

struktex.sty*

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

gedruckt am 21. September 2010

Zusammenfassung

Dieser Artikel beschreibt den Einsatz und die Implementation der \LaTeX -*package* `struktex.sty` zum Setzen von Struktogrammen nach Nassi-Shneiderman.

Inhaltsverzeichnis

1	Lizenzvereinbarung	1	5	Beispieldatei zum Einbinden in die Dokumentation	22
2	Vorwort	2	6	Verschiedene Beispieldateien	23
3	Hinweise zur Pflege und Installation sowie die Treiberdatei zur Erzeugung dieser Dokumentation	3	6.1	Beispieldatei Nr. 1	23
4	Die Benutzungsschnittstelle	5	6.2	Beispieldatei Nr. 2	24
4.1	Spezielle Zeichen und Textdarstellung	6	6.3	Beispieldatei Nr. 3	25
4.2	Makros zur Darstellung von Variablen, Schlüsselwörtern und anderen programmierspezifischen Details	7	6.4	Beispieldatei Nr. 4	29
4.3	Die Makros zur Erzeugung von Struktogrammen	9	7	Makros zur Erstellung der Dokumentation des <code>struktex.sty</code>	31
			8	Makefile	35
			9	Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)emacs und AUCT\TeX	40

*Diese Datei hat die Versionsnummer v133, wurde zuletzt bearbeitet am 2010/09/21, und die Dokumentation datiert vom 2010/09/21.

1 Lizenzvereinbarung

This package is copyright © 1995 – 2010 by:

Jobst Hoffmann, c/o University of Applied Sciences Aachen
Aachen, Germany
E-Mail: j.hoffmann_(at)_fh-aachen.de

This program can be redistributed and/or modified under the terms of the LaTeX Project Public License, distributed from the CTAN archives as file `macros/latex/base/lppl.txt`; either version 1 of the License, or (at your option) any later version.

2 Vorwort

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit \LaTeX zu zeichnen. Das Makropaket wird im folgenden immer `StrukturTeX` 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 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 (\emptyset) ist in der Darstellung auffälliger als das standardmäßige Zeichen („ \emptyset “) und somit besser für die Darstellung von Struktogrammen geeignet.

Weiterhin ist aus dem `oz.sty` die Idee übernommen, Variablenamen 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 4.1a und 4.1b, für das `\switch` mit der Version 4.2a – erledigt, nicht jedoch für Systeme, die die entsprechenden `\special{...}`-Befehle nicht unterstützen. Erreicht werden kann dies jedoch durch Einsatz entsprechender Makros aus dem `curves.sty`. Seit der Version 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 8.0a realisiert worden.

Weitere Zukunftspläne sind:

1. Ein `\otherwise`-Zweig beim `\switch` (abgeschlossen durch die Version 4.2a).
2. Die Neuimplementation der `declaration`-Umgebung mittels der `list`-Umgebung gemäß [GMS94, Abs. 3.3.4] (abgeschlossen mit der Version 4.5a).

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

3. Die Anpassung an L^AT_EX 2_ε im Sinne eines Packages (abgeschlossen durch die Version 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 4.5a).
6. die vollständige Implementation der Makros `\pVar`, `\pKey`, `\pFonts`, `\pTrue`, `\pFalse` und `\pBoolValue` (erledigt vor Version 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 4.4a.
8. Die Unabhängigkeit der Dokumentation von anderen `.sty`-Dateien wie etwa dem `JHfMakro.sty` (abgeschlossen mit der Version 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 sowie die Treiberdatei zur Erzeugung dieser Dokumentation

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

```

LIESMICH,
README,
struktex.ins,
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.

```
latex struktex.ins
```

die Datei `struktex.ins` formatiert. Dieser Formatierungslauf erzeugt elf weitere Dateien. Dies sind zunächst die drei `.sty`-Dateien `struktex.sty`, `struktxf.sty` und `struktxp.sty`, die beim Einsatz des `struktex.sty` benötigt werden; weiterhin sind es die beiden Dateien `struktex_test_0.nss` und `strukdoc.sty`, die zur Erzeugung der hier

vorliegenden Dokumentation benötigt werden. Dazu kommen drei Testdateien `struktex_test_i.nss`, $i = 1(2)3$, sowie die beiden Dateien `struktex.makemake` und `struktex.mk` (vgl. Abschnitt 8).

Die Dokumentation wird wie üblich durch

```
latex struktex.dtx
latex struktex.dtx
makeindex -s gind.ist struktex.idx
latex struktex.dtx
```

erzeugt.² Das Ergebnis dieses Formatierlaufes ist die Dokumentation in Form einer `.dvi`-Datei, die in gewohnter Weise weiterbearbeitet werden kann. Weitere Informationen zum Arbeiten mit der integrierten Dokumentation findet man in [Mit01] und [MDB01]. Die Dateien `tst_strf.tex`, `tst_strp.tex` schließlich sind Dateien zum Austesten der hier beschriebenen Makros.

Die Installation wird abgeschlossen, indem die Datei `struktex.sty` in ein Verzeichnis verschoben wird, das von $\text{T}_{\text{E}}\text{X}$ 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.³

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.

```
1 (*driver)
2                                     % select the formatting language:
3 \expandafter\ifx\csname primarylanguage\endcsname\relax%
4 \def\primarylanguage{ngerman}%
5 \def\secondarylanguage{english}%
6 \else%
7 \def\secondarylanguage{ngerman}%
8 \fi
9
10 \documentclass[a4paper, \secondarylanguage, % select the language
11         \primarylanguage]{ltxdoc}
12
13 \PassOptionsToPackage{obeyspaces}{url} % must be done before any package is
14                                     % loaded
```

²Die Erzeugung der Dokumentation kann durch den Einsatz einer `make`-Datei vereinfacht werden, vgl. Abschnitt 8

³Wenn die automatische Installation (vgl. Abschnitt 8) vorgenommen wird, erfolgt diese entsprechend.

```

15
16 \usepackage{babel}           % for switching the documentation language
17 \usepackage{struktex}       % the style-file for formatting this
18                             % documentation
19
20 \usepackage[pict2e, % <----- to produce finer results
21                             % visible under xdvi, alternatives are
22                             % curves or emlines2 (visible only under
23                             % ghostscript), leave out if not
24                             % available
25     verification]
26     {struktex}
27 \GetFileInfo{struktex.sty}
28
29 \EnableCrossrefs
30 %\DisableCrossrefs % say \DisableCrossrefs if index is ready
31
32 %\RecordChanges % say \RecordChanges to gather update information
33
34 %\CodelineIndex % say \CodelineIndex to index entry code by line number
35
36 \OnlyDescription % say \OnlyDescription to omit the implementation details
37
38 \MakeShortVerb{\\} % |\foo| acts like \verb+\foo+
39
40 %%%%%%%%%%%%%%%
41 % to avoid underfull ... messages while formatting two/three columns
42 \hbadness=10000 \vbadness=10000
43
44 \typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
45
46 \def\languageNGerman{10} % depends on language.dat, put
47                          % \the\language here
48
49 \begin{document}
50 \makeatletter
51 \@ifundefined{selectlanguageEnglish}{\selectlanguage{english}}
52 \makeatother
53 \DocInput{struktex.dtx}
54 \end{document}
55 </driver>

```

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. `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

em \TeX -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.

2. **verification:**

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

3. **nofiller:**

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

4. **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.

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 Spezielle Zeichen und Textdarstellung

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

`\MathItalics` Mit diesen beiden Makros kann die Darstellung von Variablenamen beeinflusst
`\MathNormal` werden:

```

\MathNormal
\l
NeuerWert = AlterWert + Korrektur
\l

```

und

<i>NeuerWert = AlterWert + Korrektur</i>	$\begin{array}{l} \text{\MathItalics} \\ \text{\[} \\ \text{NeuerWert = AlterWert + Korrektur} \\ \text{\]} \end{array}$
--	--

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

$\begin{array}{l} \text{\pVariable} \\ \text{\pVar} \\ \text{\pKeyword} \\ \text{\pKey} \\ \text{\pComment} \end{array}$	<p>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 <i>package</i> <code>struktexp.sty</code> zusammengefasst worden. Dabei wird ab Version 122 zur Darstellung von Code auf das Paket „<code>url.sty</code>“ 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 <code>url.sty</code>, typischerweise also vor der Anweisung</p>
--	---

