1 LaTeX: Structural Facts.

1.1 Basics.

LaTeXhas the same special symbols and control sequences for printing them as $\mathcal{AMS}$-TeX with exception of `@`, this normally prints.

Control sequences in LaTeX have a special property. Some of them have optional arguments. These are put between the control sequence and its argument, closed in the square brackets []. If there are more optional arguments, they are all within one pair of brackets, separated with commas.

Some control sequences have "moving arguments", i.e. arguments that are used several times or printed at some other place than where they were used. These control sequences include `\footnote`, sectioning commands, `\addcontentsline` (see 1.7), `\markright`, `\markboth` (see 2.7), @-expressions (argument used repeatedly) and the letter environment.

Some commands are "fragile" in that they may behave unexpectedly when used within a moving argument. These include `\footnote`, `\begin`, `\end`, `\\`, `\\*`, all commands with a *-form, `\item`, `\[`, `\]`, and all commands with an optional argument. To make sure they work right in a moving argument we may precede them by `\protect`. For instance, `\footnote{First line\protect\\Second Line}`.

1.2 Basic Structure.

Every document in LaTeX starts with `\documentstyle`. This control word has an argument that describes what kind of document we want to create. Possibilities are `article`, `report`, `book`, and `letter`. This control symbol also has six optional arguments (called "document style options") describing closer the look of the document. Default size is 10 pt, one can also use `11pt` or `12pt`. Normally the text is typeset in one column, but we can use `twocolumn`. Also, it is assumed that the text will be printed on one side of the paper and things are arranged accordingly. But we can specify `twoside`. Here is an exception: if we choose the `book` style, two-sided printing is default

Displayed equations are normally centered with equation numbers printed right justified. `fleqn` will cause equations to be displayed on the left, and `leqno` will align equation numbers with the left margin. Finally, the command `\maketitle` (see 1.3) makes the title page as a part of the first page in the `article` style, otherwise as a separate page. The last optional argument `titlepage` tells `\maketitle` to do it on the separate page in every style.

Example: `\documentstyle[12pt,twoside]{article}`

After `\documentstyle` come definitions and further formatting commands. The text itself is between `\begin{document}` and `\end{document}`, the format depends on the style chosen. No `\bye` needed.

1.2.1 The `letter` style.

The document part of a letter is highly standardized. Since this is a completely different kind of a document, we cover it in this section to get rid of it. The parts between `\begin{document}` and `\end{document}` are:

`\begin{letter}{`*address*`}`, lines are separated by `\\`.

`\date{`*date*`}`, date is printed top right. This is optional. If not used, LaTeX prints the date it found on a system clock.

`\address{`*return address*`}`, lines separated by `\\`. This is optional. If it is used, the letter is formatted as personal. If not, it is formatted as a business letter to be printed on a paper with a heading.

`\opening{`*greetings*`}`. This not only prints that "Dear Mr. X" we put in as an argument, but it also uses all the information supplied up to that time to compose the header. That's why `\date` and `\address` must come before this.

The *text* of the letter itself.

`\signature{`*name*`}`, lines separated by `\\`.

`\closing{`*goodbyes*`}`. This prints "With appreciation" etc. supplied in the argument, but it also arranges the whole letter. That's why `\signature` must come before, although it is printed after the closing phrase (with a vertical space between them). Only special things can come after `\closing`.

`\cc{`*names*`}` separated by `\\`.

`\encl{`*what is enclosed*`}`

`\ps{P.S.` *P.S. part*`}`

`\end{letter}`

This is all about the letter style. From now on we will focus on the three main styles.

1.3 The `article`, `report`, and `book` styles.

The first thing we do after `\begin{document}` is to make the title page. First we supply the relevant information using

`\title{`*title*`}`,

\author{*authors*} separated by \\ or \and.
\date{*date*}. This is optional. If we don't specify a date, LaTeX substitutes the one found on the internal computer clock. This date is also accessible using the control word \today. So if we write \date{\today}, it is as if we did not do anything. We can use \date{} to get rid of the date.
\maketitle composes the title and prints it according to the style and optional argument of \documentstyle. Within the text of the title we can use \thanks to make a footnote. The argument is printed at the bottom of the page and a little asterisk * is used as a marker. This is the only difference from \footnote (see 2.7).
After this comes the text itself.

1.4 Environments.

An environment takes a portion of a text and treats it in a certain way (margins, indentation, line spacing, etc). It is marked by \begin{*environment*} and \end{*environment*}. Environments can be nested. Since \begin opens a group and \end closes it, the usual group rules apply. We saw the basic document environment and the special letter environment.
The environment titlepage creates a page with no page numbering, no formatting is supplied. For poetry there is the verse environment. There are two environments for quotations. quote doesn't indent paragraphs and is suitable for short quotations or several one-liners. quotation has paragraph indentation and is better for longer texts. They indent and justify both sides.
All text within the verbatim environment is printed exactly as is, including special characters. For this, one can also use \verb+*text*+. It is possible to use some other character instead of + to define the argument, something that is not used within *text* of course. \verb*+*text*+ also prints spaces. Note that the *text* is printed using \tt.
Special environments are used to typeset displayed formulas, tables, etc. They are treated in the appropriate sections. Let us mention the environment theorem which prints the text but precedes it with **Theorem.**

1.5 Sectioning commands.

The LaTeX has six levels of sections numbering that it keeps track of. Corresponding commands for increasing the right level by one are (in the order from the highest level): \chapter (not used in article style), \section, \subsection, \subsubsection, \paragraph, and \subparagraph. Each of these commands assigns the appropriate number to the section it starts, selects a large and/or bold font, prints the argument of this control word (if this title is too long, we can break it with \\), puts some vertical space before (or starts a new page) and after, and creates the corresponding entry in the Table of Contents (see 1.7). If we want a different text to be put into the Table of Contents, we put it as an optional argument.
The commands \section etc. print the corresponding number and the text we supplied on a line in some size and font. On the other hand, \chapter starts a new page, writes **Chapter** and the corresponding number on one line, and our text on a second line.
The sectioning commands also supply information for the running head (depending on the style, see 2.7). If we want a different one, we can supply it as an argument of \chaptermark, \sectionmark etc.
There is also a sectioning command \appendix, which resets the counter for the highest level to use capital letters. Note that this does not assign it any value. Therefore, if we use, say, \subsection after \appendix, nothing will be written for the highest level and the lower levels will be counted as if nothing happened, i.e. continue from where they left. Thus we have to follow \appendix with the sectioning command corresponding to the highest level (\section for article style, \chapter for report and book) to reset the counters, the main one will start from A.

If we just want to put a heading of the corresponding level without assigning a number (and without an entry in the Table of Contents), we use the "asterisk"-form, that is, \chapter*, \section*,...

1.6 Referring to Sections.

When we create a section using one of the commands above, we can also assign it a label using \label{*label*} right after the sectioning command. The *label* can be any text without special characters. The number of the section and a page where this label occurred is stored in the **.aux**-file. It can be referred to using \ref{*label*} (this refers to the section number) or \pageref{*label*} (substitutes the correct page number) anywhere in the text (even before the \label was defined). This, of course, requires that we run LaTeX twice. The first time we run it, the correct info is stored in the **.aux**-file, the second time it is read properly for the references. Note that if we use label after one of the asterisk-forms, the number we get when using \ref is the one assigned to the previous section, as *-forms do not increase it. The page number will be correct.
Let us note that we can similarly refer to numbered tables (see 1.8.2) and figures (see 1.8.1).

1.7 Table of Contents

The information from the sectioning commands (the titles, respective the optional arguments) is saved in the **.toc**-file. Whenever we issue the \tableofcontents command, this information is used to create the Table of Contents. Since this is usually done somewhere at the beginning, we have to run LaTeX twice to do it properly.

Note that if we create a heading using some asterisk-form, the information is not included in the Table of Contents. We can put it there by immediately using the \addcontentsline command. (We should make sure these two commands fall on the same page to get proper page number.) This command adds information to the Table of Contents, List of Figures (see 1.8.1), or List of Tables (1.8.2). It has three parameters. The first specifies to which of these lists we add (`toc`, `lof`, or `lot`), the second which level is substituted (`chapter`, `section`,..., determines the way the info is printed), and the third parameter supplies the text. For example, after

\section*{*text*}

we may put

\addcontentsline{toc}{section}{*text*}.

Note that if the Table of Contents is more that one page long, we have to run LaTeX three times. First time the **.toc**-file is empty, so LaTeX reserves an empty page with ToC heading and starts the text on page 2. Second time we run LaTeX, it makes ToC based on the info from the **.toc**-file, but TOC is longer that one page, so the text starts later than on page 2. Hence all pages in TOC are shifted. But now the proper pages are in the auxiliary file, so the third time it comes out right.

