struktex.sty*

Jobst Hoffmann
Fachhochschule Aachen, Abt. Jülich
Ginsterweg 1
52428 Jülich
Bundesrepublik Deutschland

gedruckt am 20. Juni 2025

Zusammenfassung

Dieser Artikel beschreibt den Einsatz und die Implementation des LaTeX-Paketes struktex.sty zum Setzen von Struktogrammen nach Nassi-Shneiderman.

Inhaltsverzeichnis

*Diese Datei hat die Versionsnummer v3.0a-2-gbbdacc3, wurde zuletzt bearbeitet am 2025/06/20, und die Dokumentation datiert vom 2025/01/25.

1 Lizenzvereinbarung

2 Vorwort

StrukTEX hat eine lange Entwicklung hinter sich. Bei der Entwicklung wurden verschiedene Programme zur Versionsverwaltung – cvs, subversion und aktuell git – eingesetzt, die alle unterschiedliche Möglichkeiten bieten, Versionsnummern zu definieren. Um diese unterschiedlichen Versionsnummern zeitlich zuordnen zu können, wird folgendes Schema verwendet: Versionnummern der Form „v-⟨*d*⟩.⟨*d*⟩[⟨*a*⟩]“ mit ⟨*d*⟩ als Dezimalzahl und ⟨*a*⟩ als Buchstabe, etwa v-4.1a bezeichnen die erste Entwicklungslinie (rcs). Die folgenden Versionnummern haben die Form „v⟨*n*⟩⟨*n*⟩⟨*n*⟩“ mit ⟨*n*⟩ als Dezimalziffer, z. B. v122 (subversion). Die aktuelle Entwicklung erfolgt unter git und benutzt Versionnummern der Form „v⟨*d*⟩.⟨*d*⟩[⟨*a*⟩][-⟨*d*⟩]-g⟨*x*⟩“, beispielsweise v2.1-13-gd28a927; ⟨*x*⟩ ist dabei eine Hexadezimalzahl. Allgemein gilt für das Alter und somit die Reihenfolge der Versionen

$$\text{v-}\langle d\rangle.\langle d\rangle[\langle a\rangle] < \text{v}\langle n\rangle\langle n\rangle\langle n\rangle < \text{v}\langle d\rangle.\langle d\rangle[\langle a\rangle][\text{-}\langle d\rangle]\text{-g}\langle x\rangle$$

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit LaTeX zu zeichnen. Das Makropaket wird im folgenden immer StrukTEX genannt. Es ist in der Lage, die wichtigsten Elemente eines Struktogrammes wie z. B. Verarbeitungsblöcke, Schleifenkonstrukte, Sinnbilder für Alternativen usw. zu generieren. Die Struktogramme werden mit Hilfe der Picture-Umgebung von LaTeX erzeugt.[1]

Ab Version v-4.1a werden die mathematischen Symbole von $\mathcal{AMS}$-TEX geladen, die den mathematischen Zeichensatz erweitern und andere Darstellungen von Mengensymbolen (etwa $\mathbb{N}$, $\mathbb{Z}$ und $\mathbb{R}$ für die natürlichen, ganzen und reellen Zahlen) ermöglichen. Insbesondere das Zeichen für die leere Menge ($\varnothing$) ist in der Darstellung auffälliger als das standardmäßige Zeichen („$\emptyset$“) und somit besser für die Darstellung von Struktogrammen geeignet. Die Dokumentation der Zeichen und weiterer Kommandos findet sich in 4.2

Weiterhin ist aus dem oz.sty die Idee übernommen, Variablennamen in *italics* zu setzen, ohne dass die teilweise unschönen Zwischenräume erzeugt werden.

Die Entwicklung dieses Makropaketes ist noch nicht abgeschlossen. Es war geplant, die Struktogramme unter dem Einsatz des Makros aus emlines2.sty zu zeichnen, um die durch LaTeX gegebenen Einschränkungen – es gibt nur vordefinierte Steigungen – aufzuheben. Dies ist – für das `\ifthenelse` mit den Versionen v-4.1a und v-4.1b, für das `\switch` mit der Version v-4.2a – erledigt, nicht jedoch für Systeme, die die entsprechenden `\special{...}`-Befehle nicht unterstützen.

[1] Wer es scheut, Struktogramme mittels LaTeX direkt zu schreiben, kann beispielsweise unter http://structorizer.fisch.lu/ ein Programm (Struktorizer) finden, mit dem man seine Struktogramme mittels Maus entwickeln und abschließend als LaTeX-Code exportieren kann.

Erreicht werden kann dies jedoch durch Einsatz entsprechender Makros aus dem curves.sty. Seit der Version v-8.0a wird das Paket pict2e.sty unterstützt, das mittels der üblichen Treiber die von der `picture`-Umgebung bekannten Beschränkungen auf nur wenige Steigungen im wesentlichen aufhebt, so dass sich die Benutzung der entsprechenden Option (s.u.) dauerhaft empfiehlt.

Ebenso ist es geplant, Struktogramme um Kommentarblöcke zu erweitern, wie sie in dem Buch von Futschek ([Fut89]) eingesetzt werden. Dieses ist ebenfalls mit der Version v-8.0a realisiert worden.

Weitere Zukunftspläne sind:

1. Ein `\otherwise`-Zweig beim `\switch` (abgeschlossen durch die Version v-4.2a).
2. Die Neuimplementation der `declaration`-Umgebung mittels der `list`-Umgebung gemäß [GMS94, Abs. 3.3.4] (abgeschlossen mit der Version v-4.5a).
3. Die Anpassung an LaTeX 2_ε im Sinne eines Packages (abgeschlossen durch die Version v-4.0a).
4. Die Verbesserung der Dokumentation, um Teile des Algorithmus verständlicher zu machen.
5. Die Unabhängigkeit des struktex.sty von anderen .sty-Dateien wie etwa dem JHfMakro.sty (abgeschlossen mit der Version v-4.5a).
6. Die vollständige Implementation der Makros `\pVar`, `\pKey`, `\pFonts`, `\pTrue`, `\pFalse` und `\pBoolValue` (erledigt vor Version v-7.0).
7. Die vollständige Internalisierung von Kommandos, die nur in der Umgebung `struktogramm` Sinn machen. Internalisierung bedeutet, dass diese Kommandos nur innerhalb der Umgebung definiert sind. Dies hat den Zweck, das Paket mit anderen Paketen verträglicher zu gestalten, etwa mit dem ifthenelse.sty. Begonnen wurde die Internalisierung mit der Version v-4.4a.
8. Die Unabhängigkeit der Dokumentation von anderen .sty-Dateien wie etwa dem JHfMakro.sty (abgeschlossen mit der Version v-5.0).
9. Eine alternative Darstellung der Deklarationen, wie sie von Rico Bolz vorgeschlagen wurde
10. Wiedereinführung der `make`-Ziele `dist-src` `dist-tar` und `dist-zip`.

Der derzeitige Stand der Implementierung ist an entsprechender Stelle vermerkt.

3 Hinweise zur Pflege und Installation dieser Dokumentation

Das Paket, zu dem der struktex.sty gehört, besteht aus insgesamt fünf Dateien:

`LIESMICH.md`,
`README.md`,
`struktex.dtx`,
`struktex.de.pdf` und
`struktex.en.pdf`.

Um daraus einerseits die Dokumentation, andererseits die .sty-Datei zu erzeugen, muss folgendermaßen vorgegangen werden:

Zunächst wird mit z. B.

```
pdftex struktex.dtx
```

die Datei struktex.dtx formatiert.

Dieser Formatierungslauf erzeugt einige weitere Dateien. Dies sind zunächst alle drei .sty-Dateien struktex.sty, struktxf.sty und struktxp.sty, die beiden letzteren werden von struktex.sty verwendet. Weiterhin sind dies die beiden Dateien struktex-tst-00.nss und strukdoc.sty, die für die Erstellung der hier vorgestellten Dokumentation verwendet werden. Dann gibt es fünf Testdateien struktex-test-*i*.tex, $i = 1(1)3$, struktxf.tex und struktxf.tex zum Testen der entsprechenden . sty-Dateien und nicht zuletzt zwei Skriptdateien lbuild.lua und lbuild-config.lua (siehe Abschnitt 6).

Die Dokumentation wird gemäß l3build mit dem Befehl

```
l3build doc
```

erzeugt.

Das Ergebnis dieses Formatierlaufes ist die Dokumentation in Form einer .pdf-Datei, die in gewohnter Weise weiterbearbeitet werden kann. Weitere Informationen zum Arbeiten mit der integrierten Dokumentation findet man in [Mit01] und [MDB01].

Die Installation wird abgeschlossen, indem die Datei `struktex.sty` in ein Verzeichnis verschoben wird, das von TEX gefunden werden kann, das ist in einer TDS-konformen Installation typischerweise `.../tex/latex/struktex/`, die Dokumentation wird analog in das Verzeichnis `.../doc/latex/struktex/` verschoben.[2]

Sollen Änderungen durchgeführt werden, so sollten neben diesen die Werte von `\fileversion`, `\filedate` und `\docdate` bei Bedarf entsprechend geändert werden. Weiterhin sollte darauf geachtet werden, dass die Revisionsgeschichte durch Einträge von

`\changes{`⟨*Version*⟩`}{`⟨*Datum*⟩`}{`⟨*Kommentar*⟩`}`

weitergeschrieben wird. ⟨*Version*⟩ gibt die Versionsnummer an, unter der die jeweilige Änderung durchgeführt wurde, ⟨*Datum*⟩ gibt das Datum in der Form `yy/mm/dd` an und ⟨*Kommentar*⟩ erläutert die jeweilige Änderung. ⟨*Kommentar*⟩ darf nicht mehr als 64 Zeichen enthalten. Daher sollten Kommandos nicht mit dem „\“ (*backslash*) eingeleitet werden, sondern mit dem „‘“ (*accent*).

Die nächsten Anweisungen bilden den Treiber für die hier vorliegende Dokumentation.

[2] Wenn die automatische Installation (vgl. Abschnitt 6) vorgenommen wird, erfolgt diese entsprechend.

```
%%
%% This is file ‘struktex.drv’,
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% struktex.dtx  (with options: ‘driver’)
%%
%% Copyright (C) 1989-2025 by Jobst Hoffmann.  All rights reserved.
%%
%% IMPORTANT COPYRIGHT NOTICE:
%%
%% No other permissions to copy or distribute this file in any form
%% are granted and in particular NO PERMISSION to modify its contents.
%%
%% You are NOT ALLOWED to change this file.
%%
%% Please address error reports and any problems in case of UNCHANGED versions
%% to
%%         j.hoffmann_(at)_fh-aachen.de
                                % set the default formatting language:
\expandafter\ifx\csname primarylanguage\endcsname\relax
    \def\primarylanguage{ngerman}
    \def\secondarylanguage{english}
\fi

\documentclass[a4paper, \secondarylanguage,     % select the language
        \primarylanguage]{ltxdoc}

\PassOptionsToPackage{obeyspaces}{url} % must be done before any package is
                                % loaded

\usepackage{babel}              % for switching the documentation language
\usepackage{moreverb}           % for formatting the documentation of the driver
\usepackage{strukdoc}           % the style-file for formatting this
                                % documentation

\usepackage[pict2e, % <---------- to produce finer results
                                % visible under xdvi, alternatives are
                                % curves or emlines2 (visible only under
                                % ghostscript), leave out if not
                                % available
    verification,
    outer, % <------------------- to set the position of the \ifthenelse
                                % flags to the outer edges
    debug,
    ]
    {struktex}

\OnlyDescription    % add comment char to include the implementation details

\MakeShortVerb{\|}    % |\foo| acts like \verb+\foo+

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% to avoid underfull ... messages while formatting two/three columns
\hbadness=10000 \vbadness=10000

\typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
\def\languageNGerman{\the\csname l@ngerman\endcsname} % depends on language.dat,
                                % there may be changes from release to
                                % release

```

```
63    \begin{document}
64    \makeatletter
65    \@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
66    \makeatother
67    \DocInput{struktex.dtx}
68    \end{document}
69
70    \endinput
71    %%
72    %% End of file 'struktex.drv'.
```

4 Die Benutzungsschnittstelle

Der struktex.sty wird wie jede andere `.sty`-Datei als *package* in ein LaTeX-Dokument eingebunden:

`\usepackage[`⟨*Optionen*⟩`]{struktex}`

Die folgenden Optionen stehen zur Verfügung:

1. `english`, `ngerman` oder `german`:

   Die jeweilige Option legt die Sprache für definierte Werte wie `\sTrue` fest, Standardwert ist `english`.

