Was macht eigentlich...?

Tristan Storch

Fakultät für Mathematik
Universität Bielefeld

09. Januar 2014

Inhaltsverzeichnis

TEX?

- Textsatzsystem
- Makrosprache zur Erstellung eigener Befehle
- wird selten alleine eingesetzt (meist mit LATEX)

Beispiel:

```
1 Hello World!
2 \bye
```

TEX-Processor

input processor Erstellt interne Repräsentation durch Tokens (*character tokens* und *command tokens*) den Quelltextes

expansion processor Bestimmte command tokens (z.B. `edef`) werden expandiert.

execution processor command tokens werden hier ausgewertet und die horizontalen, vertikalen und mathematischen Listen erstellt.

visual processor Die Listen werden in Paragraphen, Formeln und Seiten umgewandelt und die Ausgabe-`dvi` (DeVice independent file format) wird erstellt.

TEX-Primitives

Primitives sind 325 Befehle, die direkt in TEX integriert sind. Diese können grob in folgende Kategorien unterteilt werden:

- Makro
- Logik
- Registers
- Schrift
- Zeichen
- Mathematik
- Worttrennung
- Tabellen
- Boxen
- Glue
- Kern
- Page
- Marks
- Inserts
- Penalties
- Paragraph
- Datei-I/O
- Debugging
- Job

Plain TEX

- Makropaket (wie LATEX) von Donald Knuth, der TEX enwickelt hat.
- ca. 600 Makros in 1241 Zeilen (inklusive Kommentaren)
- andere Authoren sollten ihre eigenen Makropakete entwickeln
- viele aktuelle Makropakete, wie LATEX und AMS-TEX, basieren darauf

Wir werden keine Unterscheidung treffen zwischen TEX und Plain TEX, da in LATEX alle Plain TEX-Makros verfügbar sind.

Catcodes

In TEX haben manche Zeichen eine spezielle Bedeutung, wie z.B. \ .
TEX hat 16 *category codes* (kurz *catcodes*), um die Bedeutungen anzugeben. Diese können mit dem Befehl `\catcode` geändert werden, wovon man allerdings nicht allzu exzessiv Gebrauch machen sollte, um den Code nicht zu unübersichtlich zu machen.

Tabelle

Code	Beschreibung	Standard in TEX / Plain TEX
0	Maskierungszeichen und Kontrollsequenzen	\
1	Gruppenanfang	{
2	Gruppenende	}
3	Mathematikschalter	$
4	Ausrichtungszeichen	&
5	End of Line	(ASCII EOL)
6	Makroparameter	#
7	Hochstellen	^ und (ASCII 11)
8	Tiefstellen	_ und (ASCII 1)
9	ignorierte Zeichen	(ASCII null)
10	Leerzeichen	(ASCII space) und (ASCII tab)
11	Buchstabe	A–Z und a–z
12	andere Zeichen	alles andere
13	aktive Zeichen	~
14	Kommentarzeichen	%
15	ungültige Zeichen	(ASCII del)

makeatletter/makeatother

Makronamen können nur Zeichen der Kategorie 11 enthalten. In LaTeX enthalten interne Makros, also Makros, die nicht vom Nutzer direkt benutzt werden sollen, ein @. Um ein Makro mit einem @ im Namen definieren zu können muss zunächst der Catcode des Zeichens @ auf 11 gesetzt werden. Ist dies erledigt sollte dies wieder rückgängig gemacht werden, also der Catcode von @ auf 12 gesetzt werden. Dies erledigen die beiden Makros `\makeatletter` und `\makeatother`.

```
1 \def\makeatletter{\catcode`@ = 11}
2 \def\makeatother{\catcode`@ = 12}
```

def

- eigene Makros definieren
- wird bei Aufruf expandiert
- bis zu neun Argumente
- Argumente können mit Separatoren getrennt werden

```
\def<macroname> #1<sep_1>#2<sep_2>...#9<sep_9>{
    MAKRODEFI mit Args #1,...,#9}
```