```
\usepackage{struktex}
```

die Anweisung

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

setzen.

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

$\begin{array}{l} \text{cEineNormaleVariable} \\ \text{c_eine_normale_Variable} \\ \text{\&iAdresseEinerVariablen} \\ \text{zZeigerAufEineVariable^.sInhalt} \end{array}$	$\begin{array}{l} \text{\obeylines} \\ \text{\renewcommand{\pLanguage}{C}} \\ \text{\pVariable{cEineNormaleVariable}} \\ \text{\pVariable{c_eine_normale_Variable}} \\ \text{\pVariable{\&iAdresseEinerVariablen}} \\ \text{\renewcommand{\pLanguage}{Pascal}} \\ \text{\pVariable{zZeigerAufEineVariable^.sInhalt}} \end{array}$
--	--

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 \langle *Schlüsselwort* \rangle ein Schlüsselwort in einer Programmiersprache, wobei der Unterstrich „_“ und das *hash*-Zeichen „#“ als Teil des Schlüsselwortes erlaubt ist. Damit lässt sich setzen:

$\begin{array}{l} \text{begin} \\ \text{program} \\ \text{\#include} \end{array}$	$\begin{array}{l} \text{\obeylines} \\ \text{\pKeyword{begin}} \\ \text{\renewcommand{\pLanguage}{Pascal}} \\ \text{\pKeyword{program}} \\ \text{\renewcommand{\pLanguage}{C}} \\ \text{\pKeyword{\#include}} \end{array}$
---	--

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 $\text{T}_{\text{E}}\text{X}$ -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` Logische Werte spielen in der Programmierung eine wesentliche Rolle. Mit
`\pFalse` `\pTrue` und `\pFalse` sind entsprechende Werte vorgegeben: WAHR und FALSCH.
`\pFonts` Der Makro `\pFonts` dient der Auswahl von Fonts zur Darstellung von Variablen,
`\pBoolValue` 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` undefiniert werden. Somit liefern die Zeilen

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

das folgende Ergebnis:

nein = **not** *ja*

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

4.3 Die Makros zur Erzeugung von Struktogrammen

`struktogramm` Die Umgebung

```
\sProofOn \begin{struktogramm}(\langle Breite \rangle, \langle Höhe \rangle) [\langle Überschrift \rangle]  
\sProofOff ...  
\PositionNSS \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 Struktogramms mit \TeX den gleichen Wert wie vorher, ist aber innerhalb eines Struktogramms undefiniert 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[\langle Höhe \rangle]{\langle Inhalt \rangle},
```

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}(70,20)[1.\ \text{Versuch}]
```

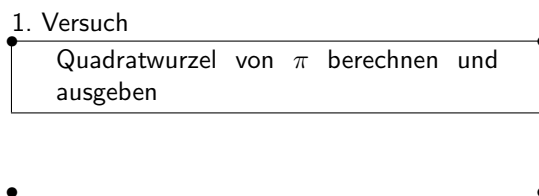
```

\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 21 beschrieben wird.

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



Die Bedeutung des optionalen Argumentes macht das folgende Beispiel deutlich.

Beispiel 2

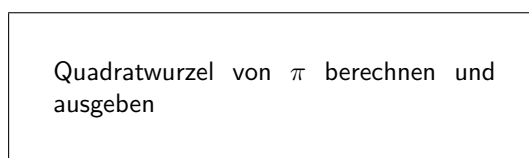
Die Höhe des Kastens wird vorgegeben:

```

\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 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 `StFuTeX` 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.

<pre> Speicherplatz bereitstellen: iVar {eine int -Variable, deren Beschrei- bung hier allein dem Zweck dient, den Makro vorzuführen} </pre>
--

Nun werden Variablen genauer spezifiziert:

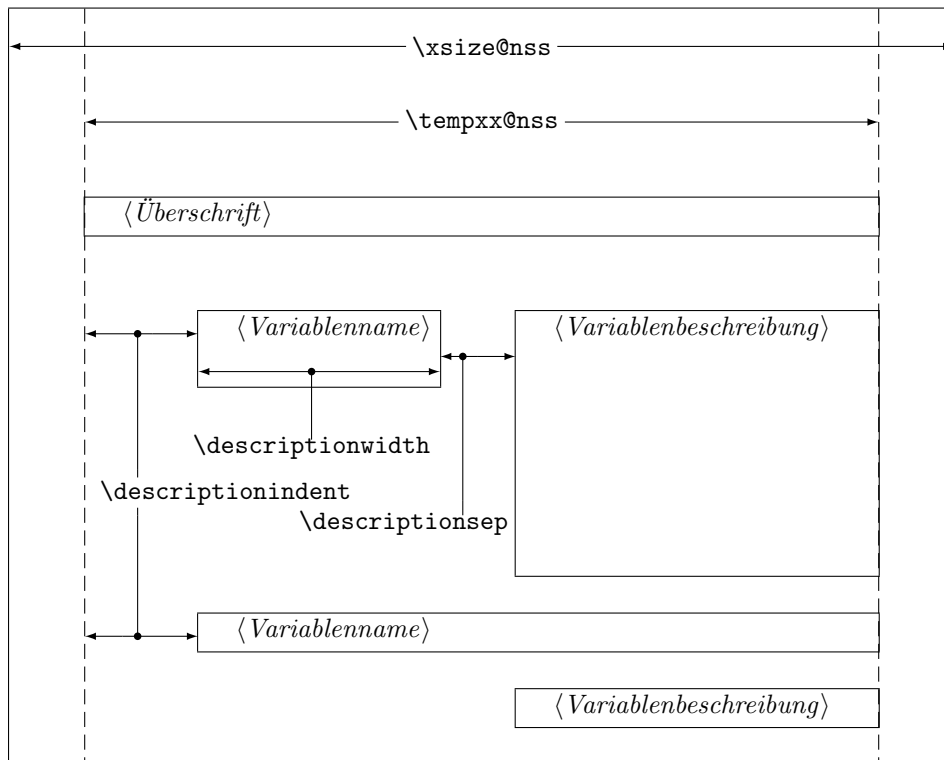


Abbildung 1: Aufbau einer Variablenbeschreibung

```

\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:	
<code>iPar</code>	{ein <code>int</code> -Parameter, dessen Bedeutung hier beschrieben wird}
lokale Variablen:	
<code>iVar</code>	{eine <code>int</code> -Variable, deren Bedeutung hier beschrieben wird}
<code>dVar</code>	{eine <code>double</code> -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:	
<code>iVar_g</code>	{eine <code>int</code> -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 $\text{T}_{\text{E}}\text{X}$ nicht aus.

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

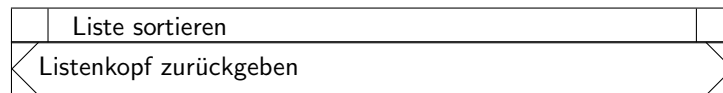
```
\sub[⟨Höhe⟩]{⟨Text⟩}
\return[⟨Höhe⟩]{⟨Text⟩}
```

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 zur"uckgeben}
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



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

```

\while[⟨Breite⟩]{⟨Text⟩}⟨Unterstruktogramm⟩\whileend
\until[⟨Breite⟩]{⟨Text⟩}⟨Unterstruktogramm⟩\untilend
\forever[⟨Breite⟩]⟨Unterstruktogramm⟩\foreverend
\exit[⟨Höhe⟩]{⟨Text⟩}
  
```

$\langle Breite \rangle$ ist die Dicke des Rahmens des Sinnbildes, $\langle Text \rangle$ 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

An Stelle von $\langle Unterstruktogramm \rangle$ können beliebige Befehle von \TeX stehen (mit Ausnahme von `\openstrukt` und `\closestrukt`), die das Struktogramm innerhalb der `\while`-, der `\until`- oder der `\forever`-Schleife bilden.

Um Kompatibilität 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`.

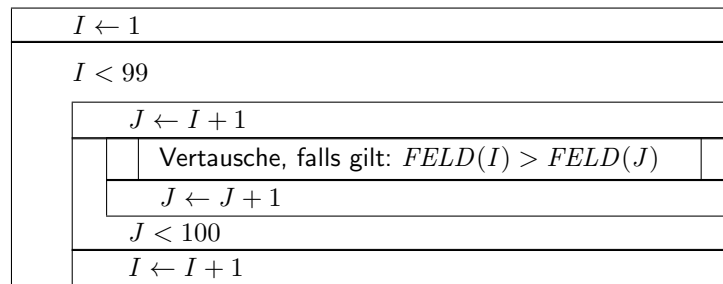
Die folgenden Beispiele zeigen den Einsatz der `\while`- und `\until`-Makros, `\forever` wird weiter unten gezeigt.

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:



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

`\ifthenelse` 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 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` bzw. der `emlines2.sty` eingesetzt wird, ist die Darstellung von Geraden mit beliebiger Steigung möglich.

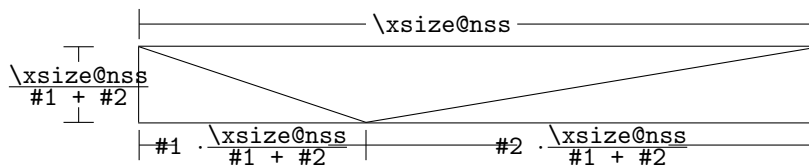
Der If-Then-Else-Befehl sieht so aus:

```

\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 $\frac{\text{\xsize@nss}}{\#1 + \#2}$ 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 5.3 wird der Bedingungs-Text durch geeigneten Umbruch beliebigen Steigungen angepasst.⁴ Die beiden anderen Texte sollten kurz sein (z. B. ja/nein

⁴Diese Erweiterung stammt von Daniel Hagedorn, dem ich hiermit herzlich danken möchte

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` 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 \emptyset ergänzt.⁵ Mit `\ifend` wird das If-Then-Else-Element beendet. Es folgen zwei Beispiele für die Anwendung.

