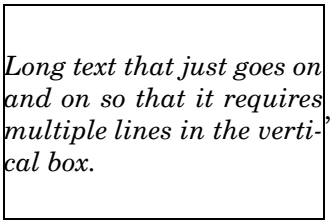
, and

`\RCCbox{30pt}{20pt}{RCC}` 

.

In all of the vertical box examples so far, the text arguments have been short enough to fit on one line, and the horizontal fills handled the positioning. What happens if they need multiple lines? The fills then apply to just the first and last lines, and we get peculiar text alignment. Thus, the input

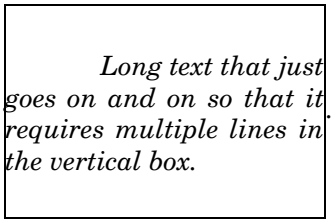
```
\fbox{\LCCbox{120pt}{80pt}{\it Long text...}}
```

produces 

,

and

```
\fbox{\CCCbox{120pt}{80pt}{\it Long text...}}
```

produces 

.

2.12 Page breaking with big vertical boxes

Typesetting of normal prose, such as in a literary novel, offers many opportunities for page breaking: the end of every line is a candidate.

Technical typesetting poses more challenges, since there are often objects of varying vertical extents, such as mathematical equations, tabular displays, and pictures, none of which can be broken across a page boundary. There may also be page headers, page footers, and footnotes, all of which must appear on the current page. TEX stores the latter, and objects defined with its *insert* mechanism, on separate lists, and leaves it up to the output routine to place them on the output page at suitable places. Large insertions are even more problematic, since their output may need to be delayed several pages until a suitable place for them is found. In the worst case, that place could be the end of the document.

If you create material with objects of large vertical extents, you must be prepared to help TEX at times. You might redesign a big table to be wider

than normal (as we do near the end of this document), or set it in a smaller type size, or break it into multiple tables. Manual rearrangement of such material may help, but sometimes, you have to rewrite the nearby prose, making it slightly longer or shorter to get things to fit. There are also low-level commands described in Chapter 15 of *The TeXbook* that allow you to tweak the line-breaking algorithm without changing the prose when your material almost, but not quite, fits a page.

3 Boxes and glue in LaTeX

The designers of LaTeX take a significantly different view of markup than the author of TeX does. The latter is an outstanding, and expert, programmer who relishes fine control over the typesetting process, and the results in his many books written since he developed TeX, and notably, later volumes of his monumental work called *The Art of Computer Programming*, are fine examples of what TeX and a careful author can produce.

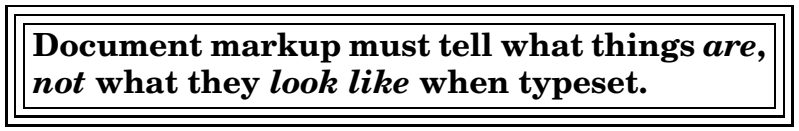
LaTeX is designed to allow an enormous variety of documents to be marked up for typesetting using much the same commands in each. Indeed, in some cases, simply by changing a *single* name in the LaTeX `\documentstyle` command, and a *single* name in the BibTeX `\bibliographystyle` command, and possibly also *single* names in `\usepackage` commands that select suitable font families for the document, one can obtain radically different typeset output, *without touching the rest of the document*. Of course, this lofty goal is often not immediately attainable without small tweaks. One reason is that a change of fonts alters TeX's line- and page-breaking decisions. The original document might have had no overfull boxes at all, whereas the new one may have many of them. This is invariably the case with narrow columns or long technical words, but happens even with the wide text widths of traditional one-column book publishing. Careful authors rewrite their prose to improve line breaks and page breaks.

Nevertheless, the goal is an admirable one, and it makes possible the development of other software that parses LaTeX syntax, and translates, or otherwise interprets it, for other purposes. Many publishers today that accept LaTeX submissions transform them in their production shops to new documents in the SGML or XML markup languages, which are painfully verbose for authors to write, but allow more reliable computer processing of the text.

