struktex.sty*

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

gedruckt am 18. August 2006

Zusammenfassung

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

Inhaltsverzeichnis

*Diese Datei hat die Versionsnummer v8.2a, wurde zuletzt bearbeitet am 2006/01/20, und die Dokumentation datiert vom 2006/01/20.

1 Vorwort

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit LaTeX zu zeichnen. Das Makropaket wird im folgenden immer StruktTeX 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.

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 ($\varnothing$) 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, 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 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).
3. Die Anpassung an LaTeX 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.

2 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 7).

Die Dokumentation wird wie üblich durch

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

erzeugt.[1] 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 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.

```
\documentclass[a4paper, english, ngerman]{ltxdoc} %swap english and ngerman
                                % for producing an english version of this text

\usepackage{babel}                  % for switching the documentation language
\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]
    {struktex}
\usepackage{url}
\GetFileInfo{struktex.sty}

\EnableCrossrefs
%\DisableCrossrefs   % say \DisableCrossrefs if index is ready

%\RecordChanges      % say \RecordChanges to gather update information

%\CodelineIndex      % say \CodelineIndex to index entry code by line number

\OnlyDescription    % say \OnlyDescription to omit the implementation details
```

[1] Die Erzeugung der Dokumentation kann durch den Einsatz einer `make`-Datei vereinfacht werden, vgl. Abschnitt 7

[2] Wenn die automatische Installation (vgl. Abschnitt 7) vorgenommen wird, erfolgt diese in die Verzeichnisse `.../doc/latex/jhf/struktex/`, `.../tex/latex/jhf/struktex/` und `.../source/latex/jhf/struktex/`.

```
25
26 \MakeShortVerb{\|}   % |\foo| acts like \verb+\foo+
27
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 % to avoid underfull ... messages while formatting two/three columns
30 \hbadness=10000 \vbadness=10000
31
32 \def\languageNGerman{3}
33
34 \begin{document}
35 \makeatletter
36 \@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
37 \makeatother
38 \DocInput{struktex.dtx}
39 \end{document}
```

3 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 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.

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 ∅ 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 St$^{\text{r}}_{\text{i}}$u$_{\text{k}}$TEX erzeugt:

```
\StrukTeX
```

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

3.1 Spezielle Zeichen und Textdarstellung

`\nat` `\integer` `\real` `\complex` `\emptyset` Wegen ihres häufigen Auftretens sind die Mengen der natürlichen, ganzen, reellen und komplexen Zahlen ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ und $\mathbb{C}$) im Mathematik-Modus über die folgenden Makros erreichbar: `\nat`, `\integer`, `\real` und `\complex`. Ebenso ist das mit `\emptyset` erzeugte „$\varnothing$“ 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:

$$NeuerWert = AlterWert + Korrektur$$

```
\MathNormal
\[
NeuerWert = AlterWert + Korrektur
\]
```

und

$$NeuerWert = AlterWert + Korrektur$$

```
\MathItalics
\[
NeuerWert = AlterWert + Korrektur
\]
```

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

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

`\pFonts` `\pBoolValue` 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`.

3.3 Die Makros zur Erzeugung von Struktogrammen

`struktogramm` `\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 zentriert in den Kasten gesetzt; ist er dafür zu lang, so wird ein Paragraph gesetzt.

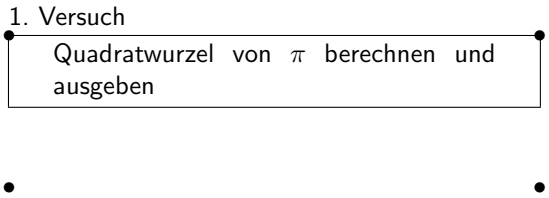
Beispiel 1

Ein einfaches Struktogramm wird mit den folgenden Anweisungen erzeugt:

```
\sProofOn
\begin{struktogramm}(70,20)[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.



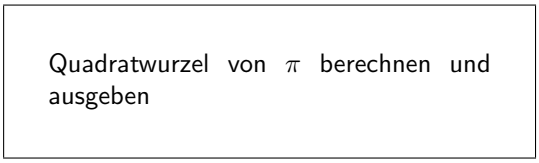
Die Bedeutung des optionalen Argumentes macht das folgende Beispiel deutlich.

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` 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 StruckTEX 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 vorzuf"uhren}
        \end{declaration}
    }
\end{struktogramm}
```