1.8 Floating Objects.

A "float" is a piece of text that does not have a fixed position. It is "tied" to a point of reference in the text, and LaTeX finds the best possible place for it. It never splits across pages. There are two kinds, tables and figures. Formally there is no difference between them, they are just counted separately. A float is defined at a certain place in the text, and LaTeX puts it as soon as possible afterwards. Often it is at the place where we defined it, unless we specify a different preference in an optional argument. But sometimes a float is carried on, when there is no room for it. Floats that could not be fitted are stored in a memory that gets flushed out when \clearpage, end of a chapter or end of a document is encountered. We can also use \cleardoublepage, which starts a new page and then prints all tables and figures. In two-sided printed style it may even insert an empty page, because the first page of the flushed-out tables/figures is made odd-numbered, i.e. right-handed.

If we have too many floats and are too picky about positioning, we will end up having them at ends of chapters, so one should be careful.

1.8.1 Figures.

The `figure` environment creates a float that counts as a figure. The command \begin{figure} has an optional argument which specifies our preference for positioning. We may choose more options in the order from the most desired to the least desired. Options are:

- `t` top of the page,
- `b` bottom of the page,
- `h` here, where I put it in the text,
- `p` on a "float page" (with no text, just a bunch of floats).

Default is `tbp`.

Then we put what we want to float, until \end{figure}. Here and there we may throw in a \caption command. It has a text as an argument, which is printed centered together with a number that \caption assigns to it. The numbering is done separately for each chapter, from the section level down figures are numbered together. If we use more \caption's within one float, it will have several numbers. If we want a caption without numbers, we have to center the text by ourselves (see 2.3), there is no \caption*. As an example, if we want to have a space reserved for a picture 2 in high, and a caption, we would do it using

```
\begin{figure}[hb]
\vspace{2 in}
\caption{This is a picture.}
\end{figure}
```

Here we prefer to have it right away, or at worst at the bottom of the page. We can assign a label to every \caption by using \label right after it. It works as usual, \ref and all the lot.

1.8.2 Tables And Figures.

The environment table for tables works in precisely the same way as figure, the only difference is that it is numbered separately from figures. The rest of this section will be common for Figures (Tables):

The contents and placements of captions are stored in the **.lof** file (**.lot** file) and can be used to make the List of Figures (List of Tables). If one wants a different entry in the corresponding list, it can be supplied using an optional argument with \caption. When we want to print this table, we use \listoffigures (\listoftables). The order in which these appear for the first time should be \tableofcontents, \listoffigures, \listoftables.

An additional entry can be put into both of these tables using the command \addcontentsline. For Figures the format is \addcontentsline{lof}{figure}{*text*}, for Tables it is \addcontentsline{lot}{table}{*text*}. Again, we should make sure this falls at the right place. We can also add directly to corresponding files using \addtocontentsfile{*file*}{*text*}.

When the document is prepared in two-column format, every figure (table) will be set inside one of the columns. If we want them to extend across both columns, we use figure* (table*). The syntax is identical, but the optional arguments h and b are not available.

1.9 Bibliography.

LaTeX offers a thebibliography environment. Each entry of this bibliography starts with \bibitem, which expects an argument. Here we supply a key by which we want to refer to this particular reference, it cannot contain a comma. Further text up to the next \bibitem or \end{thebibliography} is considered the reference.

References are numbered. If we want to supply our own labels, we put them as optional arguments to \bibitem. The command \begin{thebibliography} expects a mandatory argument, which should be some text determining the width reserved for the label. If we use the numbering and expect 99 or less references, we write \begin{thebibliography}{99}. If we use our own labels, we put the longest there.

We quote using \cite with a key as an argument. There can be more keys separated by commas. If we want to add an additional information to the reference, we put it in as an optional argument of \cite. \nocite with a key list as an argument produces no output, but writes the keys to the **.aux** file.

1.10 Index.

There is an environment theindex that helps to create it. The text between \begin{theindex} and \end{theindex} consists of lines that start with \item, \subitem or \subsubitem. It puts the lines flush left in a two-column environment. The levels are distinguished by different indentation. It helps if sections (e.g. for different letters) are separated by small spaces. The command \indexspace creates just the right amount of a vertical skip.

This still requires that we put the page numbers into it ourselves. There is some help available. If we put \makeindex into the preamble, LaTeX creates a **.idx** file. We can use the command \index with a word as an argument. Whenever we do it, the word and the page this happened on is put into the **.idx** file. Note that if we don't have \makeindex in the preamble, all \index's are simply ignored.

So, if we put \index{norm} after we use "norm" in our text, and it falls on page 52, there will be \indexentry{norm}{51} in the **.idx** file. What next? We can take an editor, sort it alphabetically, remove duplicates (when two \index with the same name fall on one page), put the same entries together, and format it into a file we include in the theindex environment. Or we can use a program that does it automatically, including the trick that if one entry appears on pages 13, 14, and 15, it is marked as 13–15. The Unix version is called makeindex, in DOS they call it makeindx. The program takes an **.idx** file and translates it into a corresponding **.ind** file, which we can include into our master file using \input file.ind.

The makeindex program has two optional parameters: **-p** *num* starts numbering the index with page number *num*. We can also use **-p any**, then the index starts a page later than the source ended (this can be found e.g. in the **.log** file). Options **-p odd** respectively **-p even** put the index at the first odd/even page after the source ended. The option **-s** *file*.sty allows you to specify a style file, which consists of lines of <*specifier*, *attribute*> pairs.

Advanced indexing: Subentries are done using the exclamation mark. For instance, \index{basis}, \index{basis!unconditional}, \index{basis!Schauder} will do

basis, 13,15
    Schauder, 27
    unconditional, 34

depending on the pages, of course. We can use two levels of subentries.

One problem one may run into concerns alphabetizing of accented letters or entries that start with mathematical constructions. If the entry in `\index` is given as *string1*`@`*string2*, it is alphabetized according to *string1*, but *string2* is printed. For instance, `\index{space!classical!Lp@$Lp[0,1]$}`.

Some implementations of `makeindex` (but not all!) have a special operator `|`. For instance, if some entry spans several pages, we can mark the beginning with `\index{duals|(}` and the end with `\index{duals|)}`, the result should be something like duals, 13–19. Another possible use is to create a reference. For instance,

`\index{set!compact|see{compact!set}}` would print

set

compact, see compact, set.

In case it does not work, we can always add `\subitem{compact, see compact, set}` manually into the **.ind** file after running `makeindex`. Similarly we can add `\item{basis ({\it continued})}` in case the subentries of "basis" overflow to another page.

The special characters can be printed using quote, that is, `"@`, `"!`, `"|`. The quotes themselves can me printed using `\"`.

1.11 File Management.

One can take some pieces of the text and put them in separate files. The command `\input` with a filename in its argument includes its contents in the master file.

Sometimes we want to process only some of the files we use in the master file. But then the whole numbering would be wrong. This can be helped using two commands. In the appropriate places in the master file we use the control word `\include` with a filename as an argument. Note that `\include` starts a new page. Then, in preamble (i.e. before `\begin{document}`) we put `\includeonly` with a list of files separated by commas as its argument. When LaTeX runs the master file, it scans all files we put in using `\include` to get the right page numbers, numbering and references, but only the files listed in `\includeonly` are really processed and typeset.

If `\includeonly` is omitted, all files are processed. If some commands `\input` are present while `\inputonly` constructions are used, the contents of the related files will be normally processed, it is not affected by `\includeonly`.

`\include` may not appear in preamble or in a file already read by another `\include`.

All these commands assume that filenames have extensions **.tex**, so we have to specify extensions only if this is not the case.

Let us mention two more technical commands. `\typeout{`*text*`}` types the *text* into the **.log** file and on the terminal. The argument may use control sequences, they are substituted.

`\typein{`*text*`}` prints *text* on terminal and waits for a line of input. This input is included in the tex-file. However, we can precede the argument of `\typein` with an optional one which contains a control sequence. In this case this control sequence is defined to be equal to the input line.

1.12 Custom Commands.

New commands are defined using `\newcommand`. It has two parameters, first is the name of the new control sequence, the second is its meaning. If we want arguments, we put the number of them as an optional argument *between* the two mandatory ones. For instance, `\newcommand{\b}[1]{{\bf %1}}`. The new name cannot start `\end`.

We can also define a new environment. The command `\newenvironment` has three arguments. The first is the *name* of the new environment (just a name, not a control sequence), which we then use as any other environment, i.e. `\begin{`*name*`}` and `\end{`*name*`}`. The second argument defines the beginning of the environment, the third defines the ending of this environment. If we want `\begin{`*environment*`}` to have arguments, we put the number of required arguments as an optional argument after {*name*}. For example, if we make numbered lists with a heading with items in the slanted font, we could use

```
\newenvironment{mylist}[1]%
   {\underline{#1}\begin{enumerate}\sl}%
   {\end{enumerate}}
```

Both the above commands only allow to create a new thing. If we want to redefine an existing control sequence, we have to use `\renewcommand`. If we want to change an existing environment, we use `\renewenvironment`.