Beispiel 6

```
\begin{struktogramm}(95,32)
  \ifthenelse[12]{1}{2}
    {Flag f"ur 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:

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

Beispiel 7

```
\begin{struktogramm}(90,30)
  \ifthenelse{3}{4}
    {Flag f"ur 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:

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

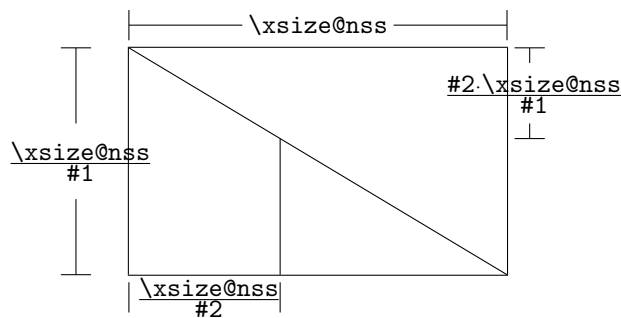
⁵Eventuell ist ein `\strut` hilfreich, um unterschiedliche Höhen von Texten auszugleichen.

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

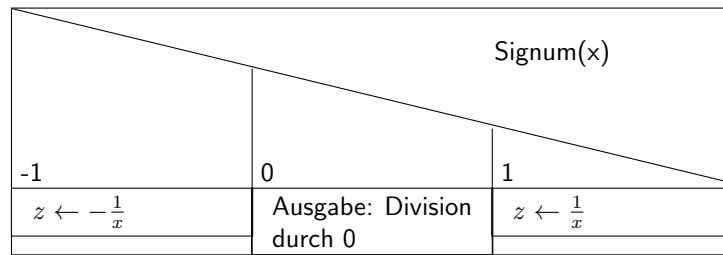


Der zweite Parameter $\langle \text{Anzahl der Fälle} \rangle$ gibt die Anzahl der zu zeichnenden Fälle an; alle Unterstruktogramme der einzelnen Fälle erhalten die gleiche Breite. Der $\langle \text{Text des 1. Falles} \rangle$ muss als Parameter des `\case`-Befehls angegeben werden, alle weiteren Fälle 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.

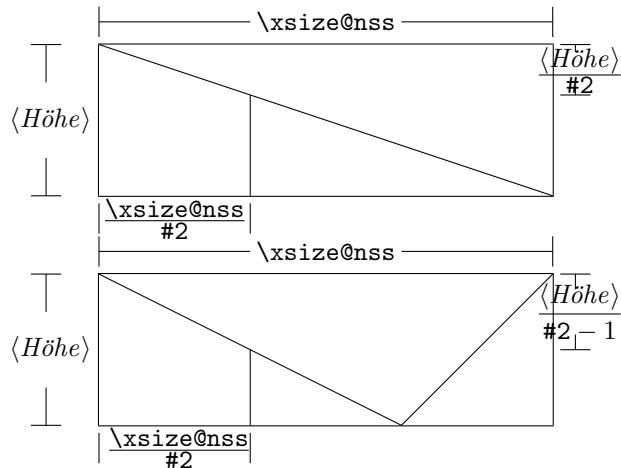
Beispiel 8

```
\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 [$\langle H\ddot{o}he \rangle$] 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 [$\langle H\ddot{o}he \rangle$] führt zu einer anderen Bedeutung von $\langle Winkel \rangle$. 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 9

```

\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 10

```
\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:

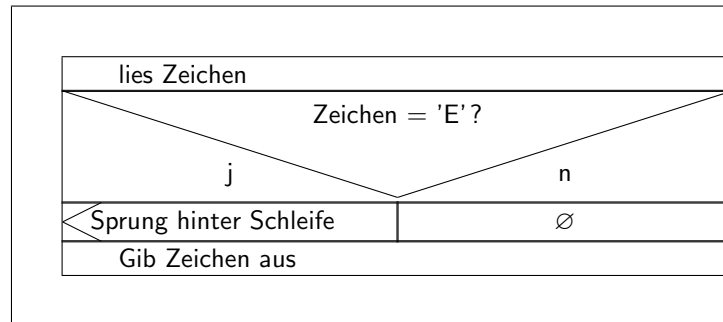
Signum(x)		
-1	1	0
$z \leftarrow -\frac{1}{x}$	$z \leftarrow \frac{1}{x}$	Ausgabe: Division durch 0

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 11

```
\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:

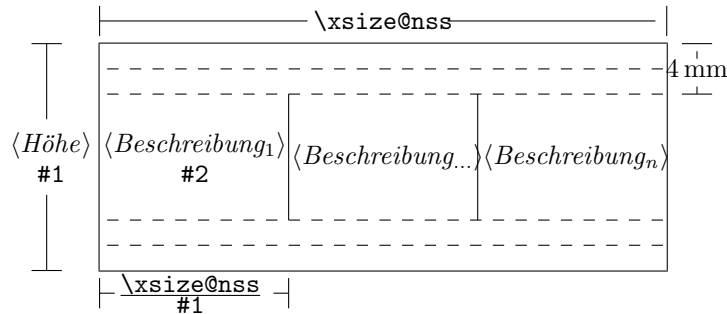


`\inparallel`
`\task`
`\inparallelend`

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`):

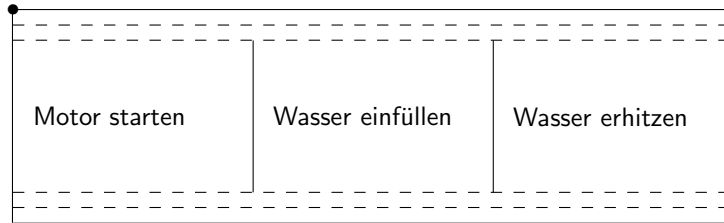


Zu beachten ist, dass die verschiedenen Tasks nicht weiter gegliedert werden dürfen.

Beispiel 12 (Anwendung von `\inparallel`)

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

Diese Anweisungen ergeben das folgende Struktogramm:



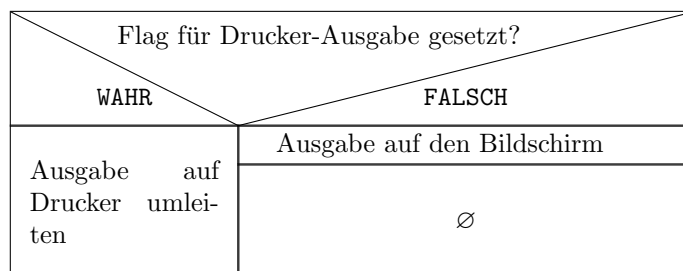
`centernss` Soll ein Struktogramm zentriert dargestellt werden, so wird dazu die Umgebung

```
\begin{centernss}
  <Struktogramm>
\end{centernss}
```

benutzt:

```
\begin{centernss}
\begin{struktogramm}(90,35)
  \ifthenelse{2}{4}
    {Flag f"ur 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:



`\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` definiert ist. Voraussetzung ist, dass die Datei, die die Anweisungen für das Struktogramm enthält, die Dateinamenserweiterung `.nss` hat, der Name der einzubindenden Datei *mus*s demzufolge ohne Erweiterung angegeben werden. Wenn die Datei `struktex-test-0.nss` das in Abschnitt 5, Zeile 2–10 gezeigte Aussehen hat, so führt die Anweisung

```
\centernssfile{struktex-test-0}
```

zu folgendem Aussehen des formatierten Textes:

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

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

```
\openstrukt{\langle width \rangle}{\langle height \rangle}
```

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[\langle Höhe \rangle]{\langle Zusicherung \rangle},
```

