TEXNICAL PROGRAMMING

Slurp
February, 2023

- **TEX engines and formats: TEX vs LATEX**
- **Typesetting in plain TEX**
- **Macros and primitives**
- **Repeated macros**
- **Lists**

TEX ENGINES AND FORMATS

TEX refers mainly to three things:

- The TEX language, a typesetting language made up of *primitives* like `\def` and `\hskip`.
- The TEX engine (`tex`), a program for compiling files written in the TEX language.
- The plain TEX format, a bare-bones TEX format.

TEX formats are self-contained macro packages whose intention it is to define macros for the user's use in creating documents.

Some well-known formats are plain TEX, LATEX, and ConTEXt.

Macro packages are not considered formats since they are not self-contained and their intention is to provide further abstraction on top of an existing format.

The format can be specified during the invocation of the `tex` command on the command line.

TEX engines are programs which compile TEX code into output. Historically this output was originally a `dvi` file, but nowadays most output is a `pdf` file.

Some well-known formats are ε-TEX, pdfTEX, LuaTEX, and XETEX.

Engines tend to add primitives to the base TEX language: for example, ε-TEX adds the `\numexpr` and related primitives, pdfTEX adds the `\pdfliteral` primitive among others.

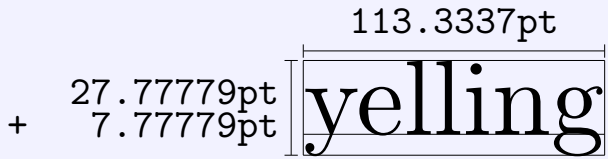
Note that invoking the program engine in the command line by itself also specifies the format: `pdftex` runs the pdfTEXengine with the plain TEX format, while `pdflatex` runs the pdfTEXengine with the LATEX format.

Questions?

Typesetting in Plain TEX

TEX creates its pages by creating a vertical list and filling it with boxes, glue, leaders, penalties, kerns, etc. There are two types of boxes, horizontal boxes and vertical boxes.

Boxes have width, height, and depth. The height of a box is visually its positive vertical displacement relative to the baseline, and its depth is its negative vertical displacement relative to the baseline. For example:



\hbox

`\hbox to/spread` ⟨*dimension*⟩`{`⟨*horizontal material*⟩`}`: Creates a horizontal box, if *to dimension* is specified then the width of the box is *dimension*.

If *spread dimension* is specified then the width of the box is *dimension* more than its natural dimension.

The height and depth of the box is equal to the height and depth of its contents.

\vbox

`\vbox to/spread` ⟨*dimension*⟩`{`⟨*horizontal material*⟩`}`: Creates a vertical box, if *to dimension* is specified then the height of the box is *dimension*.

If *spread dimension* is specified then the height of the box is *dimension* more than its natural dimension.

The width of the box is the width of its contents, and the depth of the box is the depth of the final box in it.

The box's baseline is the same as that of the final box inside, to align to the top box, use `\vtop`.

Between two boxes there is something called *glue*, which connects the boxes. Glue is one type of blank space (the other being a kern).

Glue has three attributes: its natural length, its maximum stretchiness, and its maximum shrinkage. Glue stretches and shrinks only when it needs to, and TEX uses these attributes in order to fit material into widths the material couldn't properly fit into in its natural width.

For example:

```
1  \hbox to 5cm{hello\hskip 3cm plus 2cm}there
```

Creates

hello there

Without the `plus2cm` we'd get the same output but with an overfull hbox warning.

The amount of stretchiness and shrinkage can be infinite.

All the dimensions may be negative as well.

`\hskip` `\hskip` ⟨*natural length*⟩ `plus` ⟨*stretch*⟩ `minus` ⟨*shrink*⟩: Adds horizontal glue with the specified natural length, maximal stretch and shrinkage.

The stretch and shrink are optional.

`\vskip` `\vskip` ⟨*natural length*⟩ `plus` ⟨*stretch*⟩ `minus` ⟨*shrink*⟩: Adds vertical glue with the specified natural length, maximal stretch and shrinkage.

The stretch and shrink are optional.

`\kern` `\kern` ⟨*dimension*⟩: Adds a kern whose dimension is *dimension*. Kerns, unlike glue are nonbreaking, nonstretching, and nonshrinking. The orientation of the kern (horizontal or vertical) is inferred by the context.