We can also define math environments analogous to the one for theorems. Format is `\newtheorem`, two parameters and then an optional one. First parameter gives the new environment's *name*, the second defines the *caption* that will be printed at the beginning of the environment. After the caption a number will be printed. We can use an optional

argument (comes last in square brackets) to provide a name of an already defined counter (usually connected with section numbering). The counter of the new environment is reset whenever the counter from the optional argument is increased.

Another possible format is to write `\newtheorem` with the environment's *name* parameter first, then an optional parameter (in square brackets) giving a name of another theorem-like environment and finally the second obligatory argument giving the *caption*. The new environment will have the format of the one that is referred to.

2 LaTeX: Text Mode.

Here we summarize main differences in the text mode. First we note that LaTeX has a command `\mbox` which works similarly to `\hbox`. This can be used in both modes and the text inside is always treated in the text mode, so it also doubles as `\text`. The argument is not broken into lines and creates a new thing, a unit which can be used further.

2.1 Characters.

Here follows the table of accents known by LaTeX. Notice subtle differences compared to $\mathcal{AMS}$-TeX, in particular it is obvious why `\.` does not work as a "not sentence ending period".

`\'a`	á	`\`a`	à	`\~a`	ã	`\"a`	ä	`\^a`	â	`\v a`	ă	`\u a`	ă
`\H a`	a̋	`\=a`	ā	`\b a`	a̲	`\.a`	ȧ	`\d a`	ạ	`\c a`	ą	`\t{aa}`	a͡a

The table for Swedish, Polish, and Germans from $\mathcal{AMS}$-TeX works in LaTeX, as well as `?`` for ¿, `!`` for ¡, dotless i and j, and `\accent23 u` for ů. We also have

`\S`	§	`\P`	¶	`\dag`	†	`\ddag`	‡	`\copyright`	©	`\pounds`	£

Note that `|`, `<` and `>` are only used by `\tt`. Conversely, `\H`, `\.`, `\l`, and `\L` are not allowed in `\tt`.

2.2 Fonts.

LaTeX has the same basic fonts as TeX, plus `\sf` for sans serif and `\sc` for Small Caps. For emphasis we use `\em`, which toggles between `\it` and `\rm`. It also has size commands that behave in the same way, i.e. as switches. These are `\tiny` for tiny , `\scriptsize` for scriptsize, `\footnotesize` for footnotesize, `\small` for small, `\normalsize` for normalsize, `\large` for large, `\Large` for Large, `\LARGE` for LARGE, `\huge` for huge, and `\Huge` for Huge.

One can combine size and font commands. The size command must go first, since it switches to `\rm` of the corresponding size, we then change it to the font we want.

New fonts are defined using `\newfont`{*control-seq*}{*font-name*}.

2.3 Formatting the Text.

LaTeX has the same dashes as $\mathcal{AMS}$-TeX, does quotes in the same way and has `\,` to use between quotes. It doesn't have `\.`, this `.\` does the ordinary period. LaTeX uses `\@.` for the after-sentence period. It also has `\dots` for the ellipsis ..., recall that "`\ `" makes a space.

The command `\clearpage` is the LaTeX equivalent of `\newpage`: it breaks a page without spreading the text down. The commands `\pagebreak` and `\nopagebreak` force, resp. prevent a page break. In LaTeX, these commands also accept an optional argument that "weakens" the effect of these two commands. This argument is a number ranging from 1 (a mild suggestion) to 4 (a rather strong suggestion). One can use these commands within a paragraph, then they take effect after the current line ends.

There is a switch `\samepage`. The text under its influence is not broken into pages. Sometimes it may be too much for LaTeX and he breaks anyway. In that case we may enforce our will using `\nopagebreak` and `\pagebreak` in appropriate places.

Commands `\linebreak` and `\nolinebreak` also accept an optional argument defining the force of suggestion. The former breaks a line exactly when used and the line is spread out to the right margin.

The control word `\newline` breaks a line without spreading it. LaTeX has a synonym `\\` for it. This works just like `\newline`, but has an optional argument which specifies the size of a vertical space that is to be left after a linebreak.

For instance, \\[.5in] would break a line and then leave half an inch. If we want to use the [character after \\, we can for instance type \\{}[to let LaTeX know that we don't want any optional argument.

The special form * inhibits a page break before the new line.

If we use \newline or \linebreak in the argument of a sectioning command (or \caption etc.), it will not be carried into the Table of Contents as this is a moving argument. In order to get this command work in ToC, we have to precede it by \protect.

As in $\mathcal{AMS}$-TeX, ~ is an unbreakable space. We also note that \mbox may be used to prevent LaTeX from hyphenating a word. Hyphenation works the same as in $\mathcal{AMS}$-TeX with \-, the \hyphenation command must be used between \documentstyle and \begin{document}.

Paragraphs are indented. We can prevent this using \noindent before the paragraph. Extra indentation may be added using \indent.

There is an environment center for centering text. Lines are separated by \\ (which can take an argument to make a space). The flushleft and flushright environments work similarly, putting things to the left, resp. right. The logical line we put in may be longer than a physical line on the paper. If this happens, the text is broken and each resulting line is centered separately without adjusting the spaces (respective put left, right). Instead of some lines we can put in a box of a certain size that resulted from other constructions (like tables, see 2.6.2).

The switch \raggedright tells LaTeX to stop adjusting spaces when breaking lines, so the right edge of the text will be ragged. It corresponds to the flushleft environment, but unlike that one it can be used inside other environments like quote or parbox. It doesn't start a new paragraph. The same is true about \raggedleft and \centering (this one is especially handy at the beginning of tables or figures).

\raggedbottom makes pages exactly the height of the text, whereas \flushbottom adjusts spaces to have all pages of the same height.

Two document styles can be handled using commands. After \onecolumn (resp. \twocolumn), LaTeX starts a new page and produces output in a single column (resp. two columns).

The command \vspace makes a vertical space of size specified in its argument. It does not start a paragraph. If used within a paragraph, it makes a space after the current line. It is ignored if it is the last thing on a page. The *-form \vspace* works just like \vspace but is not ignored. Note that we can have negative arguments.

Commands \bigskip, \medskip and \smallskip make vertical spaces, too, their sizes depend on document style. Unlike in $\mathcal{AMS}$-TeX, they don't start a new paragraph. All commands for vertical spaces may be used between lines in the center, ... constructions. It is definitely not recommended to put them into tables.

For horizontal spaces we have \hspace, which expects the size of the space as an argument. If this falls at the end of a line, it is ignored. This can be helped by using \hspace*, which does the same as \hspace but is not ignored.

The expandable space is called \fill in LaTeX. It is a space as long as possible and can be used in place of another length, for instance as an argument of \hspace, \vspace, or an optional argument for \\ (ignored in tables, or rather, in tables the available space is nil). We can try this trick:

```
\noindent On the left.\hspace{\fill}On the right.\\
```

does

On the left. On the right.

In particular, \hspace{\fill} works like \hfil from $\mathcal{AMS}$-TeX, and \vspace{\fill} works line \vfil. LaTeX also has \hfill, \vfill, \hrulefill, and \dotfill.

Not surprisingly, \underline underlines the text supplied as an argument. The underlined text will not be broken into lines.

2.4 Boxes.

A box is an element of certain size that LaTeX treats as a single letter, i.e. it can be a part of a line, moved, etc. A box can be created by certain environments (for instance tabular for making tables). Also, \mbox creates a box that is as wide as the text in it.

The command \makebox does the same as \mbox, takes an argument and puts it into a box, but it has two optional arguments. The first specifies the width of the box, default is the width of the text in the argument. The second argument specifies the position of the text in a box. The default is centered, we may put l or r as well.

Example:

```
\makebox[5 cm][l]{Line one}Text 1.\\
\makebox[5 cm]{Line two}Text 2.
```

does

Line one Text 1.

Line two Text 2.

Note that the argument of \mbox, resp. \makebox is text, not necessarily a plain in-line text. We may put other things into it, for instance other boxes created by these and following commands, as long as it is not longer than one line.

We can save a box created as above and use it several times. First we reserve a name using \newsavebox{*name*}, where *name* is a control symbol. Then we define it using \sbox or \savebox. The former saves an \mbox construction. The first argument is the name of the box (defined by \newsavebox), the second argument is the text we put into a box. The latter control word is for saving \makebox constructions. Its first argument (mandatory) is for the name of the box, then arguments follow as in \makebox (i.e. first two optional, then one obligatory). Once we reserve a name using \newsavebox, we can redefine it over and over.

After defining a box, we can use it by \usebox with the box name as an argument.

We can create \mbox, move it up/down, and specify a different vertical size for the box. The command is \raisebox. Its first argument is a specification of how much should the text be raised with respect to the line. The second argument is the text itself. Just like \mbox, the text is not broken. The box we create in this way goes from the baseline to the text, so if we move the text down (a negative shift), the box will go from the baseline down.