2. `emlines`, `curves` oder `pict2e`:

   Durch Angabe einer der drei Optionen ist es möglich, beliebige Steigungen in Struktogrammen zu zeichnen. Erstere Option ist sinnvoll, wenn mit dem emTeX-Paket von Eberhard Mattes gearbeitet wird (DOS oder OS/2), ansonsten wird der Einsatz von `pict2e` empfohlen. Der Einsatz des Paketes curves.sty (Ian Maclaine-cross), das das Zeichnen von Geraden beliebiger Steigungen durch das Setzen vieler einzelner Punkte ermöglicht, ist durch das Erscheinen des Paketes pict2e.sty (Hubert Gäßlein und Rolf Niepraschk) im Prinzip obsolet, aus Kompatibilitätsgründen wird er weiter unterstützt. Durch die Angabe einer der genannten Option wird das jeweilige Paket (emlines2.sty, curves.sty bzw. pict2e.sty) automatisch geladen, Standardwert ist `pict2e`.

3. `verification`:

   Nur wenn diese Option gesetzt ist, steht `\assert` als Kommando zur Verfügung.

4. `nofiller`:

   Setzen dieser Option lässt jeden Freiraum leer. Ansonsten wird Freiraum in Alternativen als ∅ markiert.

5. `draft`, `final`:

   Diese Optionen dienen in üblicher Weise dazu, den Entwurf beziehungsweise die endgültige Fassung zu kennzeichnen (vgl. `\sProofOn`/`\sProofOff`). Im Entwurfsmodus werden die vier Eckpunkte eines Struktogramms in der vom Benutzer vorgegebenen Größe ausgegeben, diese Markierung fällt in der endgültigen Fassung weg. Der Standardwert ist `final`.

6. `debug`:

   Mit dem Setzen dieser Option werden Zeilen der Form

   `===> dbg` ⟨*Text*⟩

   in die .log-Datei geschrieben.

7. `outer`:

   Das Setzen dieser Option führt dazu, die ja/nein Flaggen statt auf der Mitte der Grundlinie jeweils links bzw. rechts außen erscheinen zu lassen.

Nach dem Laden der `.sty`-Datei stehen verschiedene Kommandos und Umgebungen zur Verfügung, die das Zeichnen der Struktogramme ermöglichen.

`\StrukTeX` Zunächst sei der Befehl erwähnt, der das Logo StrukTEX erzeugt:

```
\StrukTeX
```

Damit kann in Dokumentationen auf die hier vorliegende Stil-Option verwiesen werden.

4.1 Die Makros zur Erzeugung von Struktogrammen

`struktogramm` (*env.*) `\sProofOn` `\sProofOff` `\PositionNSS` Die Umgebung

```
\begin{struktogramm}(⟨Breite⟩,⟨Höhe⟩)[⟨Überschrift⟩]
...
\end{struktogramm}
```

erzeugt Raum für ein neues Struktogramm. Die beiden Parameter legen die Breite und die Höhe des Platzes fest, der für das Struktogramm reserviert wird. Die Angaben werden in Millimetern gemacht, wobei der aktuelle Wert von `\unitlength` keine Rolle spielt. Dabei entspricht die Breite der tatsächlichen Breite, die tatsächliche Höhe wird den Erfordernissen angepasst. Stimmt die angegebene Höhe nicht mit den tatsächlichen Erfordernissen überein, läuft das Struktogramm in den umgebenden Text hinein oder es bleibt Raum frei. Es gibt einen Schalter `\sProofOn`, mit dem die angegebene Größe des Struktogramms durch vier Punkte gezeigt wird, um Korrekturen einfacher durchführen zu können. `\sProofOff` schaltet diese Hilfe entsprechend wieder ab. Die Überschrift dient zur Identifizierung des Struktogramms, wenn man sich von anderer Stelle, etwa aus einem anderen Struktogramm heraus auf dieses hier beziehen will.

Die Struktogramm-Umgebung basiert auf der `picture`-Umgebung von LaTeX. Die in der `picture`-Umgebung gebräuchliche Längeneinheit `\unitlength` wird bei den Struktogrammen nicht benutzt; die Längeneinheit ist aus technischen Gründen auf 1 *mm* festgelegt. Weiterhin müssen alle Längenangaben ganzzahlige Werte sein. `\unitlength` hat zwar nach dem Zeichnen eines Struktogrammes mit StrukTEX den gleichen Wert wie vorher, ist aber innerhalb eines Struktogrammes umdefiniert und darf dort auch nicht geändert werden.

`\assign` Das Hauptelement eines Struktogramms ist ein Kasten, in dem eine Operation beschrieben wird. Ein solcher Kasten wird mit `\assign` erzeugt. Die Syntax ist

```
\assign[⟨Höhe⟩]{⟨Inhalt⟩},
```

wobei die eckigen Klammern wie üblich ein optionales Argument bezeichnen. Die Breite und die Höhe des Kastens werden den Erfordernissen gemäß automatisch angepasst, man kann aber mittels des optionalen Argumentes die Höhe des Kastens vorgeben.

Der *Text* wird normalerweise linksbündig in den Kasten gesetzt; ist er dafür zu lang, so wird ein Paragraph (im Blocksatz) gesetzt.

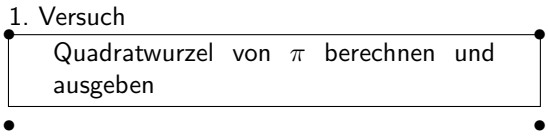
Beispiel 1

Ein einfaches Struktogramm wird mit den folgenden Anweisungen erzeugt:

```
\sProofOn
\begin{struktogramm}(70,12)[1.\ Versuch]
    \assign{Quadratwurzel von \(\pi\) berechnen und ausgeben}
\end{struktogramm}
\sProofOff
```

Diese Anweisungen führen zu folgendem Struktogramm, wobei der Anwender wie auch bei der zugrundeliegenden `picture`-Umgebung für eine geeignete Positionierung zu sorgen hat. Die Positionierung erfolgt in dieser Dokumentation im Regelfall mit der `quote`-Umgebung, man kann ein Struktogramm aber auch mit der `center`-Umgebung zentrieren. Die Breite des Struktogramms ist mit 70mm vorgegeben, die Höhe mit 12mm. Eine Alternative ist durch die `centernss`-Umgebung gegeben, die auf Seite 19 beschrieben wird.

Gleichzeitig wird die Wirkung von `\sProofOn` und `\sProffOff` gezeigt, wobei die zu große vorgegebene Größe des Struktogrammes zu beachten ist.



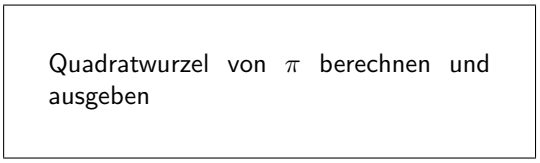
Die Bedeutung des optionalen Argumentes macht das folgende Beispiel deutlich.

Beispiel 2

Die Höhe des Kastens wird vorgegeben:

```
\begin{center}
\begin{struktogramm}(70,20)
    \assign[20]{Quadratwurzel von \(\pi\) berechnen und ausgeben}
\end{struktogramm}
\end{center}
```

Diese Anweisungen führen zu folgendem Struktogramm, wobei zu beachten ist, dass die `struktogramm`-Umgebung mittels einer `center`-Umgebung zentriert wurde, wobei die Breite des Struktogramms wiederum mit 70mm vorgegeben ist, die Höhe diesmal aber mit 20mm.



`declaration` (*env.*) Die `declaration`-Umgebung dient der Beschreibung von Variablen bzw. der Beschreibung der Schnittstelle. Ihre Syntax ist

```
\begin{declaration}[⟨Überschrift⟩]
...
\end{declaration}
```

`\declarationtitle` Die Überschriftsangabe ist optional. Lässt man die Angabe weg, so wird standardmäßig die Überschrift: „Speicher bereitstellen:“ erzeugt. Will man einen anderen Text haben, wird dieser mit `\declarationtitle{`⟨*Überschrift*⟩`}` global festgelegt. Will man für ein einzelnes Struktogramm einen speziellen Titel erzeugen, so gibt man diesen in den eckigen Klammern an.

`\description` `\descriptionindent` `\descriptionwidth` `\descriptionsep` Innerhalb der `declaration`-Umgebung werden die Beschreibungen der einzelnen Variablen mit

```
\description{⟨Variablenname⟩}{⟨Variablenbeschreibung⟩}
```

erzeugt. Dabei ist zu beachten, dass ⟨*Variablenname*⟩ keine schließende eckige Klammer „]“ beinhalten darf, da dieser Makro mittels des `\item`-Makros definiert worden ist. Eckige Klammern sind in diesem Fall als `\lbracket` und `\rbracket` einzugeben.

Das Aussehen einer Beschreibung lässt sich mit drei Parametern steuern: `\descriptionindent`, `\descriptionwidth` und `\descriptionsep`; die Bedeutung der Parameter ist der Abbildung 1 zu entnehmen (`\xsize@nss` und `\xin@nss` sind interne Größen, die von StruktTEX vorgegeben werden). Die Vorbesetzung dieser Werte ist folgendermaßen:

```
% \descriptionindent=1.5em
% \descriptionwidth=40pt
% \descriptionsep=\tabcolsep
%
```

Die Bedeutung von `\descriptionwidth` ist darin zu sehen, dass ein Variablenname, der kürzer als `\descriptionwidth` ist, eine Beschreibung erhält, die auf der gleichen Höhe liegt; ansonsten wird die Beschreibung eine Zeile tiefer begonnen.

Beispiel 3

Zunächst wird nur eine einzelne Variable beschrieben.

```
\begin{struktogramm}(95,20)
    \assign%
    {%
        \begin{declaration}
            \description{\pVar{iVar}}{eine \pKey{int}-Variable,
                                deren Beschreibung hier allein dem
                                Zweck dient, den Makro vorzuführen}
        \end{declaration}
    }
\end{struktogramm}
```

Das zugehörige Struktogramm, wobei zu beachten ist, dass durch die leeren eckigen Klammern keine Überschrift erzeugt wird.

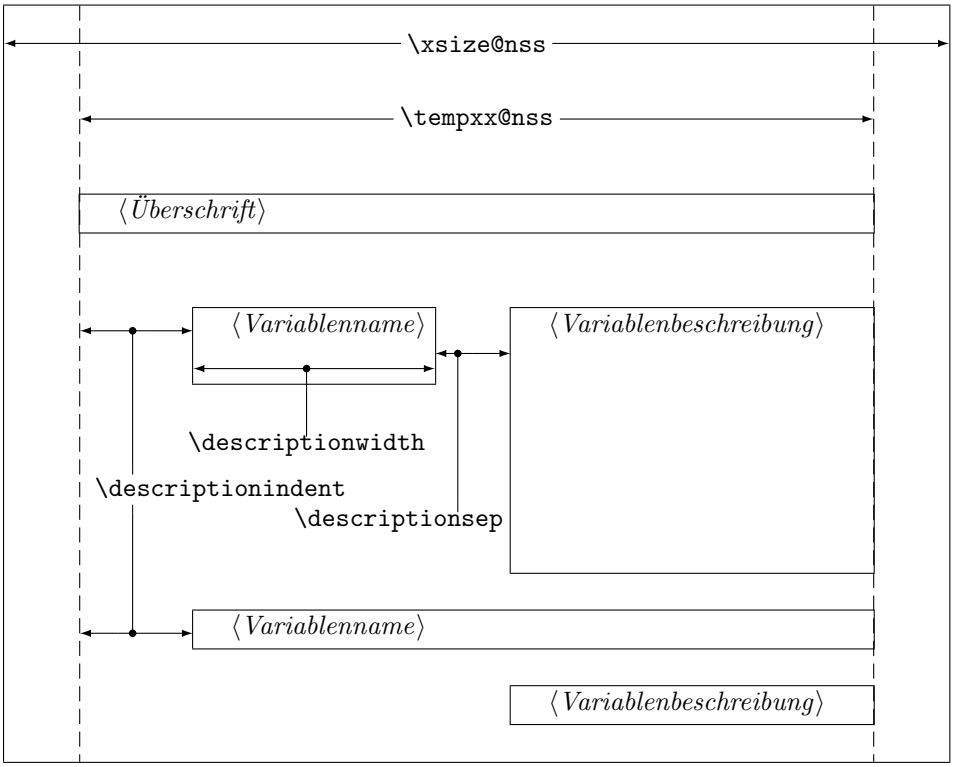


Abbildung 1: Aufbau einer Variablenbeschreibung

Speicherplatz bereitstellen:
iVar {eine *int* -Variable, deren Beschreibung hier allein dem Zweck dient, den Makro vorzuführen}

Nun werden Variablen genauer spezifiziert:

```
\begin{struktogramm}(95,50)
    \assign{%
        \begin{declaration}[Parameter:]
            \description{\pVar{iPar}}{ein \pKey{int}-Parameter,
                        dessen Bedeutung hier beschrieben wird}
        \end{declaration}
        \begin{declaration}[lokale Variablen:]
            \description{\pVar{iVar}}{eine \pKey{int}-Variable,
                        deren Bedeutung hier beschrieben wird}
            \description{\pVar{dVar}}{eine \pKey{double}-Variable,
                        deren Bedeutung hier beschrieben wird}
        \end{declaration}
    }
\end{struktogramm}
```

Das ergibt:

Parameter:	
iPar	{ein *int* -Parameter, dessen Bedeutung hier beschrieben wird}
lokale Variablen:	
iVar	{eine *int* -Variable, deren Bedeutung hier beschrieben wird}
dVar	{eine *double* -Variable, deren Bedeutung hier beschrieben wird}

Zuletzt die globale Vereinbarung eines Titels:

```
\def\declarationtitle{globale Variablen:}
\begin{struktogramm}(95,13)
    \assign{%
    \begin{declaration}
        \description{\pVar{iVar_g}}{eine \pKey{int}-Variable}
    \end{declaration}
  }
\end{struktogramm}
```

Dies ergibt das folgende Aussehen:

globale Variablen:	
iVar_g	{eine *int* -Variable}

Hier ist die lokale Umsetzung des \catcodes des Unterstrichs zu beachten, die erforderlich ist, wenn man einen Unterstrich in einem Makroargument einsetzen möchte. Diese lokale Umsetzung wird zwar auch schon bei \pVar gemacht, reicht aber bei der Makroexpansionstechnik von TEX nicht aus.