TEX has 3 orders of infinities for glue stretching:

`fil` — First order `fil`: `\hskip 0pt plus 1fil\relax` creates glue which has no natural length but has infinite stretchiness. A primitive version, `\hfil`, exists as well in place of the code above.

`fill` — Second order `fill`: `\hskip 0pt plus 1fill\relax` creates glue which also has no natural length and infinite stretchiness. It takes precedent over first order infinities. A primitive version, `\hfill`, exists as well.

`filll` — Third order `filll`: Same as the other two, but takes precedent over both of them. No primitive version exists.

Vertical versions of `\hfil` and `\hfill` exist, `\vfil` and `\vfill`.

Another important primitive is `\hss` which can both shrink and stretch infinitely. It is analogous to `\hskip 0pt plus 1fil minus 1fil`.

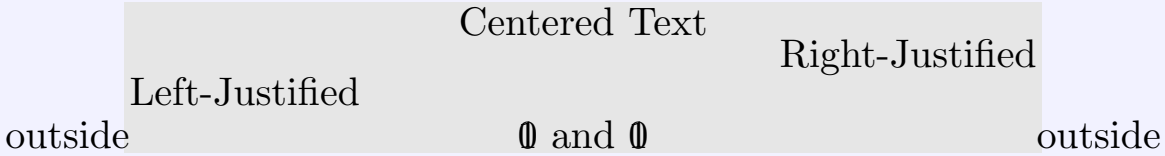
It too has a vertical version `\vss`.

```
\def\line{\hbox to \hsize}
\def\centerline#1{\line{\hfil#1\hfil}
\def\rightline #1{\line{\hfil#1}}
\def\leftline  #1{\line{#1\hfil}
\def\rlap#1{\hbox to 0pt{#1\hss}}
\def\llap#1{\hbox to 0pt{\hss#1}}
```

`\line` creates a box which spans the entire line.
`\centerline` centers input relative to the line.
`\rightline` and `\leftline` right and left-justify input respectively.
`\rlap` typesets input and then seems to move back as if it hadn't been typeset.
`\llap` moves back the width of its material and then typesets it.

```
\centerline{Centered Text}
\rightline{Right-Justified}
\leftline{Left-Justified}

\quitvmode\llap{outside}\hfill1\llap{0} and \rlap{1}0
    \hfill\rlap{outside}
```



\hrule

`\hrule width` ⟨*dimen*⟩ `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩

Creates a horizontal rule (a horizontal line). All of the dimensions are optional. If the width is not specified, then the width spans the width of the smallest box enclosing the rule. If the height is not specified, it is by default 0.4pt. If the depth is not specified, it is by default 0pt.

This must be used in vertical mode (not horizontal mode, it is a horizontal rule because it is generally used to make horizontal lines in vertical mode).

\vrule

`\vrule width` ⟨*dimen*⟩ `height` ⟨*dimen*⟩ `depth` ⟨*dimen*⟩

Creates a vertical rule (a vertical line). All of the dimensions are optional. If the width is not specified, it is by default 0.4pt. If the height or the depth are not specified, they span the height or depth of the smallest box enclosing the rule.

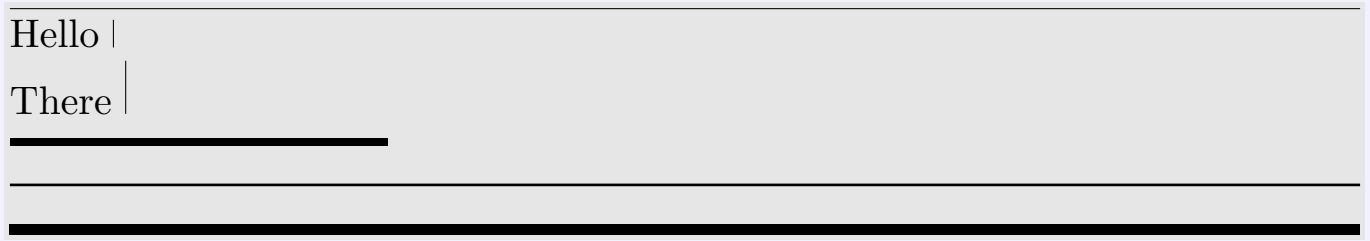
This must be used in vertical mode.

```
\hrule
Hello \vrule

There \vrule \vbox to 20pt{}

\vbox{\line{}\hrule height 3pt}

\vskip.5cm
\hrule height 1pt
\vskip.5cm
\hrule height 2pt
\hrule height 2pt
```



```
\def\boxed#1{\vbox{\hrule \hbox{\vrule #1\vrule}\hrule}}
\def\badbox#1{\hbox{\vrule \vbox{\hrule #1\hrule}\vrule}}

\boxed{Typesetting in plain \TeX}

\vskip.5cm
\badbox{Typesetting in plain \TeX}
```