We can also use two optional arguments to specify how much should the box go counted from the baseline. These arguments go between the mandatory two! First says how high the box should go, the second how much below it should stretch. If we are not careful, we may define a box that has some text outside it! An example will come later.

The real treat is that \fbox makes a box just like \mbox, but draws a visible box around. \framebox draws a box around a box it creates just like \makebox. Say, \framebox[1 cm][r]{Hi!} does Hi!.

If the text we supply is longer than the space specified, it is properly positioned and left to stick out. For instance, if a longer text is to be right-aligned, then: text starts, then the box starts, and the end of text will be next to the end of the box. If we don't like this, we can use \parbox. This control word expects two arguments. First specifies the width of the box to be created, the second argument is some text. This text is broken into lines that fit into the box (correspondingly taller) of the given width. One may even start a new paragraph in the text, but it is not indented.

The \parbox box is centered if used within a line. An optional argument allows it to be top-aligned t or bottom-aligned b. Note that such a box can be (a part of) an argument of \framebox, which is how we can make a framed paragraph.

The contents of \parbox can include quite complicated things, like lists etc. But even more powerful is an environment minipage. \begin{minipage} has two arguments. The first is optional and specifies the alignment of the box, the second is mandatory and defines the width. Everything up to \end{minipage} is then set into a box of the given width. This behaves like a small page within a page. We can even make footnotes within a minipage, they are numbered separately using lower-case letters and printed within the minipage on its bottom, separated by a line.

The last thing we mention here is lines. The command \rule makes a box filled with black paint, the dimensions of which are two mandatory arguments, first is the width. This command can be also used to create lines (thin boxes) or "struts", invisible lines used to get the right dimension of a box. By default, the lower left corner is placed on the baseline. We can change this by putting a length value as an optional argument. Now comes the promised, all (almost) including example.

```
\framebox{
This
\framebox{\raisebox{3pt}[1pt][2pt]{ is }}
a
\parbox[t]{1.5cm}{very crazy and overall overdone}
example
\raisebox{-8pt}{\fbox{of}}
black art \rule[2pt]{4mm}{3mm}.
}
```

looks like:

This is a very crazy and overall overdone example of black art .

2.5 Lists.

There are three different environments for producing lists. They have the same form. Items are supplied using the control word `\item`, the text up to the next `\item` or `\end` is used as the item.

The `itemize` environment simply puts a black "bullet" • in front of each item. The `enumerate` environment numbers items. In both cases we can override the implicit label by supplying our own as an optional argument of `\item`.

The `description` environment puts no label, we have to describe all items using the optional argument. This environment is actually superfluous, we can use `itemize` (or `enumerate`, but this will then add a level to numbering) and use the optional argument for all items.

We can nest up to four of these environments. The `itemize` environment uses bullets for the first level, dashes for the second, asterisks for the third and dots for the last one. The `enumerate` environment uses arabic numbers with a period for the first level, letters enclosed in parentheses for the second level, and roman numerals, resp. capital letters with a period for third, resp. fourth level. If we use the optional argument in this environment, the numbering won't increase by one for this level.

```
\begin{itemize}
\item First level, line 1.
 \begin{enumerate}
 \item Second level, line 1.
 \item[:]Second level, line 2.
  \begin{description}
  \item[No.\ 1] Third level, line 1.  This one is actually quite long, possibly to illustrate how it
works when a very long text is broken into lines.

  It also contains another paragraph.
  \item[No.\ 2] Third level, line 2.
   \begin{enumerate}
   \item Fourth level, line 1.
   \end{enumerate}
  \end{description}
 \item Second level, line 3.
 \end{enumerate}
\item [$\heartsuit$] First level, line 2.
\end{itemize}
```

will print as

- First level, line 1.
  1. Second level, line 1.
  : Second level, line 2.

     No. 1 Third level, line 1. This one is actually quite long, possibly to illustrate how it works when a very long text is broken into lines.
     It also contains another paragraph.

     No. 2 Third level, line 2.
     (a) Fourth level, line 1.
  2. Second level, line 3.

♡ First level, line 2.

There is also a special environment `list` for customized lists. The format is analogous, entries start with `\item`. But here `\begin{list}` has two parameters. The first defines the default label (which can be again overriden by an optional argument of `\item`). The second argument contains a list of formatting specifications. If we skip some formatting parameter, it will be assigned the default value. Note that it *has* to be two arguments, so if we want defaults, we have to put {}. For the formatting parameters we can use, see 4.2.5. When we put these to the second argument, we don't use `\setlength`, we just put `\`*parameter length* statements one after another.

2.6 Tables.

LaTeX has two environments that put things into rows and columns.

2.6.1 Tabs.

The environment that does tabs is called `tabbing`. Between `\begin{tabbing}` and `\end{tabbing}` we put lines of text ended by `\\`, the last one ends with `\end{tabbing}`, i.e. no `\\`. These lines are not broken, so we should make sure they do not continue out of the page.

Within each line we can use the mark `\=` to set a tab stop at the place where this sign occured. We can use more of these within one line, they are assigned numbers. Each line starts to count from 1 again, so when we use `\=` within some of the successive lines, it will redefine the previously set tab of the corresponding number. Thus it may happen that if we define two tabs in one line and then define only one of them in another line, the tab number one will denote a place (the new position) more to the right than tab number two (which is still left as before).

Another control symbol we can use is `\>`, which translates as "go to the next tab in numerical order". Thus it may mean to go to the left.

In the following example we not only use these commands, but also print them to indicate their position in the text:

```
\begin{tabbing}
We will set tabs, one $\backslash$=\= and another $\backslash$=\=.\\
Now $\backslash$>\>once and $\backslash$>\>again.\\
Now we will reset the first tab, to the right from number two $\backslash$=\=.\\
Now go $\backslash$>\>one, go $\backslash$>\>two.
\end{tabbing}
```

makes

We will set tabs, one \=and another \=.
Now \> once and \> again.
Next we will reset the first tab, to the right from number two: \=.
Now go \> two. one, go \>

Sometimes we want to set up tabs without printing the defining line. In that case we use `\kill` instead of `\\`, such a line won't be printed but tabs used within it will be set. Let us also note that we can use the optional argument of `\\` to make a vertical space between lines (see 2.3).

Since here we work with lines as such, this construction cannot be a part of a line, it always starts from the beginning of a line and what is after follows as a new line (similarly to `displaymath` environment).

When typing such a table, we can leave empty lines in the `tabbing` environment to make it more readable.

There are more commands for tabbing. `\<` puts something to the left of the current tab. `\+` will cause the outcomes of all following actions to be one tab to the right compared to what they would be normally, `\-` does the same with moving to the left.

Less frequent is `\'`, which causes everything typed since the last `\>`, `\<`, `\'`, `\\`, or `\kill` to be moved to the right of the previous column, flush against the current column's tab.

`\‘` causes the text that follows up to the next `\\`, `\kill` or `\end{tabbing}` to be moved to the right, flushed right against the next tab stop. No `\>` or `\'` is allowed in such a line.

2.6.2 Tables.

The `tabular` environment allows one to specify the number and arrangement of columns, fill several rows, and the columns are then fixed according to the entries and a table is typeset. `\begin{tabular}` works as a control sequence that expects an argument. The argument lists columns according to their justification, `r` for right-justified, `l` for left-justified, and `c` for centred columns. We can also add `|` to tell the environment that we want a vertical rule between the corresponding columns. For example, `\begin{tabular}{|r|cc|}` would start a table with three columns, the first one is right-justified and has vertical lines around it, the next two are centred, and then there is a vertical line.

The text up to `\end{tabular}` then consists of lines defining rows. Individual entries are separated by `&`, lines are ended by `\\` (apart from the last one). Again, we can specify a vertical space with an optional argument, recall the trick if we want to start the next line with `[`. LaTeX will arange entries into columns, for each column determines its maximal width, and then puts them one after another with a little space between columns. There is also a bit of extra space between rows.

If we want to add horizontal lines, we use `\hline` at the right place, i.e. after `\\` ending the row we want to put a line under. Here is the exception to the general rule that we don't use `\\` for the last row. If we want a horizontal line after the last row, we put `\\\hline`. We also put it after the `\begin` statement if we want to start with a horizontal line. If we put two `\hline`'s in a row, there will be a small space between the lines.

There are two commands that allow one to access a part of the table. If we want to make a line under only some columns, we use `\cline` instead of `\hline`. It expects an argument of the form *number-number*, where the numbers specify range of columns to be underlined by this line.

A vertical bar across a row is done using `\vline`.

If we want to put something across more columns, we use `\multicolumn`. This expects three arguments. The first says how many columns we want to use to create our entry. The second specifies how the entry will be positioned in the reserved space (`r`, `l`, `c`). In the last argument we put the entry itself. This unit is put into the description of the corresponding line instead of the corresponding regular entries, there are `&`'s around and all. Since this unit makes several columns, we don't put `&`'s for these. Vertical lines that should have been in such a multi-column entry are ignored, so we have to put them into the optional argument as well.