[AB,C]D

```
1 \def\macro #1.#2{[#1,#2]}
2 \macro AB.CD
```

edef

- ähnlich \def
- wird bei Definition *rekursiv* expandiert

AAA
BBA

```
1 \def\basemacro{A}
2 \def\dermacro{\basemacro}
3 \def\derdermacro{\dermacro}
4
5 \edef\emacro{\derdermacro}
6 \basemacro \dermacro \emacro
7
8 \def\basemacro{B}
9 \basemacro \dermacro \emacro
```

let

- überträgt Defintion eines Makros
- Anwendungsfall: Makro auf Basis der alten Defintion neu definieren

```
1  \def\basemaca{A}
2  \def\basemacb{B}
3  \def\dermac{\basemaca}
4
5  \def\derdermac{\dermac}
6  \edef\emac{\dermac}
7  \let\lmac\dermac
8
9  \basemaca \basemacb \dermac \\\\
10 \derdermac \emac \lmac
11
12 \def\basemaca{C}
13 \def\dermac{\basemacb}
14 \derdermac \emac \lmac
```

ABA

AAA
BAC

global

- Makros gelten nur innerhalb von Gruppe `{...}`
- `global` umgeht dies
- `\gdef` äquivalent zu `\global\def`
- `\xdef` äquivalent zu `\global\edef`

```
1  \def\macrol{A}
2  \def\macrog{B}
3  \macrol \macrog
4
5  {
6    \def\macrol{a}
7    \global\def\macrog{b}
8    \macrol \macrog
9  }
10
11 \macrol \macrog
```

AB
ab
Ab

expandafter/noexpand

\expandafter expandiert zuerst die Argumente

Ausgabe: ARG

```
\def\macro[#1]{Ausgabe: #1}
\def\argmacro{[ARG]}
\expandafter\macro\argmacro
```

\noexpand verhindert unmittelbare Auswertung in \edef

AB
AY

```
\def\amacro{A}
\def\bmacro{B}
\edef\emacro{%
  \amacro \noexpand\bmacro}
\emacro

\def\amacro{X}
\def\bmacro{Y}
\emacro
```

Register

TEX können Werte jeweils eines Typs (s.U.) aufnehmen

Plain TEX ähnlich wie Variablen

Typ	Beschreibung
box	eine Box
count	eine ganze Zahl
dimen	eine Länge
muskip	ein Glue in mu
skip	ein Glue
toks	Folge von Tokens

Counter

- ganze Zahlen
- normale Rechenoperationen
- Beispiele: `\c@chapter`, `\c@section` (nicht direkt bearbeiten!)

```
0 0
5 3
8 9
0 9
```

```
1  \newcount\rcounta
2  \newcount\rcountb
3  \the\rcounta{} \the\rcountb
4
5  \rcounta 5 \rcountb 3
6  \the\rcounta{} \the\rcountb
7
8  \advance \rcounta by 3
9  \multiply \rcountb by 3
10 \the\rcounta{} \the\rcountb
11
12 \divide \rcounta by \rcountb
13 \the\rcounta{} \the\rcountb
```

Dimension

- ganze Zahlen in Einheit sp (scaled point)
- Rechnen wie mit Countern
- in absoulten (pt, mm, cm, in) oder relativen (ex, em) Längeneinheiten
- Beispiele: \paperwidth, \paperheight, \parindent.

```
1 \newdimen\rdimen \rdimen 65536 sp
2 \the\rdimen
3
4 \advance \rdimen by 3 em
5 \the\rdimen
6
7 \divide \rdimen by 5
8 \the\rdimen
```

1.0pt
33.84999pt
6.76999pt

Glue

- 3 Dimensionsangaben: Normalgröße, maximaler Strech, maximaler Shrink
- verbindet vertikale und horizontale Blöcke
- Beispiele: \baselineskip, \parskip

```
\newskip\smallskipamount
\smallskipamount=3pt plus 1pt minus 1pt

\def\smallskip{\vskip\smallskipamount}
```

if* - else

Wie in anderen Programmiersprachen kann man auch in TEX den Programmfluss mit `if-else`-Konstrukten kontrollieren. In TEX gibt es jedoch nicht nur ein `if`, sondern gleich mehrere.