As a simple example of such processing, consider the obvious grammar rule that document sectioning must be hierarchical and properly nested: a book contains chapters, which in turn contain sections, and those in turn may have subsections, and so on. It is grammatically incorrect to have a section or appendix before the first chapter, or a subsection outside of a section. LaTeX's markup commands for these textual objects do not prevent such abuses, since they just expand to TeX commands that give a particular visual appearance. However, an SGML parser follows the document

grammar rules, and rejects any document with such irregularities.

The LaTeX philosophy is simple, and worth decorating and boxing up in big bold text, since it is so different from most uses of plain TeX, and also from document production with word processors:



Consequently, the 1985 book *LaTeX: A Document Preparation System: User's Guide and Reference Manual* [4, 5], and dozens of other books on LaTeX written since then, omit discussion of almost all of the low-level TeX commands. Those books also generally avoid mention of boxes and glue, apart from brief mention of `\hfill`. Their expectation is that an author who needs, for example, to display text horizontally and vertically centered in a framed box should have a suitable command for doing so hidden away in a document style file or macro package.

The basic LaTeX command repertoire has several specialized commands for doing things like this, but also lacks many that users want. As a result, a few thousand macro packages have been written by LaTeX users around the world, and contributed to the *Comprehensive TeX Archive Network (CTAN)* collection for others to use, and if desired, modify for specific needs.

3.1 Horizontal boxes in LaTeX

LaTeX's analog of an unqualified `\hbox` is called `\mbox`. They are much the same thing, but `\mbox` is defined to be more widely usable. We have already used LaTeX's framed companion to `\mbox`, `\fbox`.

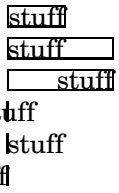
A horizontal box of specified width is provided in LaTeX with the command `\makebox[`*width*`][`*position*`]{contents}`. Bracketed command arguments in LaTeX are *always* optional. Here, the *width* is a TeX dimension, and defaults to the natural width of the contents if not given. The *position* is one of the letters `l` (flush left) or `r` (flush right); if it is omitted, the text is centered in the box. If the specified width is smaller than needed, the contents protrude from the box, and may overlap surrounding material. If the specified width is zero, then we have equivalents of the TeX `\rlap` and `\llap` commands.

Here are several examples of these three LaTeX box commands:

```
|\mbox{stuff}|                |stuff|
\fbox{stuff}                  stuff
|\makebox{stuff}|             |stuff|
|\makebox[40pt][l]{stuff}|    |stuff     |
|\makebox[40pt][r]{stuff}|    |     stuff|
|\makebox[0pt]{stuff}|       stuff
|\makebox[0pt][l]{stuff}|     |stuff
|\makebox[0pt][r]{stuff}| stuff|
```

The \makebox command has a framed companion, \framebox, with identical arguments. Like \fbox, \framebox creates a margin of width \fboxsep between the outline and the contents, but we continue with a zero value for that separation:

```
\framebox{stuff}
\framebox[40pt][l]{stuff}
\framebox[40pt][r]{stuff}
\framebox[0pt]{stuff}
\framebox[0pt][l]{stuff}
\framebox[0pt][r]{stuff}
```