Sein Einsatz ergibt sich aus dem folgenden:

```
\begin{struktogramm}(70,20)[Zusicherungen in Struktogrammen]
  \assign{\(a\gets a^2\)}
  \assert{\(a\ge 0\)}
\end{struktogramm}
```

Das dazugehörige Struktogramm sieht folgendermaßen aus:

Zusicherungen in Struktogrammen
$a \leftarrow a^2$
$a \geq 0$

5 Beipieldatei zum Einbinden in die Dokumentation

Die folgenden Zeilen bilden eine Beipieldatei, die bei der Erstellung dieser Dokumentation benötigt wird.

```
56 (*example1)
57 \begin{struktogramm}(95,40) [Text]
58   \case[10]{3}{3}{Signum(x)}{-1}
59     \assign{(z \gets - \frac{1}{x}\)}
60   \switch{0}
61     \assign{Ausgabe: Division durch 0}
62   \switch[r]{1}
63     \assign{(z \gets \frac{1}{x}\)} \caseend
64 \end{struktogramm}
65 (/example1)
```

6 Verschiedene Beipieldateien

6.1 Beipieldatei zum Austesten der Makros des `struktex.sty` ohne die Benutzung optionaler Pakete

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros benutzt werden kann. Der Inhalt ist nur in Englisch vorhanden.

```
66 (*example2)
67 \documentclass[draft]{article}
68 \usepackage{struktex}
69
70 \begin{document}
71
72 \begin{struktogramm}(90,137)
73   \assign%
74   {
75     \begin{declaration}[]
76       \description{(a, b, c\)}{three variables which are to be sorted}
77       \description{(tmp\)}{temporary variable for the circular swap}
78     \end{declaration}
79   }
80   \ifthenelse{1}{2}{(a\le c\)}{j}{n}
81   \change
82   \assign{(tmp\gets a\)}
83   \assign{(a\gets c\)}
84   \assign{(c\gets tmp\)}
85   \ifend
86   \ifthenelse{2}{1}{(a\le b\)}{j}{n}
87   \ifthenelse{1}{1}{(b\le c\)}{j}{n}
88   \change
89   \assign{(tmp\gets c\)}
90   \assign{(c\gets b\)}
91   \assign{(b\gets tmp\)}
92   \ifend
93   \change
94   \assign{(tmp\gets a\)}
```

```

95   \assign{(a\gets b)}
96   \assign{(b\gets tmp)}
97   \ifend
98 \end{struktogramm}
99
100 \end{document}
101 \end{example2}

```

6.2 Beispieldatei zum Austesten der Makros des `struktex.sty` mit dem Paket `pict2e.sty`

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros benutzt werden kann. Der Inhalt ist nur in Englisch vorhanden.

```

102 \begin{example3}
103 \documentclass{article}
104 \usepackage[pict2e, verification]{struktex}
105
106 \begin{document}
107 \def\StruktBoxHeight{7}
108 %\sProofOn{}
109 \begin{struktogramm}(90,137)
110   \assign%
111   {
112     \begin{declaration}[]
113       \description{(a, b, c)}{three variables which are to be sorted}
114       \description{(tmp)}{temporary variable for the circular swap}
115     \end{declaration}
116   }
117   \assert[\StruktBoxHeight]{\sTrue}
118   \ifthenelse[\StruktBoxHeight]{1}{2}{(a\le c)}{j}{n}
119     \assert[\StruktBoxHeight]{(a\le c)}
120   \change
121     \assert[\StruktBoxHeight]{(a>c)}
122     \assign[\StruktBoxHeight]{(tmp\gets a)}
123     \assign[\StruktBoxHeight]{(a\gets c)}
124     \assign[\StruktBoxHeight]{(c\gets tmp)}
125     \assert[\StruktBoxHeight]{(a<c)}
126   \ifend
127   \assert[\StruktBoxHeight]{(a\le c)}
128   \ifthenelse[\StruktBoxHeight]{2}{1}{(a\le b)}{j}{n}
129     \assert[\StruktBoxHeight]{(a\le b \wedge a\le c)}
130     \ifthenelse[\StruktBoxHeight]{1}{1}{(b\le c)}{j}{n}
131       \assert[\StruktBoxHeight]{(a\le b \le c)}
132     \change
133       \assert[\StruktBoxHeight]{(a \le c < b)}
134       \assign[\StruktBoxHeight]{(tmp\gets c)}
135       \assign[\StruktBoxHeight]{(c\gets b)}
136       \assign[\StruktBoxHeight]{(b\gets tmp)}
137       \assert[\StruktBoxHeight]{(a\le b < c)}
138     \ifend
139   \change
140     \assert[\StruktBoxHeight]{(b < a\le c)}
141     \assign[\StruktBoxHeight]{(tmp\gets a)}

```

```

142     \assign[\StruktBoxHeight]{\(\a\gets b\)}
143     \assign[\StruktBoxHeight]{\(\b\gets tmp\)}
144     \assert[\StruktBoxHeight]{\(\a<b\le c\)}
145   \ifend
146   \assert[\StruktBoxHeight]{\(\a\le b \le c\)}
147 \end{struktoqramm}
148
149 \end{document}
150 </example3>

```

6.3 Beispieldatei zum Austesten der Makros des `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.

```

151 (*example4)
152 \documentclass[english]{article}
153
154 \usepackage{babel}
155 \usepackage{struktex}
156
157 \nofiles
158
159 \begin{document}
160
161 \pLanguage{Pascal}
162 \section*{Default values (Pascal):}
163
164 {\obeylines
165 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
166 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
167 in math mode: \(\pVar{a}+\pVar{iV_g}\)
168 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
169 }
170
171 \paragraph{After changing the boolean values with}
172 \verb-\pBoolValue{yes}{no}-:
173
174 {\obeylines
175 \pBoolValue{yes}{no}
176 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
177 }
178
179 \paragraph{after changing the fonts with}
180 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
181
182 {\obeylines
183 \pFonts{\itshape}{\sffamily\bfseries}{}
184 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
185 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
186 in math mode: \(\pVar{a}+\pVar{iV_g}\)
187 boolean values: \sTrue, \sFalse, \pTrue, \pFalse

```

```

188 }
189
190 \paragraph{after changing the fonts with}
191 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
192
193 {\obeylines
194 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
195 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
196 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
197 in math mode: \(\pVar{a}+\pVar{iV_g}\)
198 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
199 }
200
201 \paragraph{after changing the fonts with}
202 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
203
204 {\obeylines
205 \pFonts{\itshape}{\bfseries\itshape}{}
206 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
207 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
208 in math mode: \(\pVar{a}+\pVar{iV_g}\)
209 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
210
211 \vspace{15pt}
212 Without \textit{italic correction}:
213     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
214 }
215
216 \pLanguage{C}
217 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
218 \section*{Default values (C):}
219
220 {\obeylines
221 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
222 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
223 in math mode: \(\pVar{a}+\pVar{iV_g}\)
224 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
225 }
226
227 \paragraph{After changing the boolean values with}
228 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
229
230 {\obeylines
231 \pBoolValue{\texttt{yes}}{\texttt{no}}
232 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
233 }
234
235 \paragraph{after changing the fonts with}
236 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
237
238 {\obeylines
239 \pFonts{\itshape}{\sffamily\bfseries}{}
240 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
241 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}

```

```

242 in math mode: \(\pVar{a}+\pVar{iV_g}\)
243 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
244 }
245
246 \paragraph{after changing the fonts with}
247 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
248
249 {\obeylines
250 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
251 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
252 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
253 in math mode: \(\pVar{a}+\pVar{iV_g}\)
254 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
255 }
256
257 \paragraph{after changing the fonts with}
258 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
259
260 {\obeylines
261 \pFonts{\itshape}{\bfseries\itshape}{}
262 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
263 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
264 in math mode: \(\pVar{a}+\pVar{iV_g}\)
265 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
266
267 \vspace{15pt}
268 Without \textit{italic correction}:
269     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
270 }
271
272 \pLanguage{Java}
273 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
274 \section*{Default values (Java):}
275
276 {\obeylines
277 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
278 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
279 in math mode: \(\pVar{a}+\pVar{iV_g}\)
280 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
281 }
282
283 \paragraph{After changing the boolean values with}
284 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
285
286 {\obeylines
287 \pBoolValue{\texttt{yes}}{\texttt{no}}
288 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
289 }
290
291 \paragraph{after changing the fonts with}
292 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
293
294 {\obeylines
295 \pFonts{\itshape}{\sffamily\bfseries}{}