Typesetting in plain TEX

Typesetting in plain TEX

```
\def\spas{3pt}
\def\spacebox#1{%
    \hbox{%
        \vrule%
        \vbox{%
            \hrule \vskip\spas%
            \hbox{\hskip\spas #1\hskip\spas}%
            \vskip\spas \hrule%
        }%
        \vrule%
    }%
}

\spacebox{Typesetting in plain \TeX}
```

Typesetting in plain TEX

TEX has box registers which you can use to store boxes. The number of box registers depends on which engine you use (the original TEX engine had 256 from `\box0` to `\box255`).

`\newbox` `\newbox`⟨*control sequence*⟩: Allocates a box and sets `control sequence` to reference this box.

`\setbox` `\setbox`⟨*box number*⟩ `=` ⟨*box*⟩: Sets the `box number`-th box register to be equal to `box`.

`\box` `\box`⟨*box number*⟩: uses the `box number` box register. After using this, the box register is emptied.

`\copy` `\copy`⟨*box number*⟩: like `\box` but the register is not emptied after its use.

`\unhbox` `\unhbox`⟨*box number*⟩: uses the `box number` box register and removes a level of boxing.

```
1  \setbox0=\hbox{A} \setbox1=\hbox{\unhbox0 B}
```

Makes `\box1` equal to `\hbox{AB}`.

The box register must be a horizontal box (`\hbox`). For vertical boxes, use `\unvbox`. There also exist `\unhcopy` and `\unvcopy`.

You can access the width, height, and depth of a box register using `\wd`, `\ht`, and `\dp`. The use being `\wd`⟨*box register*⟩.

In order to print a TEX variable (like the dimensions of a box), you must prepend it with the primitive `\the`.

```
1  \setbox0=\hbox{Typesetting in plain \TeX}
2  Width: \the\wd0, Height: \the\ht0, Depth: \the\dp0
```

Width: 165.91646pt, Height: 10.41666pt, Depth: 3.22916pt

Also, notice that something like

```
1  \hbox{A}B
```

Gives

A
B

This is because when TEX boxes `\hbox{A}` it is in vertical mode, and so the box is placed into the vertical list. Then when it reads `B`, it enters horizontal mode, then boxes the horizontal material (which is just the `B` here) and places that into the vertical list.

So the vertical list looks like

```
\vbox{
    \hbox{A}
    \hbox{B}
}
```

So what we'd want to do is exit vertical mode before the `\hbox`, so its added to the horizontal list instead of the vertical one. Ie. we want the vertical list to look like:

```
\vbox{
    \hbox{\hbox{A}B}
}
```

In order to enter horizontal mode, we need to add horizontal material.

We can do this by unhboxing a void register, since `\unhbox` adds horizontal material.

TEX provides a box register which is meant to be always void, `\voidb@x`, so you can do

```
1  \unhbox\voidb@x\hbox{A}B
```

```
AB
```

TEX also provides the macro `\leavevmode` which is short for this.

pdfTEX provides the primitive `\quitvmode` which achieves the same thing.

```
\def\presentbox#1{{%
    \setbox0=\hbox{#1}%
    \hbox{%
        \vbox{\tt%
            \hbox{Width: \the\wd0}%
            \hbox{Height: \the\ht0}%
            \hbox{Depth: \the\dp0}%
        }%
        \quad\boxed{%
            \rlap{#1}%
            \vrule width\wd0 height .5pt depth 0pt%
        }%
    }%
}}

\presentbox{Typesetting in plain \TeX}
```

```
Width:   165.91646pt
Height:   10.41666pt
Depth:   3.22916pt
```

Typesetting in plain TEX

```
1  \def\centersym#1#2{\quitvmode{%
2      \setbox0=\hbox{#1}%
3      \rlap{#1}\hbox to \wd0{\hss#2\hss}%
4  }}
5
6  \centersym{$\bigcup$}{$\cdot$}
```

Leaders are a generalization of TEX's concept of glue. The purpose of leaders are to *lead* your eyes across the page.