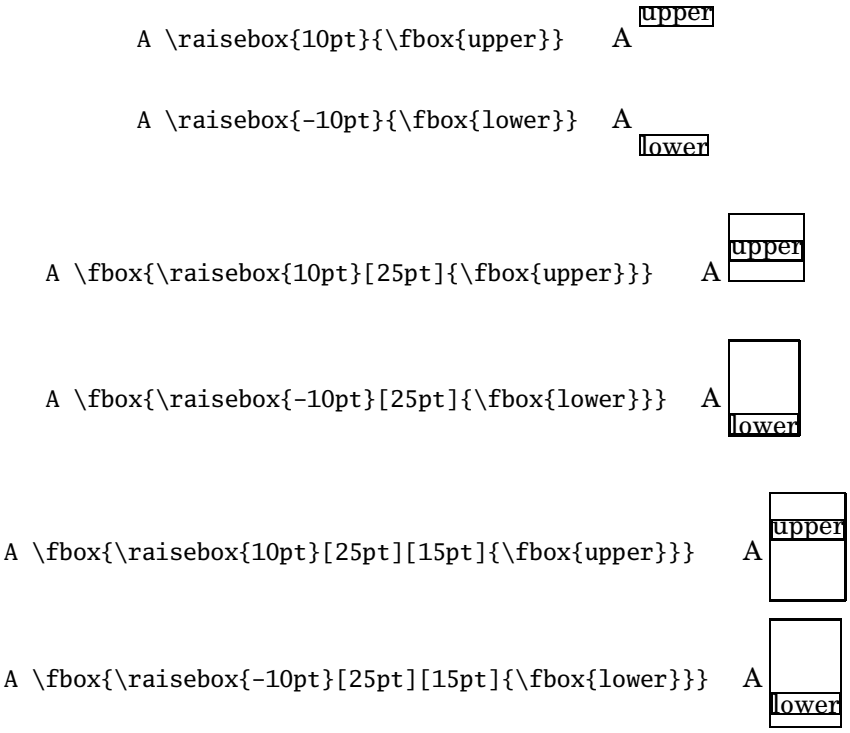
The last three examples show that the frame shrinks to a vertical bar when the box width is zero.

To help in positioning boxes within other objects, LaTeX provides a command to raise and lower boxes:

\raisebox{*raiselength*}[*height*][*depth*]{*contents*}

A negative first argument lowers the box. Here are some examples:

```
A \raisebox{10pt}{\fbox{upper}}
A \raisebox{-10pt}{\fbox{lower}}
A \fbox{\raisebox{10pt}[25pt]{\fbox{upper}}}
A \fbox{\raisebox{-10pt}[25pt]{\fbox{lower}}}
A \fbox{\raisebox{10pt}[25pt][15pt]{\fbox{upper}}}
A \fbox{\raisebox{-10pt}[25pt][15pt]{\fbox{lower}}}
```



For longer strings of text, LaTeX provides the paragraph box, which is defined like this: \parbox[*position*]{*width*}{*contents*}. The optional *position* is a letter b for alignment of the bottom line with the current baseline, or t for alignment of the top line with the surrounding baseline. Without

that argument, the box is centered vertically around the prevailing baseline. The box can be used as if it were a letter or a word, so we can put it in the middle of a sentence. The input

```
This is text \parbox{30pt}{\it and this is boxed text} and
this is more text.

This is text \fbox{\parbox{30pt}{\it and this is boxed text}}
and this is more text.
```

produces

This is text *and this is boxed text* and this is more text.

This is text *and this is boxed text* and this is more text.

With alignment arguments, we can produce these variants:

This is text *and this is boxed text with* t *alignment* and this is more text.

This is text *and this is boxed text with* b *alignment* and this is more text.

This is text *and this is boxed text with* t *alignment* and this is more text.

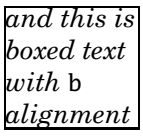
This is text *and this is boxed text with* b *alignment* and this is more text.

Flush-right typesetting generally looks bad in narrow columns, so we can insert a `\raggedright` command inside the last argument of the paragraph box to get output like this:

This is text *and this is boxed text with* t *alignment* and this is more text.

and this is
boxed text
with b
This is text *alignment* and this is more text.

This is text  and this is more text.



This is text and this is more text.

Another kind of paragraph box can be obtained in a more general, and more powerful, way with the minipage environment:

```
\begin{minipage}[position]{width}
    contents
\end{minipage}
```

The positioning works just like that for \parbox, with alignment letters b and t, and if they are omitted, a default of vertical centering.