Now is the time to make an example:

```
\begin{center}
\begin{tabular}{|l|c|c|r|} \hline
\multicolumn{1}{|c|}{\bf Left} & \multicolumn{2}{|c|}{\bf Middle} & \multicolumn{1}{|c|}{\bf Right}\\
\hline
\bf One & \bf Two & \bf Three & \bf Four\\ \hline
This & this & this & this one\\ \cline{2-3}
goes & is & is & goes\\ \cline{1-1}\cline{4-4}
to the left & centred & centred & right.\\ \hline
\end{tabular}
\end{center}
```

prints as

Left	**Middle**		**Right**
One	**Two**	**Three**	**Four**
This	this	this	this one
goes	is	is	goes
to the left	centred	centred	right.

Of course, the braces around 1,2 above are not necessary. Note that we had to specify the boldface font in every entry where it is used, as entries form groups. Note how we centred it.

There is actually another type of column, `p` for a paragraph. This type expects a length as an argument and defines a column of the prescribed width, the text we fill in in the table is written as a paragraph (without indentation). This can be also used to make one-liners but with a prescribed column width. The text substituted into a `p`-column cannot contain `\\` unless it is inside an explicit `\parbox`, inside a `minipage`, `array`, or `tabular`. It could be also used in the scope of `\centering`,`\raggedright`, or `\raggedleft`.

There is also a special kind of a column called `@`-expression. Its form is `@`{*text*}. A column is created consisting of *text*, and there is no space left between this column and its neighbours. Since the column is already filled, you don't put any entry for it. However, if we want to put some space, we can put `\extracolsep`{*dim*} and the corresponding extra space is added to the left of all successive columns unless another `\extracolsep` changes it.

Finally, `*`-expression defines several copies of the same combination of column definitions, format is `*`{*number*}{*columns*}. Note that the description of columns may contain another `*`-expression.

Before we show these features in yet another table, we remark that unlike the previous environment, a table constructed using `tabular` is considered one thing and can be therefore a part of a line. If we don't want it, we should put an empty line before and after, perhaps even `\noindent`. If we do put it into a line, it is centred. If we want the table to be aligned by its top, we use `t` as an optional argument in `\begin{tabular}` (i.e. *after* `{tabular}`), `b` will align the table by its bottom. Now one example:

```
This is
\begin{tabular}[t]{|p{2in}|r@{.}l|}\hline
This is how much you pay for an apartment if the price is reasonable:  & 335 & 55\\ \hline
This is how much you don't pay & 1,276 & 37 \\ \hline
```

```
\end{tabular}
part of a line.
```

will do

This is

This is how much you pay for an apartment if the price is reasonable:	335.55
This is how much you don't pay	1,276.37

part of a line.

Finally, there is a `tabular*` environment. It expects two arguments plus one optional. The first (new) argument specifies the width of the table. Then comes the optional argument giving a position and eventually the second obligatory argument describing columns.

2.7 Top And Bottom.

Let us start with footnotes. The command `\footnote` takes an argument and makes it into a footnote. Footnotes are automatically numbered. If I want another number (but not just any character, only integers), I supply it as an optional argument. The control word `\footnote` must come right after the marked word to avoid separation at the end of a line.

However, `\footnote` can be only used in the outer mode, i.e. not inside `\parbox` and similar par-making boxes. In that case we would use `\footnotetext`. First comes an optional parameter giving a number, then the obligatory argument giving the *text* of the footnote. This just defines the footnote. Only after we use `\footnotemark` in the text will be the mark put at that place and the *text* supplied by `\footnotetext` is put down.

Concerning notes, the control word `\marginpar` takes an argument and prints it to the margin as a note. The first line of this note is printed on the level with the annotated line (if possible), as you see. Here I put `\marginpar{Hi}`. The note appears on the right in one-sided printing. In two-sided printing it is put on the left on even pages and on the right on odd pages. This parity can be changed using switches `\reversemarginpar` and `\normalmarginpar`. One can also use `\marginpar[`*text1*`]{`*text2*`}`. The *text2* is then printed if it happens to be put on the right, the *text1* is printed if it falls on the left margin (say, if I want to have an arrow pointing at the text).

Every page has two areas that are not used to print the text, a *header* and a *footer*. What is put there is determined by the page style. The default is that the header is empty and the page number is centred in the footer. This is called the `plain` style. We can change this using `\pagestyle`, where the desired style is put as an argument. There are two more predefined page styles. The `headings` style puts nothing in the footer, the page numbers and also some other information are placed in the header. This "other info" depends on the document style. In `report` style there are just page numbers, `book` prints chapter titles (and has different odd/even pages!) etc. The `empty` style puts nothing, but the page number is still calculated and used by `\label`'s etc. Let us note that `\pagestyle{empty}` is the only way to produce a document without page numbers. If only the current page is to be affected, we can use `\thispagestyle`.

There is another page style called `myheadings`. This works like the `headings` page style, with the difference that now we supply the "other information". If the document is processed for one-sided printing, we specify the extra info as an argument of `\markright`. For a two-sided printed document we use `\markboth`, with text for even (left-hand) pages as an optional argument and text for odd (right-hand) pages as a mandatory argument (*cf.* `\marginpar`). The change is made as of the page within which this command was used.

The numbering style can be changed as well. The control word `\pagenumbering` expects a description of the style as an argument. Choices are: `arabic` for arabic numbers, `roman` for roman numbers in lower-case, `Roman` for roman numbers in upper-case, `alph` for lower-case letters, and `Alph` for upper-case letters.

This command also resets the page counter.

2.8 Pictures.

The `picture` environment allows one to draw basic pictures. First one prepares the working area. We use an imaginary grid whose default step (i.e. the side of a square there) is 1 in. If we want a different size (say .5 in), we change it using `\setlength{\unitlength}{.5 in}` (*cf.* 4.2). This change can be only done *outside* the picture environment, typically just before it. Then we define the "working bench" area in terms of number of grids horizontally and vertically, putting it as "picture arguments", say, `\begin{picture}(8,4)`. In the picture environment, dimensions and positions always come in pairs, one for horizontal and the other for vertical direction, and we put them into parantheses. They also come right after the control word, so if there are any optional parameters, they go only after the position/dimension argument.

The command `\begin{picture}` reserves the specified place and sets up a system of two coordinates with respect to the grid. The origin (0,0) is in the lower-left corner. We can use a second argument which would change the coordinates of

the lower-left corner. For instance, \begin{picture}(10,20)(5,5) would define a picture area with horizontal dimension 10 and vertical dimension 20, the lower-left corner has coordinates (5,5). Note that this only determines space reserved for the picture on a page. We can exceed the reserved limits and objects will be placed at their appropriate coordinates, perhaps out of the frame or out of a page.

Now we can put objects in this picture. This is done using the command \put with two arguments. The first argument specifies the position of the object we want to place, the second argument defines the object. The position actually specifies the coordinates of an "anchor point", which is a natural reference point for different kinds of pictures, typically the lower-left corner or a centre (if it makes sense).

The simplest thing is to put some text. \put(2,1.5){Hello, there!} would print "Hello, there!" to the grid so that the lower-left corner (the anchor point for text) of "H" has coordinates (2,1.5).

Some box-making commands can be used in the picture environment, but they have a different form. \makebox makes a box of dimensions specified by the first argument and puts the text specified by the second argument so that it is centred. This can be changed by an optional argument following the dimensions. In this optional argument we can use any meaningful combination of b (the text will be aligned with the bottom of the box), t (aligned with the top), l (left-justified), or r (right justified). The anchor point for such a box is its lower-left corner.

For instance, \put(1,0){\makebox(2,1)[b]{Text}} would make a box of size 2×1, the text "Text" will be in its vertical axis of symmetry but aligned with the bottom, and the whole box will have its lower-left corner at (1,0). A nice trick: \put(3,.5){\makebox(0,0){Centre}} will print the word "Centre" so that its centre is at (3,.5).

The control word \framebox has the same syntax and meaning as \makebox, but it draws a frame around the box. For instance, \put(6,2){\framebox(.5,.5)[tr]{Hi!}} will draw a $.5 \times .5$ box with the lower-left corner at (6,2), and the word "Hi!" will be put to the upper-right corner. If we put an empty shape inside (like \hspace{.1in}, we get \framebox to draw lines and rectangles.

Also \sbox and \savebox work in the picture environment. \sbox serves for saving boxes wrapped around some text (like if we put it to the argument of \put). It has two arguments, the first is the name of the box (after we define it), the second is the contents. For instance, after \newsavebox{\mybox} we can define \sbox{\mybox}{Text to save}. Now we can use it in the argument of \put, say, \put(0,0){\usebox{\mybox}}.