`\sub` `\return` Die Sinnbilder für einen Unterprogrammsprung und einen Aussprung aus dem Programm sehen ähnlich aus und werden mit folgenden Befehlen gezeichnet:

`\sub[`⟨*Höhe*⟩`]{`⟨*Text*⟩`}`

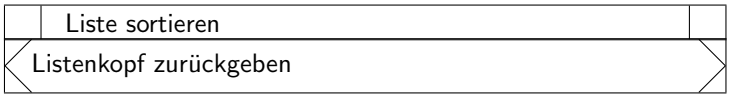
`\return[`⟨*Höhe*⟩`]{`⟨*Text*⟩`}`

Die Parameter haben dieselbe Bedeutung wie bei `\assign`. Das nächste Beispiel zeigt, wie diese Sinnbilder gezeichnet werden.

Beispiel 4

```
\begin{struktogramm}(95,20)
    \sub{Liste sortieren}
    \return{Listenkopf zurückgeben}
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



\while
\whileend
\until
\untilend
\forallin
\forallinend
\forever
\foreverend

Zum Darstellen von Schleifenkonstrukten stehen die drei Befehle `\while`, `\until` und `\forever` zur Verfügung. Die While-Schleife stellt eine Wiederholung mit vorausgehender Bedingungsprüfung (kopfgesteuerte Schleife) dar, die Until-Schleife testet die Bedingung am Schleifenende (fußgesteuerte Schleife) und die Forever-Schleife ist eine Endlosschleife, aus der man mit dem Befehl `\exit` herausspringen kann.

`\while[`⟨*Breite*⟩`]{`⟨*Text*⟩`}`⟨*Unterstruktogramm*⟩`\whileend`

`\until[`⟨*Breite*⟩`]{`⟨*Text*⟩`}`⟨*Unterstruktogramm*⟩`\untilend`

`\forever[`⟨*Breite*⟩`]`⟨*Unterstruktogramm*⟩`\foreverend`

`\exit[`⟨*Höhe*⟩`]{`⟨*Text*⟩`}`

⟨*Breite*⟩ ist die Dicke des Rahmens des Sinnbildes, ⟨*Text*⟩ ist der Bedingungstext, der in diesen Rahmen geschrieben wird. Wird die Breite nicht angegeben, richtet sich die Rahmendicke nach der Höhe des Textes. Der Text wird linksbündig in den Rahmen geschrieben. Ist der Text leer, so wird ein dünner Rahmen gezeichnet.

Ein Kontrollkonstrukt, das heute in vielen Programmiersprachen verfügbar ist, ist eine Schleife, die abhängig von der jeweiligen Sprache als `forall`-, `for ... in` oder `foreach`-Schleife bezeichnet wird. Diese Schleifenart kann prinzipiell als kopfgesteuerte Schleife angesehen werden, manche ziehen aber eine Schreibweise vor, die von der Endlosschleife abgeleitet wird. Für diesen Fall gibt es das Konstrukt

`\forallin[`⟨*Breite*⟩`]{`⟨*Text*⟩`}`⟨*Unterstruktogramm*⟩`\forallinend`

An Stelle von ⟨*Unterstruktogramm*⟩ können beliebige Befehle von StrukTEX stehen (mit Ausnahme von `\openstrukt` und `\closestrukt`), die das Struktogramm innerhalb der `\while`-, der `\until`- oder der `\forever`-Schleife bilden.

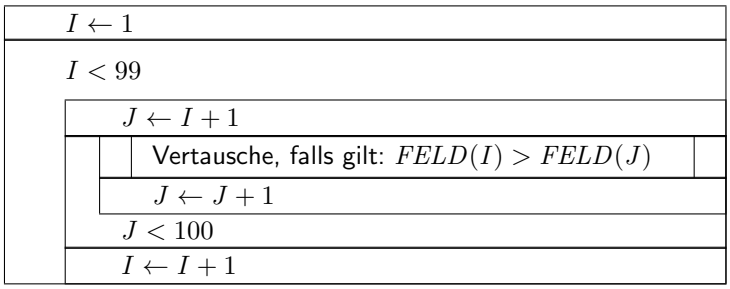
Um Kompatiblität mit der Weiterentwicklung des struktex.sty von J. Dietel zu erreichen, gibt es die Makros `\dfr` und `\dfrend` mit derselben Bedeutung wie `\forever` und `\foreverend`.

Die beiden folgenden Beispiele zeigen den Einsatz der `\while`- und `\until`-Makros sowie der `\forallin`-Makros, `\forever` wird weiter unten gezeigt.

Beispiel 5

```
\begin{struktogramm}(95,40)
    \assign{\(I \gets 1\)}
    \while[8]{\(I < 99\)}
        \assign{\(J \gets I+1\)}
        \until{\(J < 100\)}
            \sub{Vertausche, falls gilt: \(FELD(I) > FELD(J)\)}
            \assign{\(J \gets J+1\)}
        \untilend
        \assign{\(I \gets I+1\)}
    \whileend
 \end{struktogramm}
```

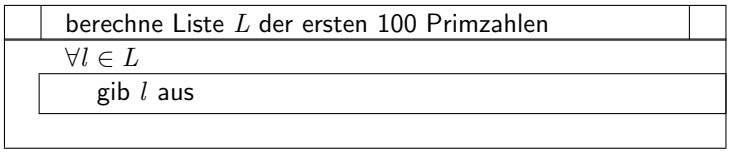
Diese Anweisungen führen zu folgendem Struktogramm:



Beispiel 6

```
\begin{struktogramm}(95, 25)
    \sub{berechne Liste \(L\) der ersten 100 Primzahlen}
    \forallin{\(\forall l\in L\)}
        \assign{gib \(l\) aus}
    \forallinend
 \end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



Die `\exit`-Anweisung gibt nur im Zusammenhang mit einfachen oder mehrfachen Verzweigungen Sinn, daher wird sie im Anschluss an die Diskussion der Verzweigungen vorgestellt.

`\ifthenelse` `\change` `\ifend`

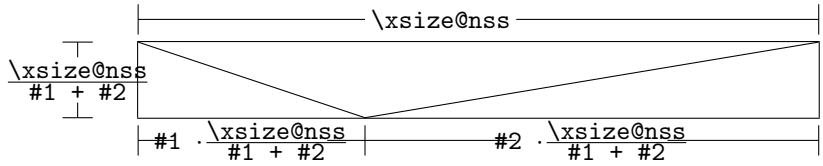
Zur Darstellung von Alternativen stellt StUkTEX die Sinnbilder für einen If-Then-Else-Block und für mehrfache Alternativen eine Case-Konstruktion zur Verfügung. Da in der klassischen `picture`-Umgebung von LATEX nur Linien mit bestimmten Steigungen gezeichnet werden können, muss der Benutzer bei beiden Befehlen selbst den Winkel bestimmen, mit dem die notwendigen schrägen Linien gezeichnet werden sollen (hier ist also etwas mehr ‚Handarbeit' nötig).

Wenn hingegen der curves.sty, der emlines2.sty oder der pict2e.sty eingesetzt wird, ist die Darstellung von Geraden mit beliebiger Steigung möglich.

Der If-Then-Else-Befehl sieht so aus:

`\ifthenelse[`⟨*Höhe*⟩`]{`⟨*Linker Winkel*⟩`}{`⟨*Rechter Winkel*⟩`}`
`{`⟨*Bedingung*⟩`}{`⟨*Linker Text*⟩`}{`⟨*Rechter Text*⟩`}`
⟨*Unterstruktogramm*⟩
`\change`
⟨*Unterstruktogramm*⟩
`\ifend`

Für den Fall, dass das optionale Argument ⟨*Höhe*⟩ nicht angegeben ist, sind ⟨*Linker Winkel*⟩ (`#1`) und ⟨*Rechter Winkel*⟩ (`#2`) Ziffern zwischen 1 und 6; diese bestimmen die Steigung der beiden Unterteilungslinien des If-Then-Else-Blocks (großer Wert = kleine Steigung). Größere Werte werden auf 6 gesetzt, kleinere auf 1. Das genaue Verhalten der Steigungen ist dem folgenden Bild zu entnehmen; `\xsize@nss` ist dabei die Breite des aktuellen Unterstruktogrammes. Wird die ⟨*Höhe*⟩ vorgegeben, so bestimmt dieser Wert statt des Ausdruckes $\frac{\texttt{\textbackslash xsize@nss}}{\texttt{\#1 + \#2}}$ die Höhe des Bedingungsrechteckes.

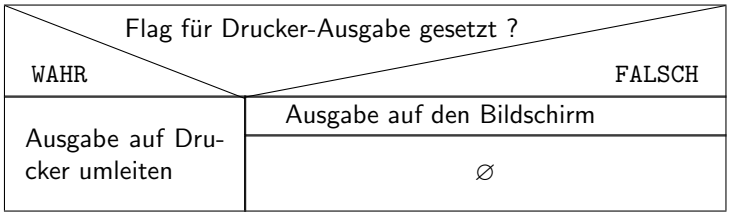


⟨*Bedingung*⟩ wird in das so gebildete obere mittlere Dreieck gesetzt; die Parameter ⟨*Linker Text*⟩ und ⟨*Rechter Text*⟩ werden in das linke bzw. rechte untere Dreieck gesetzt. Der Bedingungs-Text kann in seinem Dreiecks-Feld umgebrochen werden. Ab Version v-5.3 wird der Bedingungstext durch geeigneten Umbruch beliebigen Steigungen angepasst.[3] Die beiden anderen Texte sollten kurz sein (z. B. ja/nein oder true/false), da sie nicht umgebrochen werden können und sonst über ihr Dreiecks-Feld hinausragen. Um an dieser Stelle Einheitlichkeit zu erzielen, sollten die Makros `\pTrue` und `\pFalse` (vgl. Abschnitt 4.3) benutzt werden. Hinter `\ifthenelse` werden die Befehle für das linke, hinter `\change` die für das rechte „Unterstruktogramm“ geschrieben. Falls diese beiden Struktogramme nicht gleich lang sind, wird ein Kasten mit einem ∅ (vgl. Abschnitt 4.2) ergänzt.[4] Mit `\ifend` wird das If-Then-Else-Element beendet. Es folgen zwei Beispiele für die Anwendung.