In particular, verbatim text produced with the \verb command is illegal in macro arguments, so it cannot be used with \fbox, \framebox, \makebox, \mbox, or \parbox, but it *can* be used inside a minipage. The input

```
\begin{minipage}{170pt}
    This is inline verbatim \verb=\verb|\%{}|=, and this
    is a verbatim display:
    %
    \begin{verbatim}
      #include <stdio.h>
      #include <stdlib.h>
      int main(void)
      {
          printf("Hello, world\n");
          exit (EXIT_SUCCESS);
      }
    \end{verbatim}
\end{minipage}
```

produces this mid-sentence box This is inline verbatim \verb|\%{}|, and this is a verbatim display:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("Hello, world\n");
    exit (EXIT_SUCCESS);
}
```

and the sentence then continues normally.

The tables on 29 show two ways of setting tabular displays, using implicit and explicit paragraph boxes to control text positioning within each cell. They are set in this author's widecenter environment, which allows material to spill into both margins, remedying a curious defect of LaTeX's normal center environment.

3.2 Floats in LaTeX

We briefly discussed in section 2.12 on page 22 the problem that large objects present for page breaking. LaTeX handles figures and tables by a different mechanism than TeX does. LaTeX calls them *floats*, and like TeX's insertions, they are stored on lists that are separate from the main page galley, and then placed on output pages by the LaTeX output routine.

As in TeX, document tweaks may sometimes be needed to get LaTeX floats to appear close to the point of their first reference in the text. In rare cases, it may even be necessary to insert a \clearpage command at a suitable place to force a page break, and empty the list of pending floats. Such last resorts should *only* be used for the absolutely final version of a document, because effort spent on such fine tuning of earlier versions is simply wasted.

You can help LaTeX by designing your floats with small vertical extents, and using one or more of the float placement options, b (bottom), h (approximately here), ! (really here), or t (top), that appear in square brackets following the \begin{*floatclass*} command. You might also look at the documentation of the various parameters that control float placement. They are described in Appendix C of *LaTeX: A Document Preparation System: User's Guide and Reference Manual* [4, 5].

In large documents with many floats, it is useful in draft printings to mark references to floats with a boxed note in the margin. That way, you can quickly check that the references are near their objects. This author uses definitions like these for that purpose:

```
\RequirePackage[dvips]{color}
\RequirePackage{varioref}       % for \vref and \vpageref

\newcommand{\figlabel}[1]{\label{fig:#1}}
```

Table 1: Wide table set with p-style column formatting in the LaTeX `tabular` environment.

Book	Description	Publisher
The TeXbook [2]	a book about high-quality typesetting, particularly for mathematical and technical material	Addison–Wesley
The Advanced TeXbook [7]	a book that takes the reader into the depths of TeX; it has particularly thorough treatment of the subject of TeX output routines, a topic that receives relatively little attention in *The TeXbook*	Springer-Verlag
LaTeX: A Document Preparation System: User's Guide and Reference Manual [4, 5]	a book about a document-independent logical markup system that is built on top of the TeX typesetting system	Addison–Wesley
The LaTeX Companion [1, 6]	a book that covers some of the most important LaTeX packages for color, font selection, graphics, and specialized typesetting	Addison–Wesley

Table 2: Wide table set with `l`-style column formatting in the LaTeX `tabular` environment. Each cell of data is set with a `\parbox` command, without a positioning argument, so that the cell paragraph is centered vertically.

Book	Description	Publisher
The TeXbook [2]	a book about high-quality typesetting, particularly for mathematical and technical material	Addison–Wesley
The Advanced TeXbook [7]	a book that takes the reader into the depths of TeX; it has particularly thorough treatment of the subject of TeX output routines, a topic that receives relatively little attention in *The TeXbook*	Springer-Verlag
LaTeX: A Document Preparation System: User's Guide and Reference Manual [4, 5]	a book about a document-independent logical markup system that is built on top of the TeX typesetting system	Addison–Wesley
The LaTeX Companion [1, 6]	a book that covers some of the most important LaTeX packages for color, font selection, graphics, and specialized typesetting	Addison–Wesley