```

```

296 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
297 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
298 in math mode: \(\pVar{a}+\pVar{iV_g}\)
299 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
300 }
301
302 \paragraph{after changing the fonts with}
303 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
304
305 {\obeylines
306 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
307 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
308 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
309 in math mode: \(\pVar{a}+\pVar{iV_g}\)
310 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
311 }
312
313 \paragraph{after changing the fonts with}
314 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
315
316 {\obeylines
317 \pFonts{\itshape}{\bfseries\itshape}{}
318 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
319 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
320 in math mode: \(\pVar{a}+\pVar{iV_g}\)
321 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
322
323 \vspace{15pt}
324 Without \textit{italic correction}:
325     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
326 }
327
328 \pLanguage{Python}
329 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
330 \section*{Default values (Python):}
331
332 {\obeylines
333 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
334 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
335 in math mode: \(\pVar{a}+\pVar{iV_g}\)
336 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
337 }
338
339 \paragraph{After changing the boolean values with}
340 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
341
342 {\obeylines
343 \pBoolValue{\texttt{yes}}{\texttt{no}}
344 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
345 }
346
347 \paragraph{after changing the fonts with}
348 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
349

```

```

350 {\obeylines
351 \pFonts{\itshape}{\sffamily\bfseries}{}
352 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
353 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
354 in math mode: \(\pVar{a}+\pVar{iV_g}\)
355 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
356 }
357
358 \paragraph{after changing the fonts with}
359 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
360
361 {\obeylines
362 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
363 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
364 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
365 in math mode: \(\pVar{a}+\pVar{iV_g}\)
366 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
367 }
368
369 \paragraph{after changing the fonts with}
370 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
371
372 {\obeylines
373 \pFonts{\itshape}{\bfseries\itshape}{}
374 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
375 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
376 in math mode: \(\pVar{a}+\pVar{iV_g}\)
377 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
378
379 \vspace{15pt}
380 Without \textit{italic correction}:
381 M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
382 }
383
384 \end{document}
385 %%
386 %% End of file 'struktex-test-2.tex'.
387 \example4)

```

6.4 Beispieldatei zum Austesten der Makros des `strukt xp.sty`

Die folgenden Zeilen werden in einem anderen Zusammenhang benutzt, Java-Methoden zu dokumentieren. An dieser Stelle wird ein eigener Weg gewählt, da das sonst übliche Arbeiten mit `lstinl ine` zu Fehlern führt.

```

388 (*example5)
389 \documentclass{article}
390
391 \usepackage{strukt xp, strukt xf}
392
393 \makeatletter
394 \newlength{\fdesc@len}
395 \newcommand{\fdesc@label}[1]%
396 {%

```

```

397 \settowidth{\fdesc@len}{\fdesc@font #1}%
398 \advance\hsize by -2em
399 \ifdim\fdesc@len>\hsize% % term > labelwidth
400 \parbox[b]{\hsize}%
401 {%
402 \fdesc@font #1%
403 }\\%
404 \else% % term < labelwidth
405 \ifdim\fdesc@len>\labelwidth% % term > labelwidth
406 \parbox[b]{\labelwidth}%
407 {%
408 \makebox[0pt][l]{\fdesc@font #1}\\%
409 }%
410 \else% % term < labelwidth
411 {\fdesc@font #1}%
412 \fi\fi%
413 \hfil\relax%
414 }
415 \newenvironment{fdescription}[1][\tt]%
416 {%
417 \def\fdesc@font{#1}
418 \begin{quote}%
419 \begin{list}{}%
420 {%
421 \renewcommand{\makelabel}{\fdesc@label}%
422 \setlength{\labelwidth}{120pt}%
423 \setlength{\leftmargin}{\labelwidth}%
424 \addtolength{\leftmargin}{\labelsep}%
425 }%
426 }%
427 {%
428 \end{list}%
429 \end{quote}%
430 }
431 \makeatother
432
433 \pLanguage{Java}
434
435 \begin{document}
436
437 \begin{fdescription}
438 \item[\index{Methoden>drawImage}(Image img,
439 int dx1,
440 int dy1,
441 int dx2,
442 int dy2,
443 int sx1,
444 int sy1,
445 int sx2,
446 int sy2,
447 ImageObserver observer)=%
448 \Expr{\pKey{public} \pKey{abstract} \pKey{boolean} drawImage(Image img,
449 \pKey{int} dx1,
450 \pKey{int} dy1,

```

```

451             \pKey{int} dx2,
452             \pKey{int} dy2,
453             \pKey{int} sx1,
454             \pKey{int} sy1,
455             \pKey{int} sx2,
456             \pKey{int} sy2,
457             ImageObserver observer)}}%
458     \pExp{public abstract boolean drawImage(Image img, int dx1, int
459         dy1, int dx1, int dy2, int sx1, int sy1, int sx2, int sy2,
460         ImageObserver observer)}}%
461 \ldots
462 \end{fdescription}
463 \end{document}
464 %%
465 %% End of file 'struktex-test-5.tex'.
466 \</example5>

```

7 Makros zur Erstellung der Dokumentation des `struktex.sty`

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.

```

467 (*strukdoc)
468 \RequirePackage{ifpdf}
469 \newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
470 \ifpdf \RequirePackage[colorlinks]{hyperref}\else
471     \def\href#1{\texttt}\fi
472 \ifcolor \RequirePackage{color}\fi
473 \RequirePackage{nameref}
474 \RequirePackage{url}
475 \renewcommand\ref{\protect\T@ref}
476 \renewcommand\pageref{\protect\T@pageref}
477 \@ifundefined{zB}{}{\endinput}
478 \providecommand\pparg[2]{%
479     {\ttfamily}\meta{#1},\meta{#2}{\ttfamily}}
480 \providecommand\envb[1]{%
481     {\ttfamily}\char'\begin\char'\{#1\char'\}}
482 \providecommand\enve[1]{%
483     {\ttfamily}\char'\end\char'\{#1\char'\}}
484 \newcommand{zBspace}{z.,B.}
485 \let\zB=zBspace
486 \newcommand{dhSPACE}{d.,h.}

```

```

487 \let\dh=\dhspace
488 \let\foreign=\textit
489 \newcommand\Abb[1]{Abbildung~\ref{#1}}
490 \def\newexample#1{%
491   \@ifnextchar[{\@oexmpl{#1}}{\@nexmpl{#1}}
492 \def\@nexmpl#1#2{%
493   \@ifnextchar[{\@xnexmpl{#1}{#2}}{\@ynexmpl{#1}{#2}}
494 \def\@xnexmpl#1#2[#3]{%
495   \expandafter\@ifdefinable\csname #1\endcsname
496     {\@definecounter{#1}\@newctr{#1}[#3]}%
497     \expandafter\xdef\csname the#1\endcsname{%
498       \expandafter\noexpand\csname the#3\endcsname \@exmplcountersep
499         \@exmplcounter{#1}}%
500     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
501     \global\@namedef{end#1}{\@endexample}}
502 \def\@ynexmpl#1#2{%
503   \expandafter\@ifdefinable\csname #1\endcsname
504     {\@definecounter{#1}}%
505     \expandafter\xdef\csname the#1\endcsname{\@exmplcounter{#1}}%
506     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
507     \global\@namedef{end#1}{\@endexample}}
508 \def\@oexmpl#1[#2]#3{%
509   \ifundefined{c@#2}{\@nocounterr{#2}}%
510     {\expandafter\@ifdefinable\csname #1\endcsname
511       {\global\@namedef{the#1}{\@nameuse{the#2}}}%
512       \global\@namedef{#1}{\@exmpl{#2}{#3}}%
513       \global\@namedef{end#1}{\@endexample}}}}
514 \def\@exmpl#1#2{%
515   \refstepcounter{#1}%
516   \@ifnextchar[{\@yexmpl{#1}{#2}}{\@xexmpl{#1}{#2}}
517 \def\@xexmpl#1#2{%
518   \@beginexample{#2}{\csname the#1\endcsname}\ignorespaces}
519 \def\@yexmpl#1#2[#3]{%
520   \@opargbeginexample{#2}{\csname the#1\endcsname}{#3}\ignorespaces}
521 \def\@exmplcounter#1{\noexpand\arabic{#1}}
522 \def\@exmplcountersep{.}
523 \def\@beginexample#1#2{%
524   \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
525   \item[{\bfseries #1\ #2}]\mbox{}\\sf}
526 \def\@opargbeginexample#1#2#3{%
527   \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
528   \item[{\bfseries #1\ #2\ (#3)}]\mbox{}\\sf}
529 \def\@endexample{\endlist}
530
531 \newexample{tExample}{\ifnum\language=\languageNGerman Beispiel\else Example\fi}
532
533 \newwrite\struktex@out
534 \newenvironment{example}%
535   {\begingroup% Lets keep the changes local
536   \@bsphack
537   \immediate\openout \struktex@out \jobname.tmp
538   \let\do\@makeother\dospecials\catcode'\^^M\active
539   \def\verbatim@processline{%
540     \immediate\write\struktex@out{\the\verbatim@line}}%