Beispiel 7

```
\begin{struktogramm}(95,32)
    \ifthenelse[12]{1}{2}
        {Flag für Drucker-Ausgabe gesetzt ?}{\sTrue}{\sFalse}
        \assign[15]{Ausgabe auf Drucker umleiten}
    \change
        \assign{Ausgabe auf den Bildschirm}
    \ifend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:

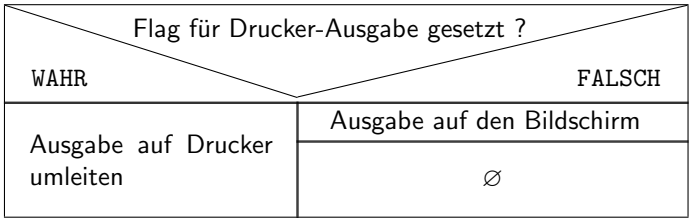


[3] Diese Erweiterung stammt von Daniel Hagedorn, dem ich hiermit herzlich danken möchte
[4] Eventuell ist ein `\strut` hilfreich, um unterschiedliche Höhen von Texten auszugleichen.

Beispiel 8

```
\begin{struktogramm}(90,30)
    \ifthenelse{3}{4}
        {Flag für Drucker-Ausgabe gesetzt ?}{\sTrue}{\sFalse}
        \assign[15]{Ausgabe auf Drucker umleiten}
    \change
        \assign{Ausgabe auf den Bildschirm}
    \ifend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



`\case` `\switch` `\caseend`

Das Case-Konstrukt hat folgende Syntax:

`\case[`⟨*Höhe*⟩`]{`⟨*Winkel*⟩`}{`⟨*Anzahl der Fälle*⟩`}{`⟨*Bedingung*⟩`}{`⟨*Text des 1. Falles*⟩`}`

⟨*Unterstruktogramm*⟩

`\switch[`⟨*Position*⟩`]{`⟨*Text des 2. Falles*⟩`}`

⟨*Unterstruktogramm*⟩

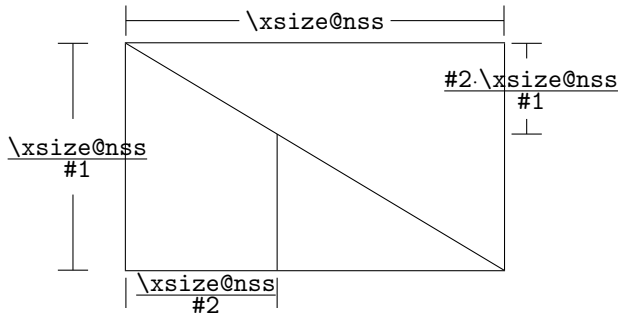
...

`\switch[`⟨*Position*⟩`]{`⟨*Text des n. Falles*⟩`}`

⟨*Unterstruktogramm*⟩

`\caseend`

Ist die ⟨*Höhe*⟩ nicht angegeben, so erhält die Unterteilungslinie des Case-Sinnbildes die durch ⟨*Winkel*⟩ angegebene Steigung (die bei `\ifthenelse` erwähnten Winkelwerte). In das obere der durch diese Linie entstandenen beiden Dreieck wird der Text ⟨*Bedingung*⟩ gesetzt. Die Größenverhältnisse ergeben sich aus der folgenden Skizze (`\xsize@nss` ist die aktuelle Breite des (Unter-)Struktogramms):

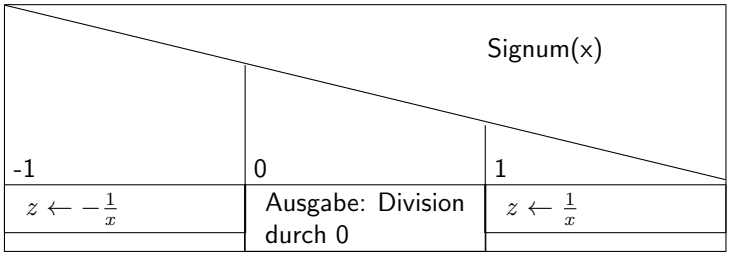


Der zweite Parameter ⟨*Anzahl der Fälle*⟩ gibt die Anzahl der zu zeichnenden Fälle an; alle Unterstruktogramme der einzelnen Fälle erhalten die gleiche Breite. Der ⟨*Text des 1. Falles*⟩ muss als Parameter des `\case`-Befehls angegeben werden, alle weiteren Fälle werden über den `\switch`-Befehl eingeleitet. Hinter dem Text folgen dann die Befehle für das eigentliche Unterstruktogramm des jeweiligen Falles. Der letzte Fall wird mit `\caseend` abgeschlossen. Ein Case-Sinnbild mit drei Fällen zeigt das folgende Beispiel.

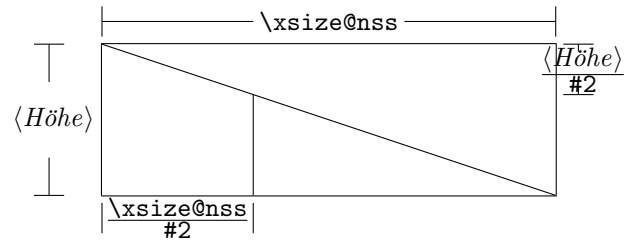
Beispiel 9

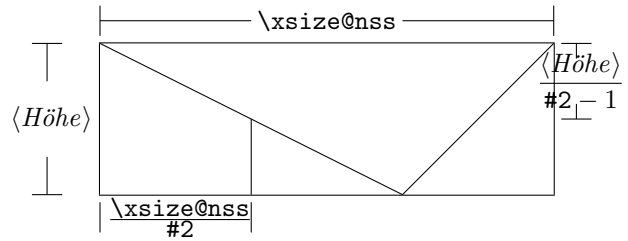
```
\begin{struktogramm}(95,30)
    \case{4}{3}{Signum(x)}{-1}
        \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
        \assign{Ausgabe: Division durch 0}
    \switch{1}
        \assign{\(z \gets \frac{1}{x}\)}
    \caseend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



Der optionale Parameter `[`⟨*Höhe*⟩`]` ist nur einzusetzen, wenn die Option „`curves`“, „`emlines2`“ oder „`pict2e`“ gesetzt ist; ist das nicht der Fall, können die Struktogramme durcheinander kommen. Die Erweiterung des `\switch`-Kommandos mit `[`⟨*Höhe*⟩`]` führt zu einer anderen Bedeutung von ⟨*Winkel*⟩. Ist der Wert gerade, wird wie zuvor eine gerade Linie zur Aufteilung des zugrundeliegenden Rechtecks gezeichnet; ist der Wert hingegen ungerade, wird der letzte Fall wie im folgenden gezeigt als Sonderfall gezeichnet.





Beispiel 10

```
\begin{struktogramm}(95,30)
    \case[10]{4}{3}{Signum(x)}{-1}
        \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
        \assign{Ausgabe: Division durch 0}
    \switch{1}
        \assign{\(z \gets \frac{1}{x}\)}
    \caseend
\end{struktogramm}
```

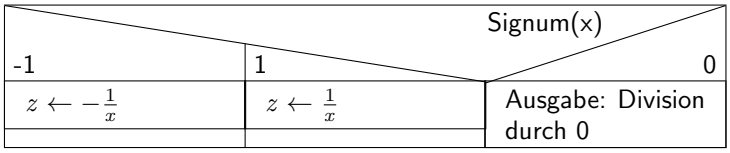
Diese Anweisungen führen zu folgendem Struktogramm:

Signum(x)		
-1	0	1
$z \leftarrow -\frac{1}{x}$	Ausgabe: Division durch 0	$z \leftarrow \frac{1}{x}$

Ist der erste Parameter hingegen ungerade, wird ein Standardzweig gezeichnet; der Wert für den Standardfall sollte dann rechtsbündig ausgerichtet werden.
Beispiel 11

```
\begin{struktogramm}(95,30)
    \case[10]{5}{3}{Signum(x)}{-1}
        \assign{\(z \gets - \frac{1}{x}\)}
    \switch{1}
        \assign{\(z \gets \frac{1}{x}\)}
    \switch[r]{0}
        \assign{Ausgabe: Division durch 0}
    \caseend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:

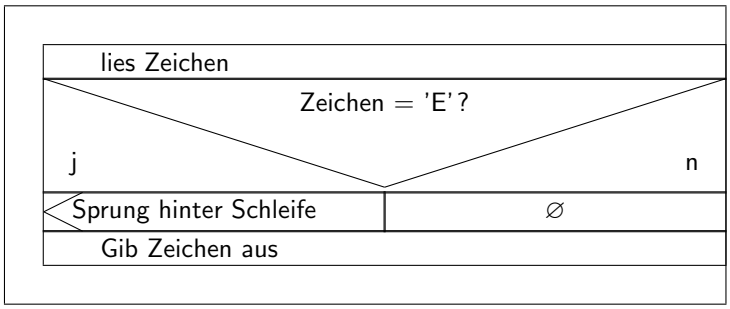


Das folgende Beispiel zeigt, wie mittels einfacher Verzweigung aus einer Endlosschleife gesprungen werden kann. Das Beispiel lässt sich ohne weiteres auf eine mehrfache Verzweigung übertragen.

Beispiel 12

```
\begin{struktogramm}(95,40)
    \forever
        \assign{lies Zeichen}
        \ifthenelse{3}{3}{Zeichen = 'E'?}
                         {j}{n}
            \exit{Sprung hinter Schleife}
        \change
        \ifend
        \assign{Gib Zeichen aus}
    \foreverend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



`\inparallel` `\task` `\inparallelend`

Heutzutage sind Prozessoren mit mehreren Kernen oder auch massive Parallelrechner ein übliches Werkzeug zur Ausführung von Programmen. Um die Fähigkeiten dieser Prozessoren auszunutzen, sollten entsprechende parallele Algorithmen entwickelt und implementiert werden. Der Makro `\inparallel` und die zugehörigen Makros `\task` und `\inparallelend` ermöglichen die Darstellung paralleler Verarbeitung in einem Programm. Die Syntax lautet:

`\inparallel[`⟨*Höhe der 1. Task*⟩`]{`⟨*Anzahl paralleler Tasks*⟩`}{`⟨*Beschreibung der 1. Task*⟩`}}`

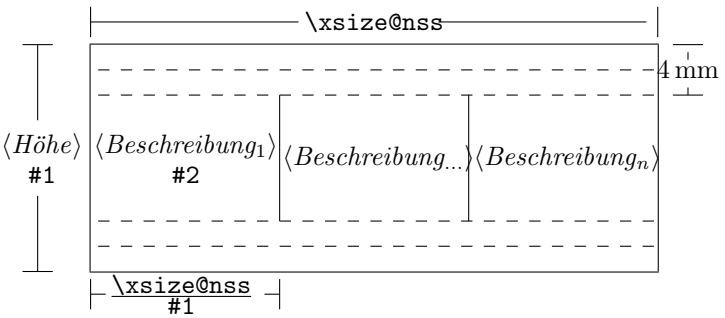
`\task[`⟨*position*⟩`]{`⟨*Beschreibung der 2. Task*⟩`}`

...

`\task[`⟨*position*⟩`]{`⟨*Beschreibung der n. Task*⟩`}`

`\inparallelend`

Das Layout eines mit diesen Kommandos erzeugten Kastens ist der folgenden Abbildung zu entnehmen (die Makroparameter `#1` und `#2` beziehen sich auf die Parameter von `\inparallel`):

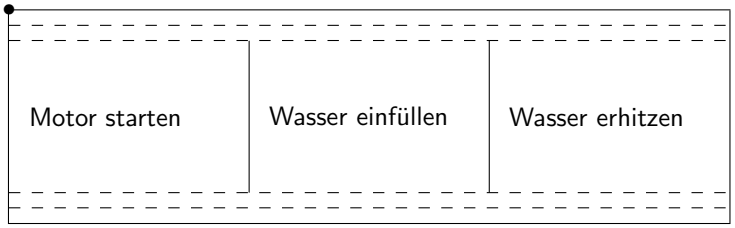


Zu beachten ist, dass die verschiedenen Tasks nicht weiter gegliedert werden dürfen.

Beispiel 13 (Anwendung von `\inparallel`)

```
\begin{struktogramm}(95,40)
    \inparallel[20]{3}{Motor starten}
    \task{Wasser einfüllen}
    \task{Wasser erhitzen}
    \inparallelend
\end{struktogramm}
```

Diese Anweisungen ergeben das folgende Struktogramm:



`centernss` (*env.*) Soll ein Struktogramm zentriert dargestellt werden, so wird dazu die Umgebung

```
\begin{centernss}
  ⟨Struktogramm⟩
\end{centernss}
```

benutzt:

```
\begin{centernss}
\begin{struktogramm}(90,35)
    \ifthenelse{2}{4}
            {Flag für Drucker-Ausgabe gesetzt?}{\sTrue}{\sFalse}%
```