```
\newcommand{\figref}[1]{Figure~\ref{fig:#1}
                        \vpageref{fig:#1}\markref{figure}}

\newcommand{\shortfigref}[1]{Figure~\ref{fig:#1}\markref{figure}}

\newcommand{\markref}[1]{%
                          \marginpar{\fcolorbox{black}{yellow}%
                                    {\small \bfseries #1 xref}}%
                        }
```

The varioref package provides a way to refer to a labeled object with a numbered page reference when the object is far away, or with a phrase like *on the next page*, or without a page number when it is on the same page. Using separate labeling commands, and label namespaces, for different object classes is a good idea when there are many labels, since it reduces the risk of using the wrong label.

When the final version is ready for printing, the marginal notes can easily be removed by adding this empty definition at the end of the document's private style file:

```
\renewcommand{\markref}[1]{}
```

Some documents need more kinds of floats than just the figures and tables that LaTeX provides by defaults. For example, a book about computer programming is likely to have many sample programs that could usefully be displayed as program floats. The LaTeX `\newfloat` command is the key to defining a new class of floats, but it takes a bit more work than might be expected from its documentation. Here is an example from a book on programming whose private style file gives each program float a background color and an index entry:

```
\RequirePackage[dvips]{color}
\RequirePackage{coloralias}         % for indirect color names
\RequirePackage{rgb}                % for color names

\definecoloralias{programcolor}{azure}

%%% Define a private float style that allows us to control where the
%%% caption goes, and what font is used in the caption name and
%%% number:

\newcommand{\fs@pgmplaintop}
{%
    \fs@plain % we share much of the setup of plain
    \def \@fs@mid {\vspace\belowcaptionskip\relax}%
    \let \@fs@iftopcapt = \iftrue
    %% We want other floats to have bold names and numbers too; the
    %% default in plaintop puts them in rmfamily (float.sty:143).
```

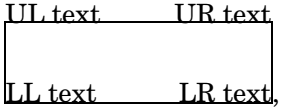
```
    \def \@fs@cfont {\bfseries}%
}

%%% When a float comes in the middle of a colored text display, it
%%% incorrectly inherits the text color of that display.  Redefine the
%%% float caption macro to reset the text color.

\setlength {\belowcaptionskip} {6pt} % was 0pt, but that seems tight

\renewcommand{\floatc@plain}[2]
{%
    {% group to restrict color change
        \color{textcolor}%
        \setbox \@tempboxa = \hbox{{\@fs@cfont #1:} #2}%
        \ifdim \wd \@tempboxa > \hsize
            {\@fs@cfont #1:} #2\par
        \else
            \hbox to \hsize {\hfil \box \@tempboxa \hfil}%
        \fi
        \smallskip % extra space below caption
    }%
}

%%% Contrary to documentation, floatstyle must be set BEFORE newfloat;
%%% otherwise, captions go at their default bottom position.  Except
%%% for figures, I want captions at the top of all floats.

\floatstyle{pgmplaintop}

%%% Programs are a new float object, and create a \jobname.lop file
%%% that is read and typeset in the front matter.  The float name
%%% "Program" is reused in the front matter, and in the captions, so
%%% it cannot be arbitrary.

\newfloat{Program}{htb!p}{lop}[chapter]

%% avoid "Package hyperref Warning: bookmark level for unknown
%% Program defaults to 0"
\providecommand*{\toclevel@Program}{0}

%%% Usage: \programlisting{caption}{body}
\newcommand{\programlisting}[2]
{%
    \begin{Program}%
        \vrule width \textwidth height 0.2ex
        %% We want other floats to have bold names and numbers too; the
        %% default in plaintop puts them in rmfamily (float.sty:143).
        \renewcommand {\@fs@cfont}{\bfseries}%
        \caption{#1}%
        \begin{small}%
```