```

```

541 \verbatim@start}%
542 {\immediate\closeout\struktex@out\@esphack\endgroup%
543 %
544 % And here comes the part of Tobias Oetiker
545 %
546 \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
547 \noindent
548 \makebox[0.45\linewidth][l]{%
549 \begin{minipage}[t]{0.45\linewidth}
550 \vspace*{-2ex}
551 \setlength{\parindent}{0pt}
552 \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
553 \begin{trivlist}
554 \item\input{\jobname.tmp}
555 \end{trivlist}
556 \end{minipage}}%
557 \hfill%
558 \makebox[0.5\linewidth][l]{%
559 \begin{minipage}[t]{0.50\linewidth}
560 \vspace*{-1ex}
561 \verbatiminput{\jobname.tmp}
562 \end{minipage}}
563 \par\addvspace{3ex plus 1ex}\vskip -\parskip
564 }
565
566 \newtoks\verbatim@line
567 \def\verbatim@startline{\verbatim@line{}}
568 \def\verbatim@addtoline#1{%
569 \verbatim@line\expandafter{\the\verbatim@line#1}}
570 \def\verbatim@processline{\the\verbatim@line\par}
571 \def\verbatim@finish{\ifcat$\the\verbatim@line$\else
572 \verbatim@processline\fi}
573
574 \def\verbatimwrite#1{%
575 \@bsphack
576 \immediate\openout \struktex@out #1
577 \let\do\@makeother\dospecials
578 \catcode'\^^M\active \catcode'\^^I=12
579 \def\verbatim@processline{%
580 \immediate\write\struktex@out
581 {\the\verbatim@line}}%
582 \verbatim@start}
583 \def\endverbatimwrite{%
584 \immediate\closeout\struktex@out
585 \@esphack}
586
587 \@ifundefined{vrb@catcodes}%
588 {\def\vrb@catcodes{%
589 \catcode'\!12\catcode'\[12\catcode'\]12}}{}
590 \begingroup
591 \vrb@catcodes
592 \lccode'\!='\ \lccode'\[='\ \lccode'\]='\
593 \catcode'\~=\active \lccode'\~='\^^M
594 \lccode'\C='\C

```

```

595 \lowercase{\endgroup
596   \def\verbatim@start#1{%
597     \verbatim@startline
598     \if\noexpand#1\noexpand~%
599     \let\next\verbatim@
600     \else \def\next{\verbatim@#1}\fi
601     \next}%
602   \def\verbatim@#1~{\verbatim@@#1!end\@nil}%
603   \def\verbatim@@#1!end{%
604     \verbatim@addtoline{#1}%
605     \futurelet\next\verbatim@@@}%
606   \def\verbatim@@@#1\@nil{%
607     \ifx\next\@nil
608       \verbatim@processline
609       \verbatim@startline
610       \let\next\verbatim@
611     \else
612       \def\@tempa##1!end\@nil{##1}%
613       \@temptokena{!end}%
614       \def\next{\expandafter\verbatim@test\@tempa#1\@nil~}%
615       \fi \next}%
616   \def\verbatim@test#1{%
617     \let\next\verbatim@test
618     \if\noexpand#1\noexpand~%
619       \expandafter\verbatim@addtoline
620         \expandafter{\the\@temptokena}%
621       \verbatim@processline
622       \verbatim@startline
623       \let\next\verbatim@
624     \else \if\noexpand#1
625       \@temptokena\expandafter{\the\@temptokena#1}%
626     \else \if\noexpand#1\noexpand[%
627       \let\@tempc\@empty
628       \let\next\verbatim@testend
629     \else
630       \expandafter\verbatim@addtoline
631         \expandafter{\the\@temptokena}%
632       \def\next{\verbatim@#1}%
633     \fi\fi\fi
634     \next}%
635   \def\verbatim@testend#1{%
636     \if\noexpand#1\noexpand~%
637       \expandafter\verbatim@addtoline
638         \expandafter{\the\@temptokena[]}%
639       \expandafter\verbatim@addtoline
640         \expandafter{\@tempc}%
641       \verbatim@processline
642       \verbatim@startline
643       \let\next\verbatim@
644     \else\if\noexpand#1\noexpand[%
645       \let\next\verbatim@@testend
646     \else\if\noexpand#1\noexpand!%
647       \expandafter\verbatim@addtoline
648         \expandafter{\the\@temptokena[]}%

```

```

649         \expandafter\verbatim@addtoline
650         \expandafter{\@tempc}%
651         \def\next{\verbatim@!}%
652         \else \expandafter\def\expandafter\@tempc\expandafter
653         {\@tempc#1}\fi\fi\fi
654         \next}%
655     \def\verbatim@ttestend{%
656         \ifx\@tempc\@currenvir
657         \verbatim@finish
658         \edef\next{\noexpand\end{\@currenvir}}%
659         \noexpand\verbatim@rescan{\@currenvir}}%
660     \else
661         \expandafter\verbatim@addtoline
662         \expandafter{\the\@temptokena}%
663         \expandafter\verbatim@addtoline
664         \expandafter{\@tempc}}%
665     \let\next\verbatim@
666     \fi
667     \next}%
668     \def\verbatim@rescan#1#2~{\if\noexpand~\noexpand#2~\else
669     \@warning{Characters dropped after '\string\end{#1}'}\fi}}
670
671 \newread\verbatim@in@stream
672 \def\verbatim@readfile#1{%
673     \verbatim@startline
674     \openin\verbatim@in@stream #1\relax
675     \ifeof\verbatim@in@stream
676     \typeout{No file #1.}%
677     \else
678     \@addtofilelist{#1}%
679     \ProvidesFile{#1}[(verbatim)]%
680     \expandafter\endlinechar\expandafter\m@ne
681     \expandafter\verbatim@read@file
682     \expandafter\endlinechar\the\endlinechar\relax
683     \closein\verbatim@in@stream
684     \fi
685     \verbatim@finish
686 }
687 \def\verbatim@read@file{%
688     \read\verbatim@in@stream to\next
689     \ifeof\verbatim@in@stream
690     \else
691     \expandafter\verbatim@addtoline\expandafter{\expandafter\check@percent\next}%
692     \verbatim@processline
693     \verbatim@startline
694     \expandafter\verbatim@read@file
695     \fi
696 }
697 \def\verbatiminput{\begingroup\MacroFont
698     \@ifstar{\verbatim@input\relax}%
699     {\verbatim@input{\frenchspacing\@vobeyspaces}}}
700 \def\verbatim@input#1#2{%
701     \IfFileExists {#2}{\@verbatim #1\relax
702     \verbatim@readfile{\@filef@und}\endtrivlist\endgroup\@doendpe}%