For saving the \makebox constructions we have \savebox. It has the name of the box as the first argument, other arguments are just like in the \makebox case. We use it with \usebox. Note that if we make the definitions using \sbox or \savebox within a particular picture environment, it will not be available outside.

There is a command corresponding to \fbox. It is called \frame and draws a box around the text it obtains as an argument. The anchor point is the lower-left corner. We can also use \dashbox. It works just like \framebox, but the frame is made up of dashes. It has one more argument (the first one), which specifies what multiple of \unitlength is the length of a dash. It can only have integer values, so we cannot have dashes shorter than the grid size. Say, \put(1,1){\dashbox1(6,2)[tl]{Oops!}}.

\shortstack creates a box that consists of several lines put one below another. They are supplied as an argument, lines are separated by \\. Lines are centred, but the optional argument (l or r) can make them left/right justified. The lower-left corner is the anchor point.

Example: \put(7,2){\shortstack[l]{Up\\Down}}.

One can also draw more complicated objects. The command \line draws straight lines that start at the anchor point, the origin. It has two arguments, the first defines the slope by means of specifying another point (can have negative values), the second argument specifies the line length in terms of how many times the length from (0,0) to the first argument should be taken. Thus, \put(2,3){\line(1,-2){.9}} will draw a line that starts at (2,3) and goes towards (3,1), but stops short. The points that define the slope are only allowed to have integer coordinates not exceeding 6 in magnitude with 1 as the highest common divisor.

The command \arrow works in the same way, but draws an arrow. The coordinates of the slope-defining point now can only be between -4 and 4.

The command \circle draws a circle of diameter (with respect to the grid) specified in the argument. The command \circle* draws a filled circle. The largest possible diameter for both of these control words depends on the installation, but is probably around .5 in. LaTeX has a finite set of diameters it can draw and picks the closest. The centre is the anchor point. For instance \put(0,0){\circle{.1}} would make a small circle around the origin.

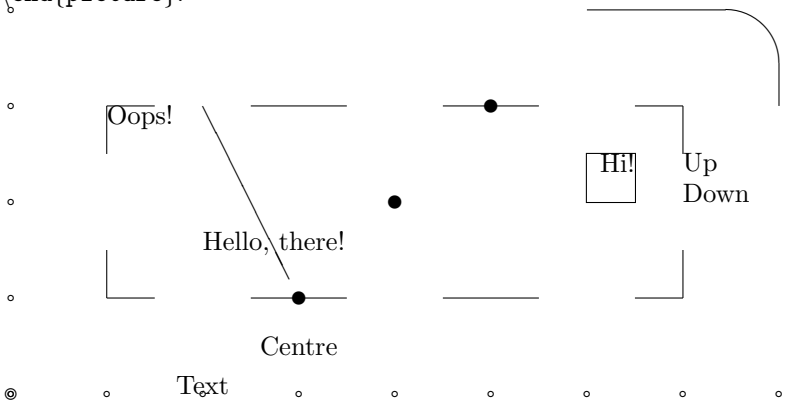
Using \oval we draw ovals, rectangles with rounded corners. It has two parameters specifying the width and length. It also has an optional argument that makes it to draw only a portion, possibilities are b, t, l, and r and combinations. For instance, \put(6,3){\oval[tr](4,2)}. The centre serves as the anchor point.

Finally, there is a command \multiput that alows one to draw one shape several times in regular intervals. It has four parameters. Position of the first figure, the step by which we should move in both directions, number of copies to make, and finally the description of the object.

Example: `\multiput(3,1)(1,1)3{\circle*{.13}}`.

One technical thing, the value of parameter `\linethickness` can be changed, but affects only horizontal & vertical lines, that is, not the thickness of slanted lines, ovals etc.

Now we do the definition of the grid and `\begin{picture}(8,4)`, and make small circles to show the integer coordinates using `\multiput(0,0)(1,0)9{\circle{.05}}`, similarly for the vertical axis. Then we use the examples above and then `\end{picture}`.



3 LaTeX: Math Mode.

LaTeX uses $ as usual, also _, ^, and ' work the same. In fact, the math mode is an environment, so `\begin{math}` and `\end{math}` are native delimiters for LaTeX. There are alternatives `\(` and `\)`, $ serves as both. To put some text into math we use `\mbox` that works just like `\text`. We can also use `\textrnm`, `\textbf` etc. LaTeX has all math accents from the table apart from `\dddot` and `\ddddot`, also has `\widehat` and `\widetilde`.

In math, `\dots` does smart dots as usual, and there are also `\ldots`, `\cdots`, `\ddots` (diagonal), and `\vdots` (vertical).

There are four spaces, a thick space `\;`, a medium space `\:`, a thin space `\,`, and a negative thin space `\!`.

There are four MM switches that affect the way math is written (limits of integrals/sums etc.). The `\displaystyle` switch will make the formula written as if it was displayed. `\textstyle` puts on the style in which formulas are written within a line; this can be also done using the construction `\mbox{$...$}`. `\scriptstyle`, `\scriptscriptstyle` use the default size for the first, resp. second order sub/superscripts. We note here that an in-line math piece delimited by $ defines a group, restricting the action of the above commands.

LaTeX knows `\overline`,`\underline`, `\overbrace` and `\underbrace`. The latter two behave in a bit different way. If we assign a sub/superscript to a corresponding braced expression, it is put under/over. Example: `\underbrace{X+\dots+X}_N` will do $\underbrace{X+\ldots+X}_{N}$.

The command `\stackrel` takes two arguments, the first is put in small type over the second one (i.e. the same as `\overset\to`). The default size of the first argument can be overriden using font commands as usual.

Fractions are done using `\frac`. Denominator and numerator are centred, the fraction line is aligned with the rest of the formula. When a fraction is in a displayed formula, the numerator and denominator are in the `\textstyle` font. If one of them contains a fraction, its numerator/denominator are in `\scriptstyle`. These defaults may be overriden using any of the font commands described above.

For roots there is `\sqrt`. The contents of the root is put in as an argument. The power is supplied as an optional argument. If the optional argument is not used, the square root is assumed. `\sqrt{a+b}` gives (in math mode of course) $\sqrt{a+b}$, `\sqrt[5]x` gives $\sqrt[5]{x}$.

3.1 Various Symbols.

Here we mainly point out differences with $\mathcal{AMS}$-TeX.

3.1.1 Funny Letters Et Al.

LaTeX has all Greek letters apart from the `\var` versions of capital letters. The command `\cal` makes caligraphic capital letters, but in LaTeX it works as a switch. Small letters and numbers change to various symbols when written under the influence of `\cal`, so it is recommended to use it in a group like {`\cal H`}. Next in the $\mathcal{AMS}$-TeX manual we have a table of math symbols, LaTeX is very similar, but instead of `\:` (does a medium space here) we have to use `\colon`; there are extra symbols `\Box` for □ and `\Diamond` for ◇.

The table of arrows is identical.

3.1.2 Math Symbols.

Apart from \and for &, LaTeX has the whole table of binary operators. It also has some extra guys:

\lhd	$\lhd$	\rhd	$\rhd$	\unlhd	$\unlhd$	\unrhd	$\unrhd$

LaTeX also closely follows the table of binary relations. It lacks \implies and \impliedby. It has extra relations \sqsubseteq $\sqsubseteq$ and \Join $\bowtie$.
LaTeX knows \int, \oint, and all sum-like large operators from the $\mathcal{AMS}$-TeX table. It also has an identical set of function names. We can define our own operator names using \mathop, in the argument we specify the font. For example, \newcommand{\dim}{\mathop{\rm dim\,}}.
It also has the same delimiters and identical commands \left and \right.

3.1.3 Fonts.

The text mode switches \bf, \rm, ... can be used in math mode, too, but they only affect the way letters are printed. Spaces are still ignored and symbols are printed as usual. Equivalently we can use \mathbf, \mathrm etc. There is a control word \boldmath that makes symbols as well as letters bold. In order not to make it too easy, this control word cannot be used in the math mode. Thus in order to get $\boldsymbol{x/y > z}$ we have to write {\boldmath$x/y>z$}. The group around was used to restrict its effect, since it is a switch. We can also cancel its effect using \unboldmath. If we want just a part of formula bold, we have to use \mbox first, and it gets really nicely complicated.

If we put \everymath{*font*} in the preamble, all math will be written in the font we specify. \everydisplay changes the font only for displayed mathematics.

3.2 Displayed Math Mode.

In LaTeX formulas are displayed using the environment displaymath. Since this is used so often, there is a shortcut \[for \begin{displaymath} and \] for \end{displaymath}. Inside this environment we can put many things (see further), but together this construction cannot exceed one line. Blank lines are not allowed within this environment. If we use a displaymath piece in the middle of a paragraph, the text that follows right afterwards is not indented. In fact, the $$ switch also works.
The environment equation behaves in the same way, but assigns a number to the whole thing. The position of this number depends on the style chosen (see 1.2). One can refer to this number (by number or a page) using \label and \ref, \pageref as described in 1.6. The command \label has to be used right before the closing \end{equation}.