```
            \noindent
            \index{program box}%
            \colorbox{programcolor}{\parbox{0.983\textwidth}{#2}}%
        \end{small}
        \vrule width \textwidth height 0.2ex
        \par
    \end{Program}%
}
```

3.3 Boxes in picture environments

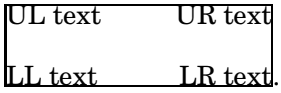
Neither TeX nor LaTeX provides an easy mechanism for referring to an absolute page position. Indeed, such a position is not known until the page is ready to be shipped out to the DVI file, and thus, commands for absolute positions would have to be saved, and then executed in the output routine. However, relative positioning is possible in LaTeX with the help of the `picture` environment and the `\put` command. The input

```
\fbox{%
          \setlength{\unitlength}{1pt}%
          \begin{picture}(100,30)(0,0)
              \put(  0,  0){LL text}
              \put(100,  0){\makebox[0pt][r]{LR text}}
              \put(100, 30){\makebox[0pt][r]{UR text}}
              \put(  0, 30){UL text}
          \end{picture}%
      }
```



produces the output , but the labels are not quite where we want them. We therefore adjust their coordinates slightly, and retry with

```
\fbox{%
          \setlength{\unitlength}{1pt}%
          \begin{picture}(100,30)(0,0)
              \put(  0,  0){LL text}
              \put(100,  0){\makebox[0pt][r]{LR text}}
              \put(100, 23){\makebox[0pt][r]{UR text}}
              \put(  0, 23){UL text}
          \end{picture}%
       }
```

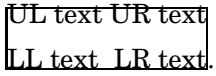


to produce .

Although working out suitable coordinate positions is tedious, it allows us finer positioning control than we have with the boxing macros that we developed for plain TeX in section 2.10 on page 17.

One further advantage of this approach is that we can easily resize the box just by refining the \unitlength. We retry with

```
\fbox{%
          \setlength{\unitlength}{0.75pt}%
          \begin{picture}(100,30)(0,0)
              \put(  0,  0){LL text}
              \put(100,  0){\makebox[0pt][r]{LR text}}
              \put(100, 23){\makebox[0pt][r]{UR text}}
              \put(  0, 23){UL text}
          \end{picture}%
      }
```



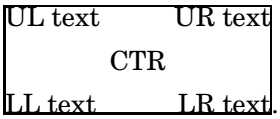
to produce . This example shows that we really do need some helper macros that allow text boxes to be positioned by any of their four corners, or their centers.

To find a proper solution to this problem, we need to use some low-level TeX commands for measuring the height of boxes containing arbitrary text, and then create boxes whose reference point is one of the four corners, or the box center:

```
\newcommand{\LLBox}[1]{\setbox1 = \hbox{#1}%
                       \raisebox{\dp1}{\makebox[0pt][l]{#1}}}
\newcommand{\LRBox}[1]{\setbox1 = \hbox{#1}%
                       \raisebox{\dp1}{\makebox[0pt][r]{#1}}}
\newcommand{\ULBox}[1]{\setbox1 = \hbox{#1}%
                       \raisebox{-\ht1}{\makebox[0pt][l]{#1}}}
\newcommand{\URBox}[1]{\setbox1 = \hbox{#1}%
                       \raisebox{-\ht1}{\makebox[0pt][r]{#1}}}
\newcommand{\CCBox}[1]{\setbox1 = \hbox{#1}%
                       \raisebox{-0.5\ht1}{\makebox[0pt]{#1}}}
```

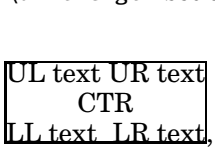
We can then easily use this input

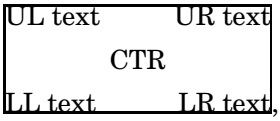
```
\fbox{%
          \begin{picture}(100,40)(0,0)
              \put(  0, 0){\LLBox{LL text}}
              \put(100, 0){\LRBox{LR text}}
              \put(50, 20){\CCBox{CTR}}
              \put(  0,40){\ULBox{UL text}}
              \put(100,40){\URBox{UR text}}
          \end{picture}%
      }.
```

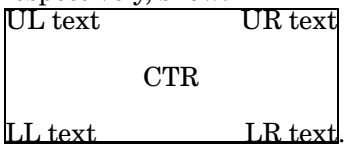


to produce this output:

Our final macros properly handle scaling, as these experiments with `\unitlength` set to 0.75 pt, 1 pt, and 1.25 pt, respectively, show:

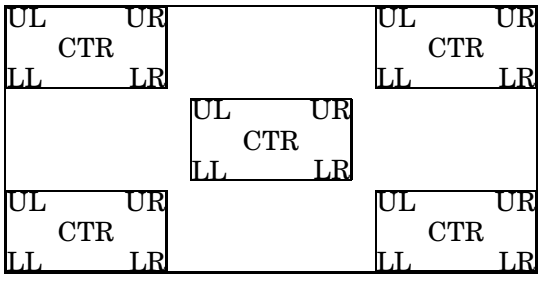






We can use them to position arbitrary nested text as well. The input