This . . . . . is an example of leaders.

The general use of leaders is

`\leaders`⟨*box or rule*⟩⟨*glue*⟩

For example the above leaders was created by

```
\leaders\hbox to 1em{\hss.\hss}\hfill
```

Note that if a line ends with leaders or glue, it is removed (it doesn't really make sense to end a line with a blank space). So the following code:

```
\quitvmode\leaders\hbox to 1em{\hss.\hss}\hfill

Next line
```

yields:

Next line

There is no leaders here because the paragraph ends with leaders. In order to get around this you can simply place an empty box, or an empty kern after the leaders.

Notice the `\quitvmode` before the `\leaders`, this is necessary because `\leaders` works in vertical mode as well, but its glue must be vertical glue (ie `\vskip`, `\vfill`, etc).

`\leaders`

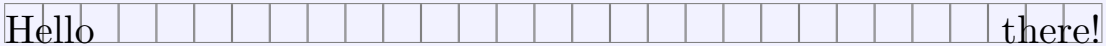
`\leaders` places leaders aligned with (what seems like) an infinite grid of boxes (either aligned with a row or column of the grid, depending on the horizontal/vertical context).

This grid of boxes has the width/height of the box/rule in the leaders, and starts left aligned with the smallest enclosing box of the material.

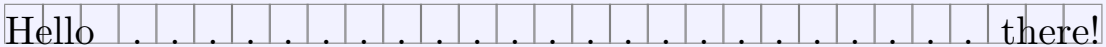
`\leaders` starts placing the leaders at the first box fully enclosed in the available space, and continues to the last available box (which is placed according to the input glue)

So for example doing something like

```
Hello\leaders\hbox to.5cm{\hss.\hss}\hfill there!
```



So the first box is placed in the 4th box (the first box which isn't occupied), and the last in the 4th to last (the last box which isn't occupied).



`\cleaders` The process here is simpler, the boxes or rules are all packed tightly next to each other and an equal amount of glue is placed on either ends of the leaders:

leaders : . . . . . end
cleaders: . . . . . end

`\xleaders` The boxes or rules here are all spaced equally apart. If there are q boxes/rules, then an equal amount of glue is placed in the $q + 1$ spots between/around them.

leaders : . . . . . end
xleaders: . . . . . end

Notice that while the placement of the boxes with `\leaders` is independent of the surrounding material, so multiple `\leaders` will have aligned boxes:

Typesetting in plain TEX . . . . . .
Typesetting in plain . . . . . TEX
Typesetting in . . . . . plain TEX
Typesetting . . . . . in plain TEX
. . . . . Typesetting in plain TEX

While `\cleaders` and `\xleaders` both are:

Typesetting in plain TEX . . . . . .
Typesetting in plain . . . . . TEX
Typesetting in . . . . . plain TEX
Typesetting . . . . . in plain TEX
. . . . . Typesetting in plain TEX

An important part of TEX is the concept of *alignment*.

In LATEX this takes the form of environments like `tabular` and `array`, but alignment is even more powerful.

The main TEX primitive which allows for alignment is `\halign`.

The input of `\halign` comes in two parts: the preamble and the actual alignment material. The preamble of an alignment dictates how to align the material, it is simply a list of what material to place around the alignment material.

```
\tabskip=1cm
\halign{#\hfil&\hfil#\hfil&\hfil#\cr
    Left Aligned&Center Aligned&Right Aligned\cr
    Typesetting&in plain&\TeX\cr
}
```

Left Aligned	Center Aligned	Right Aligned
Typesetting	in plain	TEX

Notice here that as opposed to LATEX where `\\` delimits the rows in an alignment, `\halign` uses `\cr` (carriage return).

Here the preamble is `#\hfil&\hfil#\hfil&\hfil#`

This means for the first entry of each row, the template is `#\hfil`, this right aligns it. What's significant about this is that the width of each entry is the maximum width of the all the entries in the column.
Similarly the second and third templates center and left-align their entries respectively.

`\tabskip` is the glue added between every column (as well as before the first and after the last column).

`\tabskip` can also be altered within the preamble of an alignment.