```
        \assign[20]{Ausgabe auf Drucker umleiten}
    \change
        \assign{Ausgabe auf den Bildschirm}
    \ifend
\end{struktogramm}
\end{centernss}
```

Das führt zu folgendem:

Flag für Drucker-Ausgabe gesetzt?	
WAHR	FALSCH
Ausgabe auf Drucker umleiten	Ausgabe auf den Bildschirm
	∅

`\CenterNssFile` Häufig gibt es den Fall, dass Struktogramme in eigenen Dateien abgelegt werden, damit sie für sich allein auf Korrektheit getestet werden können oder in anderen Zusammenhängen genutzt werden können. Sollen sie zentriert eingebunden werden, kann nicht mit der folgenden Konstruktion gearbeitet werden:

```
\begin{center}
    \input{...}
\end{center}
```

da auf diese Weise der gesamte Text innerhalb des Struktogramms zentriert würde. Um diesen Fall einfach und korrekt abhandeln zu können, kann das Makro `\CenterNssFile` eingesetzt werden, das auch in der Schreibweise `centernssfile` definert ist. Voraussetzung ist, dass die Datei, die die Anweisungen für das Struktogramm enthält, die Dateinamenserweiterung `.nss` hat, der Name der einzubindenden Datei *muss* demzufolge ohne Erweiterung angegeben werden. Wenn die Datei struktex-tst-00.nss das in Abschnitt 5.3 auf Seite 35, Zeile 61–68 gezeigte Aussehen hat, so führt die Anweisung

```
\centernssfile{struktex-tst-00}
```

zu folgendem Aussehen des formatierten Textes:

Text

Signum(x)		
-1	0	1
$z \leftarrow -\frac{1}{x}$	Ausgabe: Division durch 0	$z \leftarrow \frac{1}{x}$

\openstrukt
\closestrukt

Diese beiden Makros sind nur der Kompatibilität zu vorherigen Versionen von StruktTEX willen noch erhalten. Von der Bedeutung her entsprechen sie `\struktogramm` und `\endstruktogramm`. Die Syntax ist

`\openstrukt{`⟨*width* ⟩`}{`⟨*height* ⟩`}`

und

`\closestrukt`.

\assert

Der Makro `\assert` wurde eingeführt, um die Verifikation von Algorithmen zu unterstützen, er ist aber nur aktiv, wenn die Stil-Option `verification` gesetzt wurde. Er dient dazu, an ausgewählten Stellen Zusicherungen über den Zustand von Variablen zu markieren, die Syntax entspricht dem `\assign`:

`\assert[`⟨*Höhe*⟩`]{`⟨*Zusicherung*⟩`}`,

Sein Einsatz ergibt sich aus dem folgenden:

```
\begin{struktogramm}(70,20)[Zusicherungen in Struktogrammen]
    \assign{\(a\gets a^2\)}
    \assert{\(a\ge0\)}
\end{struktogramm}
```

Das dazugehörige Struktogramm sieht folgendermaßen aus:

Zusicherungen in Struktogrammen

$a \leftarrow a^2$
$a \geq 0$

4.2 Spezielle Zeichen und Textdarstellung

\nat
\integer
\real
\complex
\emptyset

Wegen ihres häufigen Auftretens sind die Mengen der natürlichen, ganzen, rationalen, reellen und komplexen Zahlen ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ und $\mathbb{C}$) im Mathematik-Modus über die folgenden Makros erreichbar: `\nat`, `\integer`, `\real` und `\complex`. Ebenso ist das mit `\emptyset` erzeugte „$\emptyset$“ als Zeichen für die leere Anweisung auffälliger als das standardmäßige Zeichen „$\emptyset$“ Andere Mengensymbole wie $\mathbb{L}$ (für Lösungsmenge) sind über `\(\mathbb L\)` zu erzeugen.

\MathItalics
\MathNormal

Mit diesen beiden Makros kann die Darstellung von Variablennamen beeinflusst werden:

$$NeuerWert = AlterWert + Korrektur \tag{1}$$

```
\MathNormal
\begin{equation}
    NeuerWert = AlterWert + Korrektur
    \label{eq:-neuerwert-N}
\end{equation}
```

und

$$NeuerWert = AlterWert + Korrektur \tag{2}$$

```
\MathItalics
\begin{equation}
    NeuerWert = AlterWert + Korrektur
    \label{eq:-neuerwert-I}
\end{equation}
```

Diese Kommandos sind in einem eigenen Paket struktxf.sty definiert, das wie üblich geladen werden kann:

```
\usepackage[⟨Optionen⟩]{struktxf}
```

Die folgenden Optionen stehen zur Verfügung:

1. `iso`:

   Diese Option legt die Darstellung von Zahlenmengen in Übereinstimmung mit ISO 80 000-2 fest. Ist sie gesetzt, werden die Zahlenmengen durch fette Großbuchstaben wie **Q** dargestellt, ansonsten durch Zeichen aus dem `\mathbb` Zeichensatz wie $\mathbb{Q}$. Der Standardwert ist `iso=false`.

2. `mathitalics`:

   Das Setzen dieser Option führt dazu, dass Zuweisungen, wie sie bei der Beschreibung von Algorithmen auftreten, statt wie in Gleichung (1) auf der vorherigen Seite so wie in Gleichung (2) dargestellt werden.

   Es muss darauf hingewiesen werden, dass diese Option mit den Befehlen `\boldsymbol{...}` (aus amsmath.sty) bzw. `\bm{...}` (aus bm.sty) inkompatibel ist, vor diesen Befehlen `\MathNormal` aktiviert werden muss.

4.3 Makros zur Darstellung von Variablen, Schlüsselwörtern und anderen programmierspezifischen Details

`\pVariable` `\pVar` `\pKeyword` `\pKey` `\pComment`

Struktogramme enthalten manchmal direkt zu programmierenden Code. Um hier ein einheitliches Aussehen zu erreichen, sind die hier aufgeführten Makros definiert worden. Um diese Makros auch in anderem Zusammenhang nutzen zu können, sind sie zu einem eigenen *package* struktxp.sty zusammengefasst worden. Dabei wird ab Version 122 zur Darstellung von Code auf das Paket „url.sty“ von Donald Arsenau zurückgegriffen, das es ermöglicht, verbatim gesetzte Texte als Parameter an ein anderes Makro zu übergeben. Wenn diese Texte ein Leerzeichen enthalten, das erhalten bleiben soll, muss der Benutzer vor dem Laden von url.sty, typischerweise also vor der Anweisung

```
\usepackage{struktex}
```

die Anweisung

```
\PassOptionsToPackage{obeyspaces}{url}
```

setzen.

Mit `\pVariable{`⟨*Variablenname*⟩`}` wird ein Variablenname gesetzt. ⟨*Variablenname*⟩ ist dabei ein Bezeichner eine Variable, wobei der Unterstrich „`_`“, das kaufmännische Und „`&`“ und das Dach „`^`“ als Teile des Variablennamens erlaubt sind:

```
cEineNormaleVariable
c_eine_normale_Variable
&iAdresseEinerVariablen
zZeigerAufEineVariable^.sInhalt
```

```
\obeylines
 \renewcommand{\pLanguage}{C}
 \pVariable{cEineNormaleVariable}
 \pVariable{c_eine_normale_Variable}
 \pVariable{&iAdresseEinerVariablen}
 \renewcommand{\pLanguage}{Pascal}
 \pVariable{zZeigerAufEineVariable^.sInhalt}
```

Leerzeichen werden beachtet, so dass ganze Anweisungen geschrieben werden können. Es darf als Abkürzung `\pVar` benutzt werden.

Entsprechend wird mit `\pKeyword{`⟨*Schlüsselwort*⟩`}` ein Schlüsselwort gesetzt. Dabei ist ⟨*Schlüsselwort*⟩ ein Schlüsselwort in einer Programmiersprache, wobei der Unterstrich „`_`“ und das *hash*-Zeichen „`#`“ als Teil des Schlüsselwortes erlaubt ist. Damit lässt sich setzen:

```
begin
program
#include
```

```
\obeylines
 \pKeyword{begin}
 \renewcommand{\pLanguage}{Pascal}
 \pKeyword{program}
 \renewcommand{\pLanguage}{C}
 \pKeyword{#include}
```

Auch `\pKeyword` darf abgekürzt werden: `\pKey`. Damit erzeugt dann der Quelltext

```
\renewcommand{\pLanguage}{Pascal}
\pKey{begin} \pExp{iVar := iVar + 1;} \pKey{end}
```

als Ausgabe dieses Ergebnis:

```
begin iVar := iVar + 1; end
```

In ähnlicher Weise dient `\pComment` zur Darstellung von Kommentar. Das Argument darf nur Zeichen der TEX-Kategorie *letter* haben, Zeichen, die einen Kommentar einleiten, müssen geschrieben werden. `\pComment` kann nicht abgekürzt werden. Beispielsweise ergibt

```
\pExp{a = sqrt(a);} \pComment{// Iteration}
```

die Zeile

```
a = sqrt(a); // Iteration
```

`\pTrue` `\pFalse` `\pFonts` `\pBoolValue`

Logische Werte spielen in der Programmierung eine wesentliche Rolle. Mit `\pTrue` und `\pFalse` sind entsprechende Werte vorgegeben: `WAHR` und `FALSCH`.

Der Makro `\pFonts` dient der Auswahl von Fonts zur Darstellung von Variablen, Schlüsselwörtern und Kommentar:

`\pFonts{`⟨*Variablenfont*⟩`}{`⟨*Schlüsselwortfont*⟩`} {`⟨*Kommentarfont*⟩`}`

Vorbesetzt sind die einzelnen Fonts mit

- ⟨*Variablenfont*⟩ als `\small\sffamily`,
- ⟨*Schlüsselwortfont*⟩ als `\small\sffamily\bfseries` und
- ⟨*Kommentarfont*⟩ als `\small\sffamily\slshape`.

Damit wird die obige Zeile nach

```
\pFonts{\itshape}{\sffamily\bfseries}{\scshape}
\pVar{a = }\pKey{sqrt}\pVar{(a);} \pComment{// Iteration}
```

zu

a = sqrt (a); // Iteration

Entsprechend können durch den Makro

`\sBoolValue{`⟨*Ja-Wert*⟩`}{`⟨*Nein-Wert*⟩`}`

die Werte von `\pTrue` und `\pFalse` umdefiniert werden. Somit liefern die Zeilen

```
\renewcommand{\pLanguage}{Pascal}
\sBoolValue{\textit{ja}}{\textit{nein}}
\pFalse{} = \pKey{not} \pTrue
```

das folgende Ergebnis:

nein = not *ja*

`\sVar` `\sKey` `\sTrue` `\sFalse` Die Makros `\sVar` und `\sKey` sind mit den Makros `\pVar` und `\pKey` identisch, sie werden hier nur definiert, um Kompatibilität mit früheren Versionen des struktex.sty zu gewährleisten. Dasselbe gilt auch für die Makros `\sTrue` und `\sFalse`.

4.4 Makros zur Erstellung der Dokumentation des struktex.sty-Paketes

Um die Formatierung der Dokumentation ein wenig zu erleichtern, werden ein paar Makros benutzt, die in einer eigenen .sty-Datei zusammengefasst wurden. Ein wesentlicher Teil beruht auf einer Modifikation der `newtheorem`-Umgebung aus latex.sty zur Auszeichnung der Beispiele, die Implementation der Abkürzungen wurde in [Neu96] vorgeschlagen.

Dazu wurden aus dem verbatim.sty einige Kommandos übernommen und modifiziert, damit das Schreiben und Lesen von Dateien im `verbatim`-Modus auch im Zusammenhang mit dem `docstrip`-Paket funktioniert. Schließlich wurde auch noch eine Idee von Tobias Oetiker aus seinem layout.sty, der im Zusammenhang mit *lshort2e.tex - The not so short introduction to LaTeX2e* entstand, zum parallelen Setzen von Quelle und formatiertem Text genutzt.