`\if*`	Beschreibung
`\if`	Wahr, falls Zeichencodes übereinstimmen
`\ifcat`	Wahr, falls Catcodes übereinstimmen
`\ifdim`	Dimensions-Vergleich mittels $<$, $>$ oder $=$
`\ifeof`	Wahr, falls End-Of-File Datei nicht existiert
`\iffalse`	Immer falsch
`\ifhbox`	Wahr, falls Box-Register eine horizontale Box enthält
`\ifhmode`	Wahr, falls im horizontalen Modus
`\ifinner`	Wahr, falls im internen Modus
`\ifmmode`	Wahr, falls im Mathematikmodus
`\ifnum`	Counter-Vergleich mittels $<$, $>$ oder $=$.
`\ifodd`	Wahr, falls Counter ungerade
`\iftrue`	Immer Wahr
`\ifvbox`	Wahr, falls Box-Register eine vertikale Box enthält
`\ifvmode`	Wahr, falls im vertikalen Modus
`\ifvoid`	Wahr, falls Box-Register leer ist
`\ifx`	Wahr, falls Makros gleich expandieren, Zeichencodes übereinstimmen oder Catcodes übereinstimen

Beispiele

This is false

```
\newcount\counter \counter 5
\ifnum \counter >6
  This is true
\else
  This is false
\fi
```

undefined
Makro definiert.

```
\ifx\macro\undefined
  undefined
\else
  \macro
\fi

\def\macro{Makro definiert.}
\ifx\macro\undefined
  undefined
\else
  \macro
\fi
```

loop

Mit `\loop ... \if* ... \repeat` kann man in LaTeX Code mehrfach ausführen:

Just 5 more steps.
Just 4 more steps.
Just 3 more steps.
Just 2 more steps.
Just 1 more steps.
Last step!

```
\newcount\counter \counter 5

\loop
  \ifnum \counter = 0
    Last step!
  \else
    Just \the\counter{} more
    steps.\\
  \fi
\ifnum\counter > 0
  \advance\counter by -1
\repeat
```

relax

Der Befehl `\relax` macht nichts. Er ist manchmal nötig, um ungewolltes Verhalten zu verhindern.

ohne `\relax`:

Lorem ipsum

dolor sit amet

```
1 Lorem ipsum \\
2 [10 pt] dolor sit amet
```

mit `\relax`:

Lorem ipsum
[10 pt] dolor sit amet

```
1 Lorem ipsum \\ \relax
2 [10 pt] dolor sit amet
```

```
\newif\if@noskipsec \@noskipsectrue
\newbox\voidb@x
\def\leavevmode{\unhbox\voidb@x}
\newskip\@tempskipa
\newdimen\z@ \z@=0pt % can be used both for 0pt and 0
\gdef\@afterindentfalse{\let\if@afterindent\iftrue}
\def\@nobreakfalse{\global\let\if@nobreak\iffalse}
\def\@nobreaktrue {\global\let\if@nobreak\iftrue}
\everypar{}
\newcount\@secpenalty \@secpenalty = -300
\def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
\def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
\def\@ifnch{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnch
  \else
    \ifx\@let@token\reserved@d
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
\long\def\@ifnextchar#1#2#3{%
  \let\reserved@d=#1%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifnch}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
```

```
\def\@hangfrom#1{\setbox\@tempboxa\hbox{{#1}}%
      \hangindent \wd\@tempboxa\noindent\box\@tempboxa}
\mathchardef\@M=10000
\let\@@par=\par
\def\@ssect#1#2#3#4#5{%
  \@tempskipa #3\relax
  \ifdim \@tempskipa>\z@
    \begingroup
      #4{%
        \@hangfrom{\hskip #1}%
          \interlinepenalty \@M #5\@@par}%
\let\kernel@ifnextchar\@ifnextchar
\long\def\@xdblarg#1#2{#1[{#2}]{#2}}
\long\def\@dblarg#1{\kernel@ifnextchar[{#1}{\@xdblarg{#1}}}
\newcount\c@secnumdepth
\def\@empty{}
\def\@seccntformat#1{\csname the#1\endcsname\quad}
\def\@sect#1#2#3#4#5#6[#7]#8{%
  \ifnum #2>\c@secnumdepth
    \let\@svsec\@empty
  \else
    \refstepcounter{#1}%
    \protected@edef\@svsec{\@seccntformat{#1}\relax}%
  \fi
  \@tempskipa #5\relax
  \ifdim \@tempskipa>\z@
    \begingroup
      #6{%
        \@hangfrom{\hskip #3\relax\@svsec}%
          \interlinepenalty \@M #8\@@par}%
```

```
\def\@startsection#1#2#3#4#5#6{%
  \if@noskipsec \leavevmode \fi
  \par
  \@tempskipa #4\relax
  \@afterindenttrue
  \ifdim \@tempskipa <\z@
    \@tempskipa -\@tempskipa \@afterindentfalse
  \fi
  \if@nobreak
    \everypar{}%
  \else
    \addpenalty\@secpenalty\addvspace\@tempskipa
  \fi
  \@ifstar
    {\@ssect{#3}{#4}{#5}{#6}}%
    {\@dblarg{\@sect{#1}{#2}{#3}{#4}{#5}{#6}}}}
\newcommand\section{\@startsection {section}{1}{\z@}%
                                   {-3.5ex \@plus -1ex \@minus -.2ex}%
                                   {2.3ex \@plus.2ex}%
                                   {\normalfont\Large\bfseries}}
```

LaTeX

newcommand

LaTeX stell zur Makrodefinition den Befehl `\newcommand`. Dieser hat einige Vorteile gegebüber dem Primitive `\def`.

- einfachere Syntax
- prüft, ob ein Makro schon definiert ist
- optionaler Parameter

```
1  \newcommand{\macroa}{%
2    Makro A\\}
3  \newcommand{\macrob}[2]{%
4    Makro B: #1 #2\\}
5  \newcommand{\macroc}[2][opt]{%
6    Makro C: #1 #2\\}
7
8  \macroa
9  \macrob{A}{B}
10 \macroc[A]{B}
11 \macroc{B}
```

Makro A
Makro B: A B
Makro C: A B
Makro C: opt B

renewcommand, providecommand

\renewcommand überschreibt vorhandendes Makro
\providecommand legt Makro an, falls nicht vorhanden

Befehl A
Befehl A mod
Befehl A mod
Befehl B

```
1  \newcommand{\macroa}{%
2    Befehl A}
3  \macroa
4
5  \renewcommand{\macroa}{%
6    Befehl A mod}
7  \macroa
8
9  \providecommand{\macroa}{%
10   Befehl A MOD}
11 \macroa
12
13 \providecommand{\macrob}{%
14   Befehl B}
15 \macrob
```

Beispiel: Neudefinition eines Makros mit altem

Lorem
Lorem Ipsum
Lorem Ipsum Dolor

```
\newcommand{\macro}{Lorem}
\macro

\let\oldmacro\macro
\renewcommand{\macro}{%
  \oldmacro{} Ipsum}
\macro

\makeatletter
\g@addto@macro{\macro}{ Dolor}
\makeatother
\macro
```

newenvironment

- stellt Parameter für \begin und \end bereit
- optionale Argumente nur in begin-Teil verfügbar
- kann mit \renewenvironment überschrieben werden