3.3 Multi-Line Formulas.

The environment tabular (see 2.6.2) has an analog, the array enviroment. It behaves in exactly the same way including \hline et al, but can be only used in the math mode and the entries are processed in MM as well. It does not have the p columns. Again, it is considered one unit, so one can make it a part of a formula, use \left, \right with it etc. Note the big change from $\mathcal{AMS}$-TeX: because it follows the tabular format, & has to be on both sides of the sign used (= etc.).
There is a similar environment eqnarray. The differences are:

— it already includes switch into displayed math, i.e. it cannot be used within math mode,

— it has a preset size of three columns, first is right-justified, second (the sign) centred, third is left-justified,

— therefore, \begin{eqnarray} has does not expect any argument,

— assignes a number to each line,

— one cannot use horizontal/vertical lines.

We can cancel numbering of any line by using \nonumber right before \\ that ends the line (resp. before \end). This is also the place where we can put \label to refer to the line.
The eqnarray* environment works in the same way, only does not assign numbers to lines.
Example:

```
\begin{eqnarray}
f(x) & = & x^2 +2x+1\nonumber\\
& \ge & 0.
\end{eqnarray}
```

does

$$\begin{aligned} f(x) &= x^2 + 2x + 1 \\ &\ge 0. \end{aligned} \tag{1}$$

What if we want to supply our own labels instead of the successively numbered ones? We can use the plainTEX command `\eqno`, for instance

```
$$
1+1=2 \eqno (*)
$$
```

does

$$1 + 1 = 2 \tag{$*$}$$

unfortunately this cannot be used for individual lines in a multi-line construction (we can assign a label to a whole construction, though).

Sometimes the first line in a multi-line display is too long to fit just half a page. Then we can use `\lefteqn` as follows:

```
\begin{eqnarray*}
\lefteqn{(x+y)^2+(x-y)^2+(a+b)^2+(a-b)^2+(p+q)^2+(p-q)^2+}\\
& = & x^2+y^2+a^2+b^2+p^2+q^2.
\end{eqnarray*}
```

does

$$\begin{aligned} &(x+y)^2 + (x-y)^2 + (a+b)^2 + (a-b)^2 + (p+q)^2 + (p-q)^2 \\ &\quad = \quad x^2 + y^2 + a^2 + b^2 + p^2 + q^2. \end{aligned}$$

The control words `\rule` and `\makebox` work in math and we can use them to do nice things. For instance, if we want several numbered formulas, we must use the `eqnarray` environment, but it doesn't allow one to make more columns. The solution is to make more columns within one `eqnarray` column using `\makebox`'s with fixed lengths and different justifications.

Another problem appears when we want to put fitting braces around an equation in `array` or `eqnarray` environment. We cannot use `\left\{` on one side and `\right\}` on other side, since the entries are considered separate groups and this would violate the grouping rules. Obvious solution involves `\left\{` and `\right.` on the left and analogous thing on the right. But then, if, say, there is a fraction on the right and a letter on the left, we would get different sizes. We solve this by making up a strut, `\rule{0in}{.25in}`, and put it with the letter.

LATEX also understands the plain-TEX matrix commands. If we want to typeset a matrix, we use the control word `\matrix`. The matrix is supplied as an argument, we write rows ended by `\cr`, items in each row are separated by `&`.

Sometimes we need to put some notation next to the rows and columns of a matrix, but outside it. The plain-TEX command `\bordermatrix` does just that. We put in the matrix as usual, including the outside notes (so there will be no entry for the first row and the first column), and the matrix parentheses are done ignoring the first row and the first column. Incidentally, this works in $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX as well. As an example we use

```
\[
\bordermatrix{
   & E_1 & E_2\cr
E_1 &  I  & A\cr
E_2 & A^{-1} & I\cr}
\]
```

that gives

$$\begin{array}{c} \\ E_1 \\ E_2 \end{array} \begin{array}{c} \begin{array}{cc} E_1 & E_2 \end{array} \\ \begin{pmatrix} I & A \\ A^{-1} & I \end{pmatrix} \end{array}$$

4 LaTeX: TeXnical Stuff.

If we don't want all those auxiliar files (when we are just debugging the text, for instance), we can put `\nofiles` to the preamble.

4.1 Counters.

Counter is a variable that LaTeX uses to number things. Each counter has a unique name (without a backslash). There are the following predefined counters: `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, `subparagraph`, `page`, `equation`, `figure`, `table`, `footnote`, `mpfootnote`, `enumi`, `enumii`, `enumiii`, and `enumiv`. Their names are self-explanatory, `\mpfootnote` is for footnotes in `minipage` environments. Bibliography uses the `enumi` counter.

We can set a certain value for a counter using the command `\setcounter`. It has two arguments, first specifies the counter and the second gives the value we want to assign. All standard counters are set to 0 at the beginning. When we use environments like `equation` or commands like `\item` or `\chapter`, they first increase the corresponding counter by one. The sectioning commands also set the lower-rank counters to zero, `\chapter` also resets `figure` and `table`.

We change the value of a counter using `\addtocounter` with the counter name as the first argument and the change as the second argument. This may be handy for instance if a table is split over two pages (this must be done manually by writing two tables). Then we may want to have the caption on both pages. When we use `\caption` for the second time, it would increase the table number by one. So we precede it with `\addtocounter{table}{-1}`.

We can get the value of a counter using the appropriate command. It consists of a backslash followed by `the` and the counter name as one word. For instance, we write `We are now on page~\thepage.` to get We are now on page 17.

The counters print as arabic numbers with the exceptions of the last three levels of enumerated lists, `\thempfootnote` prints lowercase letters, the style of `\thepage` may be changed by the `\pagenumbering` command, and the counter `chapter` is in upper-case letters after we use `\appendix`. These are defaults, we can change them using `\arabic`, `\roman`, `\Roman`, `\alph`, `\Alph` with the counter name as an argument. For instance, `\alph{section}` would print the section number in lower-case letters.

This can be used to redefine the counter printing commands, e.g. `\renewcommand{\thesection}{\alph{section}}`. Since these printing commands are included in the sectioning commands, after this definition our sections will be "lower-case lettered". We can do more complicated things. For instance, figures are counted by the chapter number and figure number separated by commas. We can change it, `\renewcommand{\thefigure}{\Roman{chapter}--\alph{figure}}`. Note that this does not change the numbering of chapters.

If we want to change the page numbering style and change the way page numbers are printed, we have to use `\pagenumbering` first and then `\renewcommand` for `\thepage`.

A special counter style is `\fnsymbol`. It only works with `footnote` and causes that footnotes are using symbols (from 1 to 9): *,†,‡,§,¶,‖,**,††,‡‡. We do it by `\renewcommand{\thefootnote}{\fnsymbol{footnote}}`. Of course, this requires that we don't have more than nine footnotes per chapter.

We can also define our own counters using `\newcounter` with the name of the counter as an argument. They behave in precisely the same way. In particular, the related printing `\the`*counter* command is created and its style may be changed as above. When using `\newcounter`, one can also put a name of another counter as an optional argument *after* the obligatory argument. When the counter in optional argument is incremented, the counter defined by this `\newcounter` is automatically reset. Finally, instead of `\the`*counter* we can use `\value{`*counter*`}`.

4.2 Style Parameters.

The information about the format of a document is stored in parameters. We can change them to make the document look the way we want. There are two kind of parameters.

Length parameters have names conforming to the control sequence standard, i.e. they start with a backslash, and they define a length. For instance, when starting a paragraph, LaTeX indents the first line by the length specified in `\parindent`. We can use the length parameters ourselves, for instance, `\hspace{2\parindent}` makes a space of the size of two paragraph indentations.

We can also change the default size using `\setlength` with two arguments. The first argument specifies the length, the second the new value. Another way of changing a length parameter is to write the length parameter followed by the new value, e.g. `\parskip 0 pt`.

We can set up our own parameters of length. First we define it using `\newlength`, with the name of the length as an argument. The name must follow rules for control sequences. Then we set it with `\setlength`. Similarly works `\addtolength`, which enlarges the given length by the given size.

The command \settowidth expects two arguments as well. The first is the length we want to define, the second argument is a piece of text. The length parameter will be set to the width of this text.

Some length parameters are "rubber lengths". They have a given size, but LaTeX may change it slightly if necessary (like spacing between paragraphs when fitting them in a page).

Some parameters are counters and can be handled as any other counter (see 4.1).

The default value of these parameters depends on the style chosen. The values for the 12pt size are given in tables. When two sizes are given, they refer to one-sided/two-sided printing.

Some parameters can be only changed in the preamble, some may be changed in the text.

4.2.1 Page Layout.

The size of the printed page is specified by several parameters. We start with the easier horizontal dimensions. The size of the left margin, i.e. the distance between the edge of the paper and the text, minus one inch is in \oddsidemargin. So after \oddsidemargin 0 pt we get a 1 in left margin. This parameter controls all pages in one-sided style, odd pages in two-sided style. The parameter \evensidemargin specifies the size of the left margin minus one inch for the even pages in case of two-sided printing. In \textwidth we then have the width of the printed page.