```
1 ⟨*strukdoc⟩
2 \RequirePackage{ifpdf}
3 \newif\ifcolor
4 \IfFileExists{color.sty}{\colortrue}{}
5 \RequirePackage{varioref}
```

```
\ifpdf
    \RequirePackage[colorlinks]{hyperref}
\else
    \def\href#1{\texttt}
\fi
\RequirePackage{cleveref}
\ifcolor
    \RequirePackage{color}
\fi
\RequirePackage{url}
\renewcommand\ref{\protect\T@ref}
\renewcommand\pageref{\protect\T@pageref}
\@ifundefined{zB}{}{\endinput}
\providecommand\pparg[2]{%
  {\ttfamily(}\meta{#1},\meta{#2}{\ttfamily)}}
\providecommand\envb[1]{%
  {\ttfamily\char`\\begin\char`\{#1\char`\}}}
\providecommand\enve[1]{%
  {\ttfamily\char`\\end\char`\{#1\char`\}}}
\newcommand{\zBspace}{z.\,B.}
\let\zB=\zBspace
\newcommand{\dhspace}{d.\,h.}
\let\dh=\dhspace
\let\foreign=\textit
\newcommand\Abb[1]{Abbildung~\ref{#1}}
\newcommand\sFile[1]{\textsf{#1}}
\def\newexample#1{%
  \@ifnextchar[{\@oexmpl{#1}}{\@nexmpl{#1}}}
\def\@nexmpl#1#2{%
  \@ifnextchar[{\@xnexmpl{#1}{#2}}{\@ynexmpl{#1}{#2}}}
\def\@xnexmpl#1#2[#3]{%
  \expandafter\@ifdefinable\csname #1\endcsname
    {\@definecounter{#1}\@newctr{#1}[#3]%
     \expandafter\xdef\csname the#1\endcsname{%
       \expandafter\noexpand\csname the#3\endcsname \@exmplcountersep
          \@exmplcounter{#1}}%
     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
     \global\@namedef{end#1}{\@endexample}}}
\def\@ynexmpl#1#2{%
  \expandafter\@ifdefinable\csname #1\endcsname
    {\@definecounter{#1}%
     \expandafter\xdef\csname the#1\endcsname{\@exmplcounter{#1}}%
     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
     \global\@namedef{end#1}{\@endexample}}}
\def\@oexmpl#1[#2]#3{%
  \@ifundefined{c@#2}{\@nocounterr{#2}}%
    {\expandafter\@ifdefinable\csname #1\endcsname
    {\global\@namedef{the#1}{\@nameuse{the#2}}%
  \global\@namedef{#1}{\@exmpl{#2}{#3}}%
  \global\@namedef{end#1}{\@endexample}}}}
\def\@exmpl#1#2{%
  \refstepcounter{#1}%
  \@ifnextchar[{\@yexmpl{#1}{#2}}{\@xexmpl{#1}{#2}}}
\def\@xexmpl#1#2{%
```

```
  \@beginexample{#2}{\csname the#1\endcsname}\ignorespaces}
\def\@yexmpl#1#2[#3]{%
  \@opargbeginexample{#2}{\csname the#1\endcsname}{#3}\ignorespaces}
\def\@exmplcounter#1{\noexpand\arabic{#1}}
\def\@exmplcountersep{.}
\def\@beginexample#1#2{%
    \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
    \item[{\bfseries #1\ #2}]\mbox{}\\\sf}
\def\@opargbeginexample#1#2#3{%
    \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
    \item[{\bfseries #1\ #2}\ (#3)]\mbox{}\\\sf}
\def\@endexample{\endlist}

\newexample{tExample}{\ifnum\language=\languageNGerman Beispiel\else Example\fi}

\newwrite\struktex@out
\newenvironment{example}%
 {\begingroup% Lets keep the changes local
  \@bsphack
  \immediate\openout \struktex@out \jobname.tmp
  \let\do\@makeother\dospecials\catcode`\^^M\active
  \def\verbatim@processline{%
    \immediate\write\struktex@out{\the\verbatim@line}}%
  \verbatim@start}%
 {\immediate\closeout\struktex@out\@esphack\endgroup%
%
% And here comes the part of Tobias Oetiker
%
  \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
  \noindent
  \makebox[0.45\linewidth][l]{%
  \begin{minipage}[t]{0.45\linewidth}
    \vspace*{-2ex}
    \setlength{\parindent}{0pt}
    \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
    \begin{trivlist}
      \item\input{\jobname.tmp}
    \end{trivlist}
  \end{minipage}}%
  \hfill%
  \makebox[0.5\linewidth][l]{%
  \begin{minipage}[t]{0.50\linewidth}
    \vspace*{-1ex}
    \verbatiminput{\jobname.tmp}
  \end{minipage}}
  \par\addvspace{3ex plus 1ex}\vskip -\parskip
}

\newtoks\verbatim@line
\def\verbatim@startline{\verbatim@line{}}
\def\verbatim@addtoline#1{%
  \verbatim@line\expandafter{\the\verbatim@line#1}}
\def\verbatim@processline{\the\verbatim@line\par}
\def\verbatim@finish{\ifcat$\the\verbatim@line$\else
```

```
  \verbatim@processline\fi}

\def\verbatimwrite#1{%
  \@bsphack
  \immediate\openout \struktex@out #1
  \let\do\@makeother\dospecials
  \catcode`\^^M\active \catcode`\^^I=12
  \def\verbatim@processline{%
    \immediate\write\struktex@out
      {\the\verbatim@line}}%
  \verbatim@start}
\def\endverbatimwrite{%
  \immediate\closeout\struktex@out
  \@esphack}

\@ifundefined{vrb@catcodes}%
  {\def\vrb@catcodes{%
     \catcode`\!12\catcode`\[12\catcode`\]12}}{}
\begingroup
 \vrb@catcodes
 \lccode`\!=`\\ \lccode`\[=`\{ \lccode`\]=`\}
 \catcode`\~=\active \lccode`\~=`\^^M
 \lccode`\C=`\C
 \lowercase{\endgroup
    \def\verbatim@start#1{%
      \verbatim@startline
      \if\noexpand#1\noexpand~%
        \let\next\verbatim@
      \else \def\next{\verbatim@@#1}\fi
      \next}%
    \def\verbatim@@#1~{\verbatim@@@#1!end\@nil}%
    \def\verbatim@@@#1!end{%
       \verbatim@addtoline{#1}%
       \futurelet\next\verbatim@@@}%
    \def\verbatim@@@@#1\@nil{%
       \ifx\next\@nil
         \verbatim@processline
         \verbatim@startline
         \let\next\verbatim@
       \else
         \def\@tempa##1!end\@nil{##1}%
         \@temptokena{!end}%
         \def\next{\expandafter\verbatim@test\@tempa#1\@nil~}%
       \fi \next}%
    \def\verbatim@test#1{%
           \let\next\verbatim@test
           \if\noexpand#1\noexpand~%
             \expandafter\verbatim@addtoline
               \expandafter{\the\@temptokena}%
             \verbatim@processline
             \verbatim@startline
             \let\next\verbatim@
           \else \if\noexpand#1
             \@temptokena\expandafter{\the\@temptokena#1}%
```

```
          \else \if\noexpand#1\noexpand[%
            \let\@tempc\@empty
            \let\next\verbatim@testend
          \else
            \expandafter\verbatim@addtoline
              \expandafter{\the\@temptokena}%
            \def\next{\verbatim@#1}%
          \fi\fi\fi
          \next}%
    \def\verbatim@testend#1{%
         \if\noexpand#1\noexpand~%
           \expandafter\verbatim@addtoline
             \expandafter{\the\@temptokena[}%
           \expandafter\verbatim@addtoline
             \expandafter{\@tempc}%
           \verbatim@processline
           \verbatim@startline
           \let\next\verbatim@
         \else\if\noexpand#1\noexpand]%
           \let\next\verbatim@@testend
         \else\if\noexpand#1\noexpand!%
           \expandafter\verbatim@addtoline
             \expandafter{\the\@temptokena[}%
           \expandafter\verbatim@addtoline
             \expandafter{\@tempc}%
           \def\next{\verbatim@!}%
         \else \expandafter\def\expandafter\@tempc\expandafter
           {\@tempc#1}\fi\fi\fi
         \next}%
    \def\verbatim@@testend{%
       \ifx\@tempc\@currenvir
         \verbatim@finish
         \edef\next{\noexpand\end{\@currenvir}%
                    \noexpand\verbatim@rescan{\@currenvir}}%
       \else
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena[}%
           \expandafter\verbatim@addtoline
             \expandafter{\@tempc]}%
         \let\next\verbatim@
       \fi
       \next}%
    \def\verbatim@rescan#1#2~{\if\noexpand~\noexpand#2~\else
        \@warning{Characters dropped after `\string\end{#1}'}\fi}}

\newread\verbatim@in@stream
\def\verbatim@readfile#1{%
  \verbatim@startline
  \openin\verbatim@in@stream #1\relax
  \ifeof\verbatim@in@stream
    \typeout{No file #1.}%
  \else
    \@addtofilelist{#1}%
    \ProvidesFile{#1}[(verbatim)]%
```

```
    \expandafter\endlinechar\expandafter\m@ne
    \expandafter\verbatim@read@file
    \expandafter\endlinechar\the\endlinechar\relax
    \closein\verbatim@in@stream
  \fi
  \verbatim@finish
}
\def\verbatim@read@file{%
  \read\verbatim@in@stream to\next
  \ifeof\verbatim@in@stream
  \else
    \expandafter\verbatim@addtoline\expandafter{\expandafter\check@percent\next}%
    \verbatim@processline
    \verbatim@startline
    \expandafter\verbatim@read@file
  \fi
}
\def\verbatiminput{\begingroup\MacroFont
  \@ifstar{\verbatim@input\relax}%
          {\verbatim@input{\frenchspacing\@vobeyspaces}}}
\def\verbatim@input#1#2{%
   \IfFileExists {#2}{\@verbatim #1\relax
    \verbatim@readfile{\@filef@und}\endtrivlist\endgroup\@doendpe}%
   {\typeout {No file #2.}\endgroup}}
```
⟨/strukdoc⟩

5 Verschiedene Beispieldateien

5.1 Datei zum Austesten der Makros des Paketes **struktxf.sty**

```
⟨*struktxf-00⟩
\documentclass[fleqn, 12pt, english]{article} % 12pt for better readability

\usepackage{babel}
\usepackage[iso, mathitalics]{struktxf} % the options could be omitted,
                                        % they represent the default

\nofiles

\begin{document}

\section*{Mathematical Sets:}

\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}
To compare the fonts visually, should not be done in real documents:
\begin{verbatim}
```

```
\makeatletter\@nss@isofalse\makeatother
\end{verbatim}
\makeatletter\@nss@isofalse\makeatother
\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}
or (with \verb|\@nss@isotrue|)
\makeatletter\@nss@isotrue\makeatother
\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}

\section*{Assignments:}

\begin{itemize}
  \item Default settings:
    \begin{equation}
        \label{eq:def}
        NewValue = OldValue + Correction
    \end{equation}
  \item \verb|MathNormal| settings: \MathNormal
    \begin{equation}
        \label{eq:norm}
        NewValue = OldValue + Correction
    \end{equation}
  \item \verb|MathItalics| settings: \MathItalics
    \begin{equation}
        \label{eq:ital}
        NewValue = OldValue + Correction
    \end{equation}
\end{itemize}

\end{document}
```
⟨/struktxf-00⟩

5.2 Datei zum Austesten der Makros des Paketes **struktxp.sty**

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros des struktxp.sty benutzt werden kann. Zum Testen sollten auch die Kommentarzeichen vor der Zeile `\usepackage[T1]{fontenc}` gelöscht werden. Der Text ist nur in Englisch vorgegeben.

```
⟨*example3⟩
\documentclass[english]{article}

\usepackage{babel}
```

```
\usepackage{struktxf}
\usepackage{struktxp}

\nofiles

\begin{document}

\pLanguage{Pascal}
\section*{Default values (Pascal):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{yes}{no}-:

{\obeylines
\pBoolValue{yes}{no}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
```

```
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{C}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (C):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
```

```
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{Java}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (Java):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

```

```
{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{Python}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (Python):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
```

```
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\end{document}
⟨/example3⟩
```

5.3 Beipieldatei zum Einbinden in die Dokumentation: Positionierung von Struktogrammen mit `\centernss`/`\input`

Die folgenden Zeilen bilden eine Beispieldatei, die bei der Erstellung dieser Dokumentation (siehe S. 19) benötigt wird.

```
⟨*nss-sample⟩
\begin{struktogramm}(95,40)[Text]
    \case[10]{3}{3}{Signum(x)}{-1}
        \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
        \ifnum\language=\languageNGerman%
            \assign{Ausgabe: Division durch 0}
        \else
            \assign{Output: Division by 0}
        \fi
    \switch[r]{1}
        \assign{\(z \gets \frac{1}{x}\)}
    \caseend
\end{struktogramm}
⟨/nss-sample⟩
```

5.4 Darstellung von Fallunterscheidungen mit vordefinierten Steigungen ohne die Benutzung optionaler Pakete

Die folgenden Zeilen bilden eine Beispieldatei, die zum Austesten der Makros benutzt werden kann. Es wird gezeigt, wie die Steigung von Fallunterscheidungen ohne optionale Zusatzpakete aussehen kann. Der Inhalt ist nur in Englisch vorhanden.