```
\newcommand{\samplebox}{%
\fbox{%
          \setlength{\unitlength}{0.75pt}%
          \begin{picture}(80,40)(0,0)
              \put( 0, 0){\LLBox{LL}}
              \put(80, 0){\LRBox{LR}}
              \put(40 20){\CCBox{CTR}}
              \put( 0,40){\ULBox{UL}}
              \put(80,40){\URBox{UR}}
          \end{picture}%
      }%
}

\fbox{%
          \setlength{\unitlength}{2.5pt}%
          \begin{picture}(80,40)(0,0)
              \put( 0, 0){\LLBox{\samplebox}}
              \put(80, 0){\LRBox{\samplebox}}
              \put(40,20){\CCBox{\samplebox}}
              \put( 0,40){\ULBox{\samplebox}}
              \put(80,40){\URBox{\samplebox}}
          \end{picture}%
}
```



produces this output:

We can do similar displays with mathematical material. The input

```
\fbox
```

```
{%
    \setlength{\unitlength}{2pt}%
    \begin{picture}(120,40)(0,0)
        \put(  0, 0)
            {\LLBox{$\displaystyle e = \prod_{n=1}^{100}\sin(n)$}}
        \put(120, 0){\LRBox{$\displaystyle f(x) = \left \{
                                \begin{array}{ll}
                                  g & \quad \textrm{if $x < 0$}\\
                                  h & \quad \textrm{if $x = 0$}\\
                                  i & \quad \textrm{if $x > 0$}
                                \end{array}
                                \right . $}}}
        \put( 60,20){\CCBox{$\bullet$}}
        \put(  0,40){\ULBox{$a = b + c$}}
        \put(120,40)
            {\URBox{$\displaystyle d = \sum_{n=1}^{\infty}n^{-2}$}}
    \end{picture}%
}%
```

produces this output:

$$a = b + c \qquad\qquad d = \sum_{n=1}^{\infty} n^{-2}$$

$$\bullet$$

$$e = \prod_{n=1}^{100} \sin(n) \qquad\qquad f(x) = \begin{cases} g & \text{if } x < 0 \\ h & \text{if } x = 0 \\ i & \text{if } x > 0 \end{cases}$$

Plain TEX does not have an analogue of the LATEX `picture` environment, but page 389 of *The TEXbook* sketches how one might get started on its design.

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1994.

[2] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, MA, USA, 1984.

[3] Donald E. Knuth. A simple program whose proof isn't. In W. H. J. Feijen, A. J. M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our business: a birthday salute to Edsger W. Dijkstra*, chapter 27, pages 233–242. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1990. This paper discusses the algorithm used in TEX for converting between decimal and scaled fixed-point binary values, and for guaranteeing a minimum number of digits in the decimal representation.

[4] Leslie Lamport. *LaTeX—A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985.

[5] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994.

[6] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004.

[7] David Salomon. *The Advanced TeXbook*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1995.