For vertical dimensions we first set the distance from the top edge to the header using \topmargin, it is minus one inch again. Then \headheight specifies height of header and \headsep separation between header and the body of the text. Next there is the text size \textheight, then \footskip for the distance from the body of text to the bottom of the footer, and finally the height of the footer in \footheight.

All these parameters must be set *before* \begin{document}.

Parameter	book	article & report	letter
\oddsidemargin	.25 in	.29/.55 in	.74 in
\evensidemargin	1.25 in	.82/.55 in	.74 in
\textwidth	.5 in	5.42 in	5.07 in
\topmargin	.73 in	.38 in	.38 in
\headheight	.17 in	.17 in	.17 in
\headsep	.275 in	.35 in	.63 in
\textheight	7.39 in	7.39 in	7.01 in
\footskip	.42 in	.42 in	.35 in
\footheight	0 in	0 in	.17 in

4.2.2 Text Formatting.

\topskip defines the distance between the top of the body of text on a page and the first baseline. Change in preamble only. \baselineskip defines the distance between the two successive baselines. Every paragraph uses the last encountered setting. The rubber length \parskip defines the spacing between paragraph, the last change is in effect. \parindent defines the paragraph indentation. The last defined value is used.

\baselinestretch changes the line spacing by a constant factor. This length parameter takes a real number without a unit. The change is done using \renewcommand, with \baselinestretch as the first argument and a number as the second argument. For instance, number 2 there would create double-spacing. Recommended to define in preamble, otherwise the change takes effect as of the next typesize changing command.

If the twocolumn style is used, \columnsep defines the space between columns. There is a vertical rule printed between columns, its thickness is in \columnseprule.

Parameter	book & article & report	letter
\topskip	.14 in	.14 in
\baselineskip	.2 in	.21 in
\parskip	0 in	.1 in
\parindent	1.5 em	0 em
\baselinestretch	1	1
\columnsep	.14 in	.14 in
\columnseprule	0 in	0 in

4.2.3 Notes.

There is an invisible rule (a strut) before each footnote. Thus the distance between two footnotes is the height of this rule plus the normal space between lines. The rule height is set in \footnotsep. The last value works.

Marginal notes are governed by parameters that can be set in preamble only. \marginparwidth defines the width of a marginal note, \marginparsep defines the distance between the note and the body of text, \marginparpush is the minimum vertical space between two marginal notes.

Parameter	book	report & article	letter
\footnotesep	.12 in	.12 in	.17 in
\marginparwidth	1 in	1.18/.94 in	1.25 in
\marginparsep	.1 in	.14 in	.15 in
\marginparpush	.1 in	.1 in	.07 in

4.2.4 Section Levels.

Two counters govern the section numbering. Section levels have numbers. The highest level, part, has number -1. Then chapter is 0, section is 1 etc. We can decide in preamble which levels we want to have numbered by setting the counter secnumdepth to the number of the lowest sectioning level (highest number) to be still numbered. In preamble we can also use tocdepth to set the lowest level that is still included in the Table of Contents.

Parameter	book	article	report	letter
secnumdepth	3	2	2	N/A
tocdepth	3	2	2	N/A

4.2.5 Lists.

The parameters that follow can be used in the second argument of the list environment.

\topsep is the vertical space separating list from the surrounding text in addition to paragraph spacing. \partopsep is still some extra spacing if the list is preceded by a blank line. To add space between items (in addition to paragraph separation) we use \itemsep, while \parsep defines the vertical spacing between paragraphs within an item. These are all rubber lengths. Note that \itemsep can be also used withing the standard list environments.

The item text is set \leftmargin from the left margin of the surrounding text and ends \rightmargin from the right margin of the surrounding text. These cannot be negative. If an item consists of more paragraphs, all but the first will be indented by \listparindent.

The box for labels is indented by \itemindent and \labelsep is the space that separates the box from the text of the item. The width of the box is in \labelwidth. Labels are placed to the left in this box.

Parameter	book & article & report	letter
\topsep	.125 in	.4 em
\partopsep	.04 in	0 in
\itemsep	.06 in	.04 em
\parsep	.06 in	0 in
\leftmargin	2.5 em	2.5 em
\rightmargin	0 in	0 in
\listparindent	0 in	0 in
\itemindent	0 in	0 in
\labelsep	.5 em	.5 em
\labelwidth	2 em	2 em

When using the list environment, we can also change the implicit item counter. For instance, if we want numbering by arabic numbers enclosed in hyphens counted by mycount, we would do

```
\newcounter{mycount}
\begin{list}{--\arabic{mycount}--}{\usecounter{mycount}\leftmargin 10 pt\itemindent -5 pt}
\item etc.
\end{list}
```

The command \usecounter resets the counter to 0.

4.2.6 Floats.

The number of floats on a page can be specified with the counter `totalnumber`, whereas counters `topnumber` and `bottomnumber` specify the maximum number of floats allowed at the top, resp. bottom of a page.

There are the following parameters that have real values without units and must be set using `\renewcommand` with the value as the second argument. If there are floats and text on a page, `\textfraction` defines what portion of the page must be filled with text at least. This ratio for the top of the page is defined by `\topfraction`, `\bottomfraction` is for the bottom. In the case of a float page, `\floatpagefraction` specifies the portion of the page that must be occupied by floats, the rest may be floats or white space.

Floats on a text page are separated by `\floatsep`. The distance between floats and surrounding text is given by `\textfloatsep` for floats at the top or bottom, and by `\intextsep` for floats in the middle of the page. These are rubber lengths.

In the `twocolumn` style we can have floats that go accross two columns. These are done using analogous parameters `dbltopnumber`, `\dbltopfraction`, `\dblfloatpagefraction`, `\dblfloatsep`, and `\dbltextfloatsep`.

Parameter	`book` & `article` & `report`	`letter`
`totalnumber`	3	3
`topnumber`	2	2
`bottomnumber`	1	1
`\textfraction`	.2	.2
`\topfraction`	.7	.7
`\bottomfraction`	.3	.3
`\floatpagefraction`	.5	.5
`\floatsep`	.19 in	.17 in
`\textfloatsep`	.28 in	.28 in
`\intextsep`	.19 in	.17 in
`dbltopnumber`	2	2
`\dbltopfraction`	.7	.7
`\dblfloatpagefraction`	.5 in	.5 in
`\dblfloatsep`	.19 in	.17 in
`\dbltextfloatsep`	.28 in	.28 in

4.2.7 Tables & Boxes.

The blank space attached to `tabular` columns is `\tabcolsep`. Thus the space between two adjacent columns is twice this length. `\arraycolsep` is the corresponding space for `array` environment (see math mode).

In both environments, the width of vertical lines produced by `|` in the argument, as well as the height of horizontal lines produced by `\hline` and `\cline`, is in `\arrayrulewidth`. The space between two successive `|` or horizontal line commands is `\doublerulesep`. The real-valued `\arraystretch` changes the spacing between rows and columns, must be changed with `\renewcommand`. It must be changed *before* we start with `\begin{tabular}` etc. to take an effect.

Note that if we change `\baselinestretch`, it affects the tables and we have to fix it. Say, if we make doublespacing by `\renewcommand{baselinestretch}{2}`, we correct the effect for tables by setting `\renewcommand{arraystretch}{.5}`.

When creating boxes using `\fbox` or `\framebox`, `\fboxrule` defines the thickness of lines. The amount of space between the text box and lines is in `\fboxsep`. None of these affect `\framebox` when used within the `picture` environment.

Parameter	**All Styles**
`\tabcolsep`	6 pt
`\arraycolsep`	.07 in
`\arrayrulewidth`	.006 in
`\doublerulesep`	.03 in
`\arraystretch`	1
`\fboxrule`	.006 in
`\fboxsep`	.04 in

4.2.8 Math.

Apart from the parameters mentioned in 4.2.7, we have `\jot` for the vertical space between rows of `eqnarray` environment.

A displayed equation is considered long if its left end is before (more to the left) than the end of the last line of text. The vertical space before such an equation is `\abovedisplayskip`, the space below is `\belowdisplayskip`. When the equation is short (i.e. it doesn't overlap with the last text line), we have similarly `\abovedisplayshortskip` and `\belowdisplayshortskip`. These are rubber lengths

None of these "skips" is used when we choose the `fleqn` style. Then `\topsep` determines all these vertical spaces. In this case we also have `\mathindent` that determines the indentation of displayed formulas.

Parameter	**All Styles**
`\jot`	.04 in
`\abovedisplayskip`	.17 in
`\belowdisplayskip`	.17 in
`\abovedisplayshortskip`	0 in
`\belowdisplayshortskip`	0 in
`\mathindent`	0 in