```
⟨*example1⟩
\documentclass[draft]{article}
\usepackage{struktex}

\begin{document}

\begin{struktogramm}(90,137)
```

```
    \assign%
    {
      \begin{declaration}[]
          \description{\(a, b, c\)}{three variables which are to be sorted}
          \description{\(tmp\)}{temporary variable for the circular swap}
      \end{declaration}
    }
    \ifthenelse{1}{2}{\(a\le c\)}{j}{n}
    \change
    \assign{\(tmp\gets a\)}
    \assign{\(a\gets c\)}
    \assign{\(c\gets tmp\)}
    \ifend
    \ifthenelse{2}{1}{\(a\le b\)}{j}{n}
    \ifthenelse{1}{1}{\(b\le c\)}{j}{n}
    \change
    \assign{\(tmp\gets c\)}
    \assign{\(c\gets b\)}
    \assign{\(b\gets tmp\)}
    \ifend
    \change
    \assign{\(tmp\gets a\)}
    \assign{\(a\gets b\)}
    \assign{\(b\gets tmp\)}
    \ifend
\end{struktogramm}

\end{document}
```
⟨/example1⟩

5.5 Darstellung von Fallunterscheidungen mit variablen Steigungen mit dem Paket pict2e.sty

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros benutzt werden kann. Es wird gezeigt, wie die Steigung automatisch berechnet wird, wenn das Pakete pict2e geladen wird. Der Inhalt ist nur in Englisch vorhanden.

⟨∗example2⟩
```
\documentclass{article}
\usepackage[pict2e, verification]{struktex}

\begin{document}
\def\StruktBoxHeight{7}
%\sProofOn{}
\begin{struktogramm}(90,137)
    \assign%
    {
      \begin{declaration}[]
          \description{\(a, b, c\)}{three variables which are to be sorted}
          \description{\(tmp\)}{temporary variable for the circular swap}
      \end{declaration}
    }
    \assert[\StruktBoxHeight]{\sTrue}
    \ifthenelse[\StruktBoxHeight]{1}{2}{\(a\le c\)}{j}{n}
```

```
        \assert[\StruktBoxHeight]{\(a\le c\)}
    \change
        \assert[\StruktBoxHeight]{\(a>c\)}
        \assign[\StruktBoxHeight]{\(tmp\gets a\)}
        \assign[\StruktBoxHeight]{\(a\gets c\)}
        \assign[\StruktBoxHeight]{\(c\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le c\)}
    \ifthenelse[\StruktBoxHeight]{2}{1}{\(a\le b\)}{j}{n}
        \assert[\StruktBoxHeight]{\(a\le b \wedge a\le c\)}
        \ifthenelse[\StruktBoxHeight]{1}{1}{\(b\le c\)}{j}{n}
            \assert[\StruktBoxHeight]{\(a\le b \le c\)}
        \change
            \assert[\StruktBoxHeight]{\(a \le  c<b\)}
            \assign[\StruktBoxHeight]{\(tmp\gets c\)}
            \assign[\StruktBoxHeight]{\(c\gets b\)}
            \assign[\StruktBoxHeight]{\(b\gets tmp\)}
            \assert[\StruktBoxHeight]{\(a\le b<c\)}
        \ifend
    \change
        \assert[\StruktBoxHeight]{\(b < a\le c\)}
        \assign[\StruktBoxHeight]{\(tmp\gets a\)}
        \assign[\StruktBoxHeight]{\(a\gets b\)}
        \assign[\StruktBoxHeight]{\(b\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<b\le c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le b \le c\)}
\end{struktogramm}

\end{document}
⟨/example2⟩
```

5.6 Dokumentation von Java Methoden im Vorspann eines Struktogramms mit dem Paket **struktxp.sty**

Der folgende Code wird in einem anderen Zusammenhang benutzt, nämlich um Java-Methoden zu dokumentieren. An dieser Stelle wird ein eigener Weg gewählt, da das sonst übliche Arbeiten mit `\lstinline` aus dem listings Paket zu Fehlern führt.

Der Code zeigt zusätzlich, wie die dokumentierten Methoden im Index aufgeführt werden können.

```
⟨*struktxp-00⟩
\documentclass{article}

\usepackage{index} % produce the index afterwards with the command
                   % makeindex -o struktxp-tst-00.ind struktxp-tst-00.idx
\usepackage{struktxp}

\makeindex

\makeatletter
\newlength{\fdesc@len}
```

```
\newcommand{\fdesc@label}[1]%
{%
    \settowidth{\fdesc@len}{{\fdesc@font #1}}%
    \advance\hsize by -2em
    \ifdim\fdesc@len>\hsize%               % term > labelwidth
        \parbox[b]{\hsize}%
        {%
            \fdesc@font #1%
        }\\%
    \else%                                      % term < labelwidth
    \ifdim\fdesc@len>\labelwidth%               % term > labelwidth
        \parbox[b]{\labelwidth}%
        {%
            \makebox[0pt][l]{{\fdesc@font #1}}\\%
        }%
    \else%                                      % term < labelwidth
        {\fdesc@font #1}%
    \fi\fi%
    \hfil\relax%
}
\newenvironment{fdescription}[1][\tt]%
{%
    \def\fdesc@font{#1}
    \begin{quote}%
    \begin{list}{}%
    {%
        \renewcommand{\makelabel}{\fdesc@label}%
        \setlength{\labelwidth}{120pt}%
        \setlength{\leftmargin}{\labelwidth}%
        \addtolength{\leftmargin}{\labelsep}%
    }%
}%
{%
    \end{list}%
    \end{quote}%
}
\makeatother

\pLanguage{Java}

\begin{document}

\begin{fdescription}
 \item[\index{Methods!drawImage(Image img,
                                    int dx1,
                                    int dy1,
                                    int dx2,
                                    int dy2,
                                    int sx1,
                                    int sy1,
                                    int sx2,
                                    int sy2,
                                    ImageObserver observer)@%
       \pExp{\pKey{public} \pKey{abstract} \pKey{boolean} drawImage(Image img,
```

```
                                    \pKey{int} dx1,
                                    \pKey{int} dy1,
                                    \pKey{int} dx2,
                                    \pKey{int} dy2,
                                    \pKey{int} sx1,
                                    \pKey{int} sy1,
                                    \pKey{int} sx2,
                                    \pKey{int} sy2,
                                    ImageObserver observer)}}%
       \pExp{public abstract boolean drawImage(Image img, int dx1, int
         dy1, int dx1, int dy2, int sx1, int sy1, int sx2, int sy2,
         ImageObserver observer)}]%
 \ldots
\end{fdescription}

\printindex
\end{document}
```
⟨/struktxp-00⟩

6 Prüfen und Erstellen des Pakets struktex.sty mit l3build

Der Umgang mit .dtx-Paketen ist wesentlich einfacher, wenn ein Werkzeug für die Automatisierung der wesentlichen Schritte vorliegt. Ein solches Werkzeug ist mit dem Paket l3build plattformübergreifend gegeben [LaT20].

Die Anpassung an die Gegebenheiten für ein spezielles Projekt erfolgt durch ein build.lua-Skript, das bei Bedarf durch eine Konfigurationsdatei build-config.lua begleitet wird. Beide Dateien sind hier in die Dokumentation integriert, auf ihre Benutzung wird hier nicht weiter eingegangen, sie ist in [LaT20] nachzulesen.

⟨*l3build⟩
```
#!/usr/bin/env texlua

-- Build script for "StrukTeX" files

-- Identify the bundle and module: the module may be empty in the case where
-- there is no subdivision
bundle = ""
module = "struktex"

-- Location of main directory: use Unix-style path separators
-- Should match that defined by the bundle.
maindir = "."

-- the files to be installed
installfiles = {"*.sty"}

-- the files which should not be installed
excludefiles = {"*/TODO.md"}

-- these files are to copy for unpackaging
sourcefiles = {"*.dtx", "*.ins", "build.lua", "build-config.lua"}

```

```
753 -- these files are part of the documentation
754 docfiles = {"struktex.drv", "struktex.sty"}
755
756 -- these files get tags
757 tagfiles = {"*.sty"}
758
759 -- this file must be TeXed to unpack all
760 unpackfiles = {"struktex.dtx"}
761 -- unpackopts = "-interaction=batchmode"
762
763 -- create a TDS-suited .zip-file
764 packtdszip = true
765
766 -- Files to copy to unpacking when typesetting
767 typesetsourcefiles = {"struktex.sty"}
768
769 -- Files needed to support typesetting when "sandboxed"?
770 typesetsuppfiles = {"tex-bib.bib"}
771
772 -- only using the following engines for testing the code
773 checkengines    = checkengines or {"latex", "pdftex"}
774
775 -- install also documentation (doesn't work, because
776 -- there are no *.pdf-files in the unpacked directory
777 installfiles     = installfiles or {"*.pdf", "*.sty"}
778
779 -- Load the common build code
780 dofile(maindir .. "/build-config.lua")
781
782 -- specific settings for the StrukTeX development repo, used by l3build script
783
784 function prepare_contents(file)
785   -- can't be set locally, so: set it back later on!
786   typesetopts = "--output-format=dvi"
787   tex(file)
788
789   -- create .pdf-file from .dvi-source
790   name = stripext(file)
791   dvitopdf(name, typesetdir, typesetexe, true)
792   run(typesetdir, "pdfcrop " .. name .. pdfext)
793   ren(typesetdir, name .. "-crop" .. pdfext, name .. pdfext)
794 end
795
796 function print_log(log_message)
797   print(log_message)
798   return 0
799 end
800
801 function typeset(file)
802   local name = jobname(file)
803
804   typesetopts = "--output-format=pdf"
805   local errorlevel = tex(file, typesetdir)
806
```

```
807   print("\ntypeset.errorlevel:", errorlevel, "\n")
808
809   if errorlevel == 0 then
810     -- Return a non-zero errorlevel if anything goes wrong
811     print("prepare additional information")
812     errorlevel = (
813       print_log("file: " .. file .. ", name: " .. name) +
814         print_log("going to bibtex with >" ..  name .. "<") +
815         bibtex(name, typesetdir) +
816         print_log("\nbibtex done\n") +
817         tex(file, typesetdir) +
818         print_log("going to makeindex with", name) +
819         makeindex(name, typesetdir, ".idx", ".ind", ".ilg", "gind.ist") +
820         makeindex(name, typesetdir, ".glo", ".gls", ".glg", "gglo.ist") +
821         print_log("\nmakeindex done\n") +
822         tex(file, typesetdir)
823                 )
824     print_log("\ntypesetting done\n")
825     ren(typesetdir, "struktex.pdf", "struktex." .. targetlanguage .. ".pdf")
826     print_log("\nrenaming to struktex." .. targetlanguage .. ".pdf done\n")
827   end
828
829   return errorlevel
830 end
831
832 function setversion_update_line(line, date, release)
833   local function get_date_release()
834     -- get version information from git
835     local f = io.popen("git describe", "r")
836     local release = {}
837     for entry in f:lines() do
838       release[#release + 1] = entry
839     end
840     return os.date("%Y/%m/%d", os.time()), release[1]:sub(2, -1)
841   end
842
843   date, release = get_date_release()
844
845   for _,i in pairs({"Class", "File", "Package"}) do
846     if string.match(
847       line,
848       "^\\ProvidesExpl" .. i .. " *{[a-zA-Z0-9%-%.]+}"
849     ) then
850       line = string.gsub(
851         line,
852         "{%d%d%d%d/%d%d/%d%d}( *){[^}]*}",
853         "{" .. string.gsub(date, "%-", "/") .. "}%1{" .. release .. "}"
854       )
855       break
856     end
857   end
858   return line
859 end
860
```

```
--###  doc_init_hook begin  ##########################################
--[[
  The git informations (commit name, commit date, commit author)
  must be included in the .sty files. This is done here.
  1. git describe --long HEAD: returns the most recent tag (HEAD
       can be omitted, it's the default), that's the commit name
  2. git --no-pager show -s --format=format:"<format>": returns
       the needed commit date and commit author. <format> is set
       up so that the three lines below are generated

  These informations replace the line
  %% git revision information
  by someting like
  \@git@ $Date: <commit date> $%
     $Revision: <commit name> $
  %% $Author: <commit author> $