```

```
703 {\typeout {No file #2.}\endgroup}}
704 </strukdoc>
```

8 Makefile zur automatisierten Erstellung der Dokumentation und der Tests des `struktex.sty`

Der Umgang mit `.dtx`-Paketen ist wesentlich einfacher, wenn ein Werkzeug für die Automatisierung der wesentlichen Schritte vorliegt. Für Unix/Linux basierte Systeme ist das mit `make` und einem geeigneten `Makefile` einfach zu realisieren. Hier wird der `Makefile` in die Dokumentation integriert, die spezielle Syntax mit Tabulatorzeichen wird durch ein Hilfsprogramm erzeugt, das weiter unten angegeben ist. Auf die Benutzung des `Makefile` wird hier nicht weiter eingegangen, da der erfahrene Benutzer des Werkzeugs diese aus der Datei selbst entnehmen kann.

```
705 <{*makefile}
706 #-----
707 # Purpose: generation of the documentation of the struktex package
708 # Notice: this file can be used only with dmake and the option "-B";
709 #         this option lets dmake interpret the leading spaces as
710 #         distinguishing characters for commands in the make rules.
711 #
712 # Rules:
713 #     - all-de:    generate all the files and the (basic) german
714 #                 documentation
715 #     - all-en:    generate all the files and the (basic) english
716 #                 documentation
717 #     - test:      format the examples
718 #     - history:   generate the documentation with revision
719 #                 history
720 #     - develop-de: generate the german documentation with revision
721 #                 history and source code
722 #     - develop-en: generate the english documentation with
723 #                 revision history and source code
724 #     - realclean
725 #     - clean
726 #     - clean-example
727 #
728 # Author:  Jobst Hoffmann, Fachhochschule Aachen, Standort Juelich
729 # Date:    2006/08/23
730 #-----
731
732 # The texmf-directory, where to install new stuff (see texmf.cnf)
733 # If you don't know what to do, search for directory texmf at /usr.
734 # With teTeX and linux often one of following is used:
735 #INSTALLTEXMF=/usr/TeX/texmf
736 #INSTALLTEXMF=/usr/local/TeX/texmf
737 #INSTALLTEXMF=/usr/share/texmf
738 #INSTALLTEXMF=/usr/local/share/texmf
739 # user tree:
740 #INSTALLTEXMF=$(HOME)/texmf
741 # Try to use user's tree known by kpsewhich:
742 #INSTALLTEXMF='kpsewhich --expand-var '$$TEXMFHOME''
743 # Try to use the local tree known by kpsewhich:
```

```

744 #INSTALLTEXMF='kpsewhich --expand-var '$$TEXMFLOCAL''
745 # But you may set INSTALLTEXMF to every directory you want.
746 # Use following, if you only want to test the installation:
747 #INSTALLTEXMF=/tmp/texmf
748
749 # If texhash must run after installation, you can invoke this:
750 TEXHASH=texhash
751
752 ##### Edit following only, if you want to change defaults!
753
754 # The directory, where to install *.cls and *.sty
755 CLSDIR=$(INSTALLTEXMF)/tex/latex/$(PACKAGE)
756
757 # The directory, where to install documentation
758 DOCDIR=$(INSTALLTEXMF)/doc/latex/$(PACKAGE)
759
760 # The directory, where to install the sources
761 SRCDIR=$(INSTALLTEXMF)/source/latex/$(PACKAGE)
762
763 # The directory, where to install demo-files
764 # If we have some, we have to add following 2 lines to install rule:
765 #     $(MKDIR) $(DEMODIR); \
766 #     $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
767 DEMODIR=$(DOCDIR)/demo
768
769 # We need this, because the documentation needs the classes and packages
770 # It's not really a good solution, but it's a working solution.
771 TEXINPUTS := $(PWD):$(TEXINPUTS)
772
773 # To generate the version number of the distribution from the source
774 VERSION_L := latex getversion | grep '^VERSION'
775 VERSION_S := 'latex getversion | grep '^VERSION' | \
776             sed 's+^VERSION \\(.*\) of .*\+\\1+'
777 #####
778 # End of customization section
779 #####
780
781 DVIPS = dvips
782 LATEX = latex
783 PDFLATEX = pdflatex
784
785 # postscript viewer
786 GV = gv
787
788 COMMON_OPTIONS = # \OnlyDescription\CodelineNumbered\PageIndex
789 HISTORY_OPTIONS = \RecordChanges
790 DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex
791
792 # The name of the game
793 PACKAGE = struktex
794
795 # strip off the comments from the package
796 $(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins $(PACKAGE).dtx
797 +$(LATEX) $<; \

```

```

798 sh $(PACKAGE).makemake
799
800 all-de: $(PACKAGE).de.pdf
801
802 all-en: $(PACKAGE).en.pdf
803
804 # generate the documentation
805 $(PACKAGE).de.dvi: $(PACKAGE).dtx $(PACKAGE).sty
806 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
807 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
808 +mv <(:.dtx=.dvi) <(:.dtx=.de.dvi)
809
810 $(PACKAGE).de.pdf: $(PACKAGE).dtx $(PACKAGE).sty
811 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
812 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
813 +mv <(:.dtx=.pdf) <(:.dtx=.de.pdf)
814
815 $(PACKAGE).en.dvi: $(PACKAGE).dtx $(PACKAGE).sty
816 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
817 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
818 +mv <(:.dtx=.dvi) <(:.dtx=.en.dvi)
819
820 $(PACKAGE).en.pdf: $(PACKAGE).dtx $(PACKAGE).sty
821 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
822 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
823 +mv <(:.dtx=.pdf) <(:.dtx=.en.pdf)
824
825 # generate the documentation with revision history (only german)
826 history: $(PACKAGE).dtx $(PACKAGE).sty
827 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
828 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
829 +makeindex -s gind.ist $(PACKAGE).idx
830 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
831 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
832
833 # generate the documentation for the developer (revision history always
834 # in german)
835 develop-de: $(PACKAGE).dtx $(PACKAGE).sty
836 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
837 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
838 +makeindex -s gind.ist $(PACKAGE).idx
839 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
840 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
841 ifneq (,$(findstring pdf,$(LATEX)))
842 +mv <(:.dtx=.pdf) <(:.dtx=.de.pdf)
843 else
844 +mv <(:.dtx=.dvi) <(:.dtx=.de.dvi)
845 endif
846
847 develop-en: $(PACKAGE).dtx $(PACKAGE).sty
848 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{englis
849 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{englis
850 +makeindex -s gind.ist $(PACKAGE).idx
851 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo

```

```

852 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{english}
853 ifneq (,$(findstring pdf,$(LATEX)))
854 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
855 else
856 +mv $(<:.dtx=.dvi) $(<:.dtx=.en.dvi)
857 endif
858
859 # format the example/test files
860 test:
861 for i in `seq 1 3`; do \
862     f=$(PACKAGE)-test-$$i; \
863     echo file: $$f; \
864     $(LATEX) $$f; \
865     $(DVIPS) -o $$f.ps $$f.dvi; \
866     $(GV) $$f.ps \&; \
867 done
868
869 install: $(PACKAGE).dtx $(PACKAGE).dvi
870 [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
871 [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
872 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
873 cp $(PACKAGE).sty      $(CLSDIR)
874 cp $(PACKAGE).dvi     $(DOCDIR)
875 cp $(PACKAGE).ins     $(SRCDIR)
876 cp $(PACKAGE).dtx     $(SRCDIR)
877 cp $(PACKAGE)-test-*.tex $(SRCDIR)
878 cp LIESMICH           $(SRCDIR)
879 cp README             $(SRCDIR)
880 cp THIS-IS-VERSION-$(VERSION) $(SRCDIR)
881
882 uninstall:
883 rm -f $(CLSDIR)/$(PACKAGE).sty
884 rm -fr $(DOCDIR)
885 rm -fr $(SRCDIR)
886
887 dist: $(PACKAGE).de.pdf $(PACKAGE).en.pdf $(PACKAGE).dtx $(PACKAGE).ins \
888 LIESMICH README
889 + rm -f THIS_IS_VERSION_*
890 + $(VERSION_L) > THIS_IS_VERSION_$(VERSION_S)
891 + tar cfvz $(PACKAGE)-$(VERSION_S).tgz $^ THIS_IS_VERSION_*
892 + rm getversion.log
893
894 clean:
895 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
896 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
897 -rm *.mk *.makemake
898
899 realclean: clean
900 -rm -f *.sty *.cls *.ps *.dvi *.pdf
901 -rm -f *test* getversion.* Makefile
902
903 clean-test:
904 rm $(PACKAGE)-test-*. * # this $-sign is needed for font-locking in XEmacs only
905 </makefile>

```

Die folgende Zeile, die nach `latex struktex.ins` als Datei `struktex.makemake` vorliegt, kann mit dem Kommando

```
sh struktex.makemake
```

dazu benutzt werden, die obige Datei in ein Format umzusetzen, das von üblichen `make`-Programmen wie dem GNU `make` verarbeitet werden kann.

```
906 (*setup)
907 sed -e "'echo \"s/~/@/g\" | tr '@' '\011'" struktex.mk > Makefile
908 </setup>
```

Die folgende Datei dient allein dazu, die Version des Paketes zu ermitteln.

```
909 (*getversion)
910 \documentclass{ltxdoc}
911 \nofiles
912 \usepackage{struktex}
913 \GetFileInfo{struktex.sty}
914 \typeout{VERSION \fileversion\space of \filedate}
915 \begin{document}
916 \end{document}
917 </getversion>
```