The glue inserted before the first column is equal to the value of `\tabskip` when `\halign` is called, and subsequent changes of `\tabskip` within the preamble of the alignment alter the glue inserted after the current and subsequent columns.

```
\tabskip=0pt
\halign{#\hfil\tabskip=1cm&\hfil#\hfil&\hfil#\tabskip=0pt\cr
    Left Aligned&Center Aligned&Right Aligned\cr
    Typesetting&in plain&\TeX\cr
}
```

Left Aligned	Center Aligned	Right Aligned
Typesetting	in plain	TEX

Left Aligned	Center Aligned	Right Aligned
Typesetting	in plain	TEX

For comparison, above is the output of the previous alignment.

plain TEX provides the macro `\ialign` which sets `\tabskip` to 0 and calls `\halign`.

We can also set the width of `\halign` similar to an `\hbox` via

`\halign to` ⟨*width*⟩`{...}`

Instead of `\cr`, we can use `\crcr` which does the same thing, but if `\crcr` comes after another `\cr`, it does nothing.

Thus we can create a macro which is similar to LATEX's `align` environment

```
\def\align#1{%
    \tabskip=0pt plus 1fil\relax%
    \halign to \hsize{%
        \hfil$\displaystyle##{}$\tabskip=0pt&%
        $\displaystyle{}##$\hfil\tabskip=0pt plus 1fil\cr%
        #1\crcr%
    }%
}

\align{
    \sum_{n=1}^\infty a_n &= 1\cr
    n! &= \prod_{i=1}^n i\cr
}
```

$$\sum_{n=1}^{\infty} a_n = 1$$

$$n! = \prod_{i=1}^{n} i$$

We can use the **\openup** macro to change the amount of glue added between lines (to "open up" the lines). This is used mostly in the context of alignment.

```
1  First paragraph
2
3  {\openup2\jot\halign{#\cr Hello\cr There\cr}}
4
5  Second paragraph
6
7  \halign{#\cr Hello\cr There\cr}
8
9  Third paragraph
```

First paragraph

Hello

There

Second paragraph

Hello

There

Third paragraph

`\jot` is simply a dimension (set equal to 3pt by plain TEX), it is customary to use `\openup` in terms of `\jot`s.

`\openup` is cumulative: `\openup1\jot\openup-1\jot`, has the same effect as if nothing had been done.

`\openup` changes the amount of glue added between lines, so it is necessary to keep its changes local by placing it inside a group (`{...}`).

Tokens inside the preamble of an alignment are not expanded, unless they are preceded by the `\span` primitive, which expands (ex-span-ds) the next token.

`\span` has an entirely different meaning within the alignment material, it takes the place of `&` and merges the two entries into one box whose width matches with the width (including the inter-column glue) of the columns.

Within an alignment entry `\omit` omits the current template, and instead the entry uses the trivial template `#`.

Thus `\span` and `\omit` can be used in conjunction for cells which span multiple columns.

Another useful primitive is `\noalign{`⟨*vertical material*⟩`}` which can come after `\cr` and adds *vertical material* in place of interline glue.

```
{\tabskip=0pt
\openup1\jot
\halign{#\hfil\tabskip=5pt&\hfil#\tabskip=0pt\cr
\noalign{\hrule\vskip3pt}
\omit\span\omit\hfil Alignment Example\hfil\cr
\noalign{\vskip3pt\hrule\vskip\lineskip}
Typesetting&in\cr
plain&\TeX\cr}}
```

Alignment Example	
Typesetting	in
plain	TEX

If you add a `&` before a template in an alignment's preamble, it is as if that template and all subsequent templates are repeated infinitely.

So for example the preamble

```
\hfil#&&\hfil#&#\hfil
```

Creates an alignment where the first template right-aligns, so does the next, the one after left-aligns, then right-aligns, then left-aligns, and so on (`RRLRLRL...`)

Finally, there is another alignment primitive `\valign` which does its alignment vertically; instead of aligning by rows, it aligns by columns.

The entries in `\valign` are in vertical mode. It is much less common than `\halign`.

Finally (just kidding, we have a lot more to go), let's bring together some of the concepts we've just covered and revisit our `\spacebox` macro.

We can simplify it using alignment:

```
\def\spas{3pt}
\def\alignedbox#1{%
    \vbox{\offinterlineskip%
        \tabskip=0pt%
        \halign{##\tabskip=\spas&###&###\tabskip=0pt\cr
            \noalign{\hrule}
            \vrule&%
            \tabskip=\spas\valign{##\cr\hbox{#1}\cr}&%
            \vrule\cr
            \noalign{\hrule}
        }%
    }%
}

\alignedbox{Typesetting in plain \TeX}
```

Typesetting in plain TEX

Just kidding, that wasn't simpler.