Lorem ipsum dolor

sit amet, consetetur

sadipscing elitr

```
\newenvironment{envir}{%
  \par\begingroup\tiny}{%
  \endgroup\par}

Lorem ipsum dolor

\begin{envir}
  sit amet, consetetur
\end{envir}

sadipscing elitr
```

Counters

- bequeme LaTeX-Wrapper
- Counter können an andere „gebunden“ Werden

C 4
D 0

```
\newcounter{countera}
\newcounter{counterb}[countera]

\setcounter{countera}{3}
\addtocounter{counterb}{4}

\Alph{countera}
\arabic{counterb}

\stepcounter{countera}

\Alph{countera}
\arabic{counterb}
```

Pakete und Dokumentklassen

`\documentclass{classname}` lädt die Datei `classname.cls`

`\usepackage{packagename}` lädt die Datei `packagename.sty`

Beiden Befehlen können Optionen mitgegeben werden, welche dann in den entsprechenden Dateien verarbeitet werden.

```
\def\documentclass{%
  \let\documentclass\@twoclasseserror
  \if@compatibility\else\let\usepackage\RequirePackage
    \fi
  \@fileswithoptions\@clsextension}
```

```
\def\RequirePackage{%
  \@fileswithoptions\@pkgextension}
```

Pakete schreiben

`\NeedsTeXFormat{...}` welche LaTeX-Version wird benötigt?

`\ProvidesPackage` wie heißt das Paket? Hier muss der Dateiname ohne die Endung `.sty` stehen.

`\RequirePackage` äquivalent zu `\usepackage`

`\DeclareOptions` jede Option, die man mit diesem Paket anbietet muss hiermit erstellt und verarbeitet werden

`\ExecuteOptions` Standardoptionen ausführen

`\ProcessOptions\relax` übergeben Optionen ausführen

`\endinput` muss immer am Ende der Paketdatei stehen

mypackage.sty:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mypackage}

\RequirePackage{lmodern}

\DeclareOption{sans}{%
  \renewcommand{\familydefault}{\sfdefault}%
}
\DeclareOption{roman}{%
  \renewcommand{\familydefault}{\rmdefault}%
}

\ExecuteOptions{roman}
\ProcessOptions\relax

\newlength{\pardefault}
\setlength{\pardefault}{\parindent}
\newcommand{\neverindent}{%
  \setlength{\parindent}{0pt} }
\newcommand{\autoindent}{%
  \setlength{\parindent}{\pardefault} }

\endinput
```

document.tex

```
\documentclass{article}
\usepackage[roman]{mypackage}

\begin{document}
...
\end{document}
```

Dokumentenklasse schreiben

`\ProvidesClass` Klassenäquivalent zu `\ProvidesPackage`

`\PassOptionsToClass` Optionen können damit an eine zu ladende Klasse übergeben werden

`\PassOptionsToPackage` ähnliches für Packete

`\DeclareOption*` dies wird ausgeführt für alle nicht aufgeführten Optionen

`\ClassWarning` Gibt eine Warnung beim Kompilieren aus

`\LoadClass` äquivalent zu `\documentclass`

myclass.sty:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{myclass}[2014/01/07 My Class]

\DeclareOption{10pt}{
  \PassOptionsToClass{\CurrentOption}{article}
}
\DeclareOption{sans}{
  \PassOptionsToPackage{\CurrentOption}{mypackage}
}
\DeclareOption*{
  \ClassWarning{myclass}{Unknown option '\CurrentOption'}
}

\ExecuteOptions{10pt}
\ProcessOptions\relax

\LoadClass[a4paper]{article}
\RequirePackage{mypackage}

\endinput
```

document.tex

```
\documentclass[sans]{myclass}

\begin{document}
...
\end{document}
```

Ende!

Vielen Dank für ihre Aufmerksamkeit.