9 Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)emacs und AUCT_EX

Der (X)emacs und das Paket AUCT_EX (<http://www.gnu.org/software/auctex/>) bilden ein mächtiges Werkzeug beim Erstellen von T_EX/L^AT_EX-Dateien. Wenn es eine passende Stildatei für ein L^AT_EX-Paket wie das hier vorliegende St_ukT_EX 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.

```
918 (*auctex)
919 ;;; struktex.el --- AUCTeX style for 'struktex.sty'
920
921 ;; Copyright (C) 2006 Free Software Foundation, Inc.
922
923 ;; Author: J. Hoffmann <j.hoffmann_(at)_fh-aachen.de>
924 ;; Maintainer: j.hoffmann_(at)_fh-aachen.de
925 ;; Created: 2006/01/17
926 ;; Keywords: tex
927
928 ;;; Commentary:
929 ;; This file adds support for 'struktex.sty'
930
```

```

931 ;; Code:
932 (TeX-add-style-hook
933 "struktex"
934 (lambda ()
935   ;; Add declaration to the list of environments which have an optional
936   ;; argument for each item.
937   (add-to-list 'LaTeX-item-list
938     '("declaration" . LaTeX-item-argument))
939   (LaTeX-add-environments
940     "centernss"
941     '("struktogramm" LaTeX-env-struktogramm)
942     '("declaration" LaTeX-env-declaration))
943   (TeX-add-symbols
944     '("PositionNSS" 1)
945     '("assert" [ "Height" ] "Assertion")
946     '("assign" [ "Height" ] "Statement")
947     "StrukTeX"
948     '("case" TeX-mac-case)
949     "switch" "Condition"
950     "caseend"
951     '("declarationtitle" "Title")
952     '("description" "Name" "Meaning")
953     "emptyset"
954     '("exit" [ "Height" ] "What" )
955     '("forever" TeX-mac-forever)
956     "foreverend"
957     '("ifthenelse" TeX-mac-ifthenelse)
958     "change"
959     "ifend"
960     '("inparallel" TeX-mac-inparallel)
961     '("task" "Description")
962     "inparallelend"
963     "sProofOn"
964     "sProofOff"
965     '("until" TeX-mac-until)
966     "untilend"
967     '("while" TeX-mac-while)
968     "whileend"
969     '("return" [ "Height" ] "Return value")
970     '("sub" [ "Height" ] "Task")
971     '("CenterNssFile" TeX-arg-file)
972     '("centernssfile" TeX-arg-file))
973   (TeX-run-style-hooks
974     "pict2e"
975     "emlines2"
976     "curves"
977     "struktxp"
978     "struktxf"
979     "ifthen")
980   ;; Filling
981   ;; Fontification
982   ))
983
984 (defun LaTeX-env-struktogramm (environment)

```

```

985 "Insert ENVIRONMENT with width, height specifications and optional title."
986 (let ((width (read-string "Width: "))
987       (height (read-string "Height: "))
988       (title (read-string "Title (optional): ")))
989   (LaTeX-insert-environment environment
990     (concat
991       (format "(%s,%s)" width height)
992       (if (not (zerop (length title)))
993         (format "[%s]" title))))))
994
995 (defun LaTeX-env-declaration (environment)
996   "Insert ENVIRONMENT with an optional title."
997   (let ((title (read-string "Title (optional): ")))
998     (LaTeX-insert-environment environment
999       (if (not (zerop (length title)))
1000         (format "[%s]" title))))))
1001
1002 (defun TeX-mac-case (macro)
1003   "Insert \\case with all arguments, the needed \\switch(es) and the final \\caseend.
1004 These are optional height and the required arguments slope, number of cases,
1005 condition, and the texts for the different cases"
1006   (let ((height (read-string "Height (optional): "))
1007         (slope (read-string "Slope: "))
1008         (number (read-string "Number of cases: "))
1009         (condition (read-string "Condition: "))
1010         (text (read-string "Case no. 1: "))
1011         (count 1)
1012         )
1013     (setq number-int (string-to-number number))
1014     (insert (concat (if (not (zerop (length height)))
1015                       (format "[%s]" height))
1016                   (format "{%s}{%s}{%s}{%s}"
1017                           slope number condition text))))
1018   (while (< count number-int)
1019     (end-of-line)
1020     (newline-and-indent)
1021     (newline-and-indent)
1022     (setq prompt (format "Case no. %d: " (+ 1 count)))
1023     (insert (format "\\switch{%s}" (read-string prompt)))
1024     (setq count (1+ count)))
1025   (end-of-line)
1026   (newline-and-indent)
1027   (newline-and-indent)
1028   (insert "\\caseend")))
1029
1030 (defun TeX-mac-forever (macro)
1031   "Insert \\forever-block with all arguments.
1032 This is only the optional height"
1033   (let ((height (read-string "Height (optional): ")))
1034     (insert (if (not (zerop (length height)))
1035               (format "[%s]" height)))
1036     (end-of-line)
1037     (newline-and-indent)
1038     (newline-and-indent)

```

```

1039 (insert "\\foreverend"))
1040
1041 (defun TeX-mac-ifthenelse (macro)
1042 "Insert \\ifthenelse with all arguments.
1043 These are optional height and the required arguments left slope, right slope,
1044 condition, and the possible values of the condition"
1045 (let ((height (read-string "Height (optional): "))
1046 (lslope (read-string "Left slope: "))
1047 (rslope (read-string "Right slope: "))
1048 (condition (read-string "Condition: "))
1049 (conditionvl (read-string "Condition value left: "))
1050 (conditionvr (read-string "Condition value right: ")))
1051 (insert (concat (if (not (zerop (length height)))
1052 (format "[%s]" height)
1053 (format "{%s}{%s}{%s}{%s}"
1054 lslope rslope conditionvl conditionvr)))
1055 (end-of-line)
1056 (newline-and-indent)
1057 (newline-and-indent)
1058 (insert "\\change")
1059 (end-of-line)
1060 (newline-and-indent)
1061 (newline-and-indent)
1062 (insert "\\ifend")))
1063
1064 (defun TeX-mac-inparallel (macro)
1065 "Insert \\inparallel with all arguments, the needed \\task(es) and the final \\inparallelend
1066 These are optional height and the required arguments number of tasks
1067 and the descriptions for the parallel tasks"
1068 (let ((height (read-string "Height (optional): "))
1069 (number (read-string "Number of parallel tasks: "))
1070 (text (read-string "Task no. 1: "))
1071 (count 1)
1072 )
1073 (setq number-int (string-to-number number))
1074 (insert (concat (if (not (zerop (length height)))
1075 (format "[%s]" height)
1076 (format "{%s}{%s}" number text)))
1077 (while (< count number-int)
1078 (end-of-line)
1079 (newline-and-indent)
1080 (newline-and-indent)
1081 (setq prompt (format "Task no. %d: " (+ 1 count)))
1082 (insert (format "\\task{%s}" (read-string prompt)))
1083 (setq count (1+ count)))
1084 (end-of-line)
1085 (newline-and-indent)
1086 (newline-and-indent)
1087 (insert "\\inparallelend")))
1088
1089 (defun TeX-mac-until (macro)
1090 "Insert \\until with all arguments.
1091 These are the optional height and the required argument condition"
1092 (let ((height (read-string "Height (optional): ")

```

```

1093     (condition (read-string "Condition: "))
1094     (insert (concat (if (not (zerop (length height)))
1095                       (format "[%s]" height))
1096                   (format "{%s}" condition)))
1097     (end-of-line)
1098     (newline-and-indent)
1099     (newline-and-indent)
1100     (insert "\\untilend"))
1101
1102 (defun TeX-mac-while (macro)
1103   "Insert \\while with all arguments.
1104 These are the optional height and the required argument condition"
1105   (let ((height (read-string "Height (optional): "))
1106         (condition (read-string "Condition: ")))
1107     (insert (concat (if (not (zerop (length height)))
1108                       (format "[%s]" height))
1109                   (format "{-%s-}" condition)))
1110     (end-of-line)
1111     (newline-and-indent)
1112     (newline-and-indent)
1113     (insert "\\whileend")))
1114
1115 (defvar LaTeX-struktex-package-options '("curves" "draft" "emlines" "final"
1116                                         "pict2e" "anygradient" "verification"
1117                                         "nofiller")
1118   "Package options for the struktex package.")
1119
1120 ;;; struktex.el ends here.
1121 </auctex>

```

Literatur

- [Fut89] Gerald Futschek. *Programmentwicklung und Verifikation*. Springer Verlag, Wien – New York, 1989. 2
- [GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The L^AT_EX-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994. 2
- [GMS04] Frank Mittelbach and Michel Goossens. *The L^AT_EX-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 2004.
- [Knu86] D. E. Knuth. *The T_EX-Book*. Addison-Wesley, Reading, Massachusetts, 1986.
- [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. 4
- [Mit94] Frank Mittelbach. *The doc and shortvrb Packages*, Oktober 1994.

- [Mit01] Frank Mittelbach. *The doc and shortvrb Packages*, September 2001. 4
- [Neu96] Marion Neubauer. Feinheiten bei wissenschaftlichen Publikationen – Mikrotypographie-Regeln, Teil I. *Die T_EXnische Komödie*, 8(4):23–40, Februar 1996. 31
- [Rah92] Sebastian Rahtz. *The oz package*, 1992.