But alignment is more versatile. Instead of boring rules, we will use leaders to create patterned lines, for example:

```
\def\hdotsline{\xleaders\hbox to 3pt{\hss.\hss}\hfil}
\def\vdotsline{\xleaders\vbox to 3pt{\vss\hbox{.}\vss}\vfil}
\def\dotbox#1{%
    \vbox{\offinterlineskip%
        \tabskip=0pt%
        \halign{##\cr
            \noalign{\hdotsline}
            \valign{##\cr
                \vdotsline\cr
                \hbox{#1}\cr
                \vdotsline\cr
            }\cr
            \noalign{\hdotsline}
        }%
    }%
}

\dotbox{Typesetting in plain \TeX}
```

Typesetting in plain TEX

The primary use of `\valign` here is that it puts its entries in vertical mode whose height is the maximum height in the row.

`\dotbox` doesn't add space around the text, doing so isn't terribly hard though.

TEX utilizes many many registers to control the look of its output. Among them are some registers containing glue and dimensions which controls the space around paragraphs and lines.

`\parindent` The width of indentation before the first line of a paragraph. Plain TEX sets this to 20pt. `\parindent` is a dimension.

`\parskip` The inter-paragraph glue. Plain TEX sets this to `0pt plus 1pt`.

`\leftskip` The glue added to the left of every line. Plain TEX sets this to `0pt`.

`\rightskip` The glue added to the right of every line. Plain TEX sets this to `0pt`.

For example, if we'd like to center a paragraph we could set the left and right glue to have infinite stretchability.

```
{\parindent=0pt
\leftskip=0pt plus 1fill \rightskip=0pt plus 1fill\relax
It is not the plan of this essay to discuss the
millennium-old problem of faith and reason. Theory is not
my concern at the moment. I want to instead focus attention
on a human-life situation in which the man of faith as an
individual concrete being, with his cares and hopes,
concerns and needs, joys and sad moments, is entangled.
\par
}
```

It is not the plan of this essay to discuss the millennium-old problem of faith and reason. Theory is not my concern at the moment. I want to instead focus attention on a human-life situation in which the man of faith as an individual concrete being, with his cares and hopes, concerns and needs, joys and sad moments, is entangled.

Notice that `\par` (which ends the current paragraph) must be in the same group as the paragraph. Otherwise the paragraph would be ended outside the group once `\leftskip` and `\rightskip` have already reverted back to their original values.

TEX also provides registers for further altering the shape of a paragraph with `\hangindent` and `\hangafter`.

`\hangindent` is a dimension which specifies the dimension of the "hanging indentation".

`\hangafter` is a number which specifies which lines will be indented.

Suppose `\hangafter` is n, if $n \geq 0$ then the indented lines are $n+1$, $n+2$, and so on until the end of the paragraph.

If $n < 0$ then the indented lines are 1, 2, ..., $|n|$.

```
{\hangindent=1cm \hangafter=-2
Therefore, whatever I am going to say here has been derived
not from philosophical dialectics, abstract speculation, or
detached impersonal reflections, but from actual situations
and experiences with which I have been confronted. Indeed,
the term ‘‘lecture’’ also is, in this context, a misnomer.
\par}
```

Therefore, whatever I am going to say here has been derived not from philosophical dialectics, abstract speculation, or detached impersonal reflections, but from actual situations and experiences with which I have been confronted. Indeed, the term “lecture” also is, in this context, a misnomer.

Questions?

Macros and Primitives

While many modern languages provide some form of functions (or procedures, or subroutines), TEX does not. Instead TEX is a macro-based language.

This makes sense since TEX's purpose is to typeset, not program. A macro is simply something which takes input and swaps it with some output.

Some other languages, like C, also have macros. For example

```
1  #define MACRO(a) (Input is a)
2  MACRO(hello)
```

Will create a file containing the line `Input is hello` if the C preprocessor is run on it.

TEX macros are similar: they take input and swap it with some output within the document. For example

```
1  \def\macro#1{Input is #1}
2  \macro{hello}
```

Will create output with the line `Input is hello` once it is compiled.

But this is a simple example; TEX macros are far more powerful than C's. In fact, TEX is Turing Complete, meaning any algorithm can be written using TEX (Church-Turing *hypothesis* be damned).