  <commit date> has the format "yyyy-mm-dd hh:mm:ss +nnnn"
  <commit name> has the format as described in the git describe
    man page
--]]
function docinit_hook()
  print("now set git data")
  local f = io.popen('git describe --long HEAD', 'r')
  local commit_name = f:read "*all"
  commit_name = string.sub(commit_name, 1, #commit_name - 1)

  f = io.popen('git --no-pager show -s --format=format:"%n  ' ..
                '\\@git@ \\$Date: %ci \\$%%%%%n  \\$Revision: ' ..
                '<commit name> \\$%n  %%%%%%%% \\$Author: %cn \\$%n"')
  local revision_information = f:read "*all"
  revision_information = string.sub(revision_information, 1,
      #revision_information - 1)

  print("commit_name: >>" .. commit_name .. "<<, Länge: " ..
      tostring(string.len(commit_name)))
  print("revision_information: " .. revision_information)

  revision_information = revision_information:gsub("<commit name>", commit_name)
  print("revision_information: " .. revision_information)

  filenames = {"strukdoc.sty", "struktex.sty", "struktxf.sty", "struktxp.sty"}

  for _, file in ipairs(filenames) do
    file = unpackdir .. "/" .. file
    print(file)

    local f = assert(io.open(file,"rb"))
    local content = f:read("*all")
    f:close()

    content = content:gsub("%%%% git revision information",
                           revision_information)
    print(content)
```

```
915
916     f = assert(io.open(file,"w"))
917     f:write(content)
918     f:close()
919
920     if file == unpackdir .."/" .. "struktex.sty" then
921       file = file:gsub(unpackdir, typesetdir)
922       f = assert(io.open(file,"w"))
923       f:write(content)
924       f:close()
925     end
926   end
927
928   -- this file is needed for the documentation but isn't automatically
929   -- copied
930   cp("struktex-tst-00.nss", unpackdir, typesetdir)
931
932   return 0
933 end
934 --###  doc_init_hook end  ##############################################
935
936 -- Find and run the build system
937 kpse.set_program_name("kpsewhich")
938 if not release_date then
939   dofile(kpse.lookup("l3build.lua"))
940 end
941
942 ⟨/lbuild⟩
```

Die folgende Zeilen, die nach Ausführen des Befehls `pdftex struktex.dtx` als Datei build-config.lua vorliegen, dienen der Umschaltung der Dokumentationssprache zwischen Deutsch und Englisch.

```
943 ⟨*config⟩
944 -- configuration file for the StrukTeX package
945
```

„`\z`“ ermöglicht den Umbruch von Zeichenketten.

```
946 print("This is the additional configuration for setting \z
947     the documentation language")
948
```

An dieser Stelle kann die Zielsprache die Dokumentation eingestellt werden, erlaubt ist „de“ oder „en“. `targetlanguage` muss global gelten, da der Wert am Ende des Dokumentationslaufes benötigt wird.

```
949 targetlanguage = targetlanguage or "de"
950 print("target language: >" .. targetlanguage .. "<")
951
952 local exit = os.exit
953
954 if targetlanguage == "de" then
955   typesetcmds = "\\def\\primarylanguage{ngerman}" ..
956     "\\def\\secondarylanguage{english}"
957 elseif targetlanguage == "en" then
958   typesetcmds = "\\def\\primarylanguage{english}" ..
959     "\\def\\secondarylanguage{ngerman}"
```

```
else
  print("Illegal language")
  exit(1)
end
```
⟨/config⟩

7 Stil-Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem Emacs und AUCTEX

Der (X)emacs und das Paket AUCTEX (http://www.gnu.org/software/auctex/) bilden ein mächtiges Werkzeug beim Erstellen von TEX/LATEX-Dateien. Wenn es eine passende Stildatei für ein LATEX-Paket wie das hier vorliegende StruktTEX gibt, wird die Eingabe von Quelltext durch automatisiertes Vervollständigen und viele andere Hilfsmittel sehr erleichtert. Im folgenden wird eine derartige Stildatei bereitgestellt; sie muss nach ihrer Erzeugung noch an eine entsprechende Stelle kopiert werden.

Diese Datei ist zum jetzigen Zeitpunkt noch in der Entwicklungsphase, d. h. man kann mit ihr arbeiten, aber es fehlen noch ein paar Dinge wie das *font locking*, die Anzahl der automatisch eingefügten `\switch` Kommandos sollte nicht fest gleich eins sein, sondern von der Anzahl der eingegebenen Fälle abhängig sein.

⟨*auctex⟩
```
;;; struktex.el -*-coding: utf-8; lexical-binding: t-*-
;;; AUCTeX style for ‘struktex.sty’

;; Copyright (C) 2006 - 2025 Free Software Foundation, Inc.

;; Author: J. Hoffmann <j.hoffmann_(at)_fh-aachen.de>
;; Maintainer: j.hoffmann_(at)_fh-aachen.de
;; Created: 2017/06/06
;; Keywords: tex

;;; Commentary:
;;  This file adds support for ‘struktex.sty’

;;; Code:

(TeX-add-style-hook
 "struktex"
 (lambda ()
   ;; Add declaration to the list of environments which have
   ;; an optional argument for each item.
   (add-to-list 'LaTeX-item-list
                '("declaration" . LaTeX-item-argument))
   (LaTeX-add-environments
    "centernss"
    '("struktogramm" LaTeX-env-struktogramm)
    '("declaration" LaTeX-env-declaration))
   (TeX-add-symbols
    '("PositionNSS" 1)
    '("assert" [ "Height" ] "Assertion")
```

```
    '("assign" [ "Height" ] "Statement")
    "StrukTeX"
    '("case" TeX-mac-case)
    "switch" "Condition"
    "caseend"
    '("declarationtitle" "Title")
    '("description" "Name" "Meaning")
    "emptyset"
    '("exit" [ "Height" ] "What" )
    '("forever" TeX-mac-forever)
    "foreverend"
    '("forallin" TeX-mac-forallin)
    "forallin"
    '("ifthenelse" TeX-mac-ifthenelse)
    "change"
    "ifend"
    '("inparallel" TeX-mac-inparallel)
    '("task" "Description")
    "inparallelend"
    "sProofOn"
    "sProofOff"
    '("until" TeX-mac-until)
    "untilend"
    '("while" TeX-mac-while)
    "whileend"
    '("return" [ "Height" ] "Return value")
    '("sub" [ "Height" ] "Task")
    '("CenterNssFile" TeX-arg-file)
    '("centernssfile" TeX-arg-file))
   (TeX-run-style-hooks
    "pict2e"
    "emlines2"
    "curves"
    "struktxp"
    "struktxf"
    "ifthen")
   ;; Filling
   ;; Fontification
   ))

(defun LaTeX-env-struktogramm (environment)
  "Insert ENVIRONMENT with width, height specifications and
optional title."
  (let ((width (read-string "Width: "))
        (height (read-string "Height: "))
        (title (read-string "Title (optional): ")))
    (LaTeX-insert-environment environment
                              (concat
                               (format "(%s,%s)" width height)
                               (if (not (zerop (length title)))
                                   (format "[%s]" title))))))

(defun LaTeX-env-declaration (environment)
  "Insert ENVIRONMENT with an optional title."
```

```
  (let ((title (read-string "Title (optional): ")))
    (LaTeX-insert-environment environment
                              (if (not (zerop (length title)))
                                  (format "[%s]" title)))))

(defun TeX-mac-case (macro)
  "Insert \\case with all arguments, the needed \\switch(es) and
the final \\caseend. These are optional height and the required
arguments slope, number of cases, condition, and the texts for
the different cases"
  (let ((height (read-string "Height (optional): "))
        (slope (read-string "Slope: "))
        (number (read-string "Number of cases: "))
        (condition (read-string "Condition: "))
        (text (read-string "Case no. 1: "))
        (count 1)
        )
    (setq number-int (string-to-number number))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}{%s}{%s}"
                            slope number condition text)))
    (while (< count number-int)
      (end-of-line)
      (newline-and-indent)
      (newline-and-indent)
      (setq prompt (format "Case no. %d: " (+ 1 count)))
      (insert (format "\\switch{%s}" (read-string prompt)))
      (setq count (1+ count)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\caseend")))

(defun TeX-mac-forallin (macro)
  "Insert \\forallin-block with all arguments.
This is the optional height and the description of the loop"
  (let ((height (read-string "Height (optional): "))
        (loop-description (read-string "Description of loop: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}"
                            loop-description)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\forallinend")))

(defun TeX-mac-forever (macro)
  "Insert \\forever-block with all arguments.
This is only the optional height"
  (let ((height (read-string "Height (optional): ")))
    (insert (if (not (zerop (length height)))
                (format "[%s]" height)))
```

```
1103     (end-of-line)
1104     (newline-and-indent)
1105     (newline-and-indent)
1106     (insert "\\foreverend")))
1107
1108 (defun TeX-mac-ifthenelse (macro)
1109   "Insert \\ifthenelse with all arguments.
1110 These are optional height and the required arguments
1111 left slope, right slope, condition, and the possible
1112 values of the condition"
1113   (let ((height (read-string "Height (optional): "))
1114         (lslope (read-string "Left slope: "))
1115         (rslope (read-string "Right slope: "))
1116         (condition (read-string "Condition: "))
1117         (conditionvl (read-string "Condition value left: "))
1118         (conditionvr (read-string "Condition value right: ")))
1119     (insert (concat (if (not (zerop (length height)))
1120                         (format "[%s]" height))
1121                     (format "{%s}{%s}{%s}{%s}{%s}"
1122                             lslope rslope condition conditionvl
1123                             conditionvr)))
1124     (end-of-line)
1125     (newline-and-indent)
1126     (newline-and-indent)
1127     (insert "\\change")
1128     (end-of-line)
1129     (newline-and-indent)
1130     (newline-and-indent)
1131     (insert "\\ifend")))
1132
1133 (defun TeX-mac-inparallel (macro)
1134   "Insert \\inparallel with all arguments, the needed \\task(s)
1135 and the final \\inparallelend. These are optional height and the
1136 required arguments number of tasks and the descriptions for the
1137 parallel tasks"
1138   (let ((height (read-string "Height (optional): "))
1139         (number (read-string "Number of parallel tasks: "))
1140         (text (read-string "Task no. 1: "))
1141         (count 1)
1142         )
1143     (setq number-int (string-to-number number))
1144     (insert (concat (if (not (zerop (length height)))
1145                         (format "[%s]" height))
1146                     (format "{%s}{%s}" number text)))
1147     (while (< count number-int)
1148       (end-of-line)
1149       (newline-and-indent)
1150       (newline-and-indent)
1151       (setq prompt (format "Task no. %d: " (+ 1 count)))
1152       (insert (format "\\task{%s}" (read-string prompt)))
1153       (setq count (1+ count)))
1154     (end-of-line)
1155     (newline-and-indent)
1156     (newline-and-indent)
```

```
    (insert "\\inparallelend")))

(defun TeX-mac-until (macro)
  "Insert \\until with all arguments.
These are the optional height and the required argument condition"
  (let ((height (read-string "Height (optional): "))
        (condition (read-string "Condition: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}" condition)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\untilend")))

(defun TeX-mac-while (macro)
  "Insert \\while with all arguments.
These are the optional height and the required argument condition"
  (let ((height (read-string "Height (optional): "))
        (condition (read-string "Condition: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{-%s-}" condition)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\whileend")))

(defvar LaTeX-struktex-package-options '("curves" "debug"
                                         "draft" "emlines" "final"
                                         "fixedindent" "nofiller"
                                         "outer" "pict2e" "verification")
  "Package options for the struktex package.")

;;; struktex.el ends here.
```
⟨/auctex⟩

Literatur

[Fut89] Gerald Futschek. *Programmentwicklung und Verifikation*. Springer Verlag, Wien – New York, 1989.

[GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LaTeX-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[GMS04] Frank Mittelbach and Michel Goossens. *The LaTeX-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 2004.

[Knu86] D. E. Knuth. *The TeX-Book*. Addison-Wesley, Reading, Massachusetts, 1986.

[LaT20] The LaTeX3 Project. *The* l3build *package—Checking and building packages*, Januar 2020.

[MDB94] Frank Mittelbach, Denys Duchier and Johannes Braams. *The* DocStrip *program*, Dezember 1994.

[MDB01] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński and Mark Wooding *The* DocStrip *program*, September 2001.

[Mit94] Frank Mittelbach. *The* doc *and* shortvrb *Packages*, Oktober 1994.

[Mit01] Frank Mittelbach. *The* doc *and* shortvrb *Packages*, September 2001.

[Neu96] Marion Neubauer. Feinheiten bei wissenschaftlichen Publikationen – Mikrotypographie-Regeln, Teil I. *Die TeXnische Komödie*, 8(4):23–40, Februar 1996.

[Rah92] Sebastian Rahtz. *The* oz *package*, 1999.