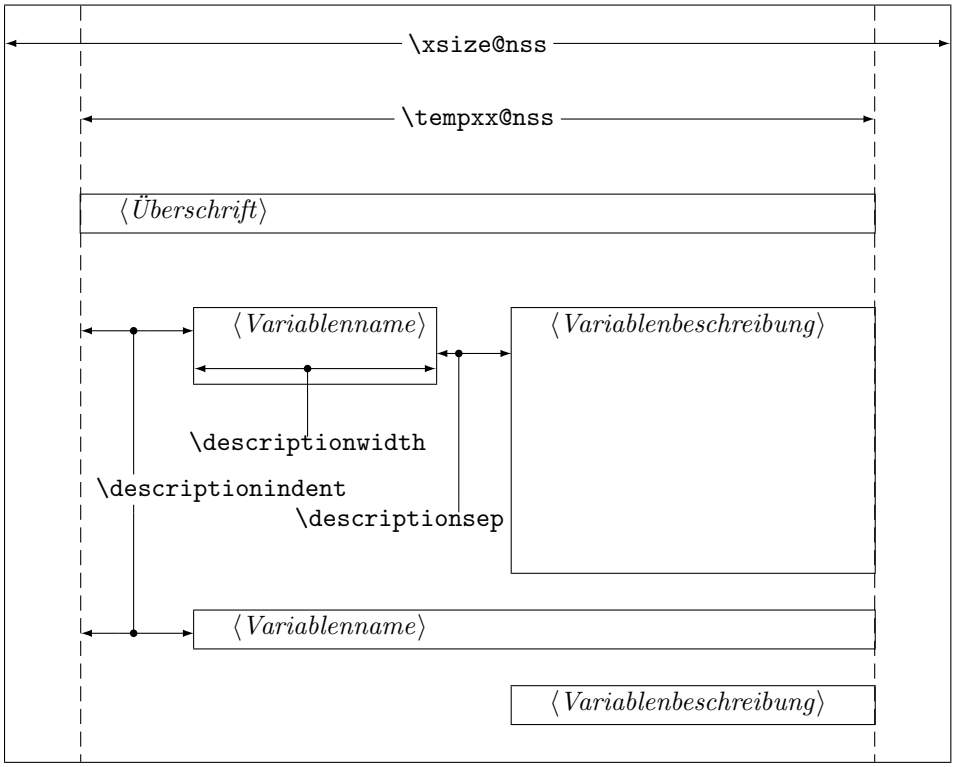


Abbildung 1: Aufbau einer Variablenbeschreibung

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

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)
    {\catcode`\_=12%
        \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*⟩}

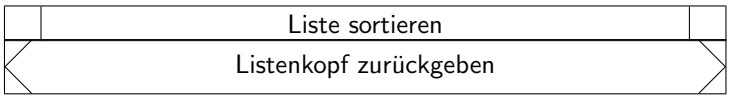
\return[⟨*Höhe*⟩]{⟨*Text*⟩}

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

Diese Anweisungen führen zu folgendem Struktogramm:



\while
\whileend
\until
\untilend
\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

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

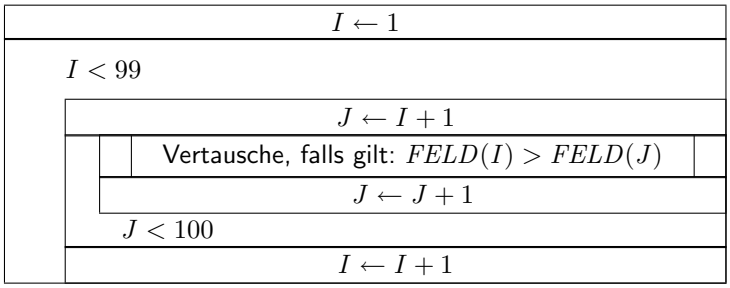
Um Kompatiblität mit der Weiterentwicklung des struktex.sty von J. Dietel zu erreichen, gibt es die Makros `\dfr` und `\dfrend` mit derselben Bedeutung wie `\forever` und `\foreverend`.

Die 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` `\change` `\ifend`

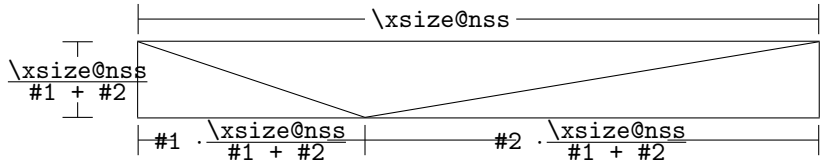
Zur Darstellung von Alternativen stellt StrukTEX die Sinnbilder für einen If-Then-Else-Block und für mehrfache Alternativen eine Case-Konstruktion zur Verfügung. Da in der Picture-Umgebung von LaTeX nur Linien mit bestimmten Steigungen gezeichnet werden können, muss der Benutzer bei beiden Befehlen selbst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

Wenn hingegen der 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 $\frac{\texttt{\textbackslash xsize@nss}}{\texttt{\#1 + \#2}}$ die Höhe des Bedingungsrechteckes.



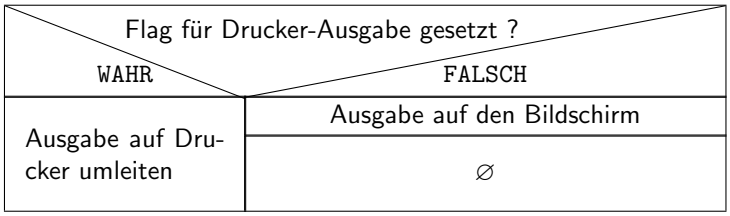
⟨*Bedingung*⟩ wird in das so gebildete obere mittlere Dreieck gesetzt; die Parameter ⟨*Linker Text*⟩ und ⟨*Rechter Text*⟩ werden in das linke bzw. rechte untere Dreieck gesetzt. Der Bedingungs-Text kann in seinem Dreiecks-Feld umgebrochen werden.

Ab Version 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` 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 ∅ ergänzt.[4] 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:



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:

[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.

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

\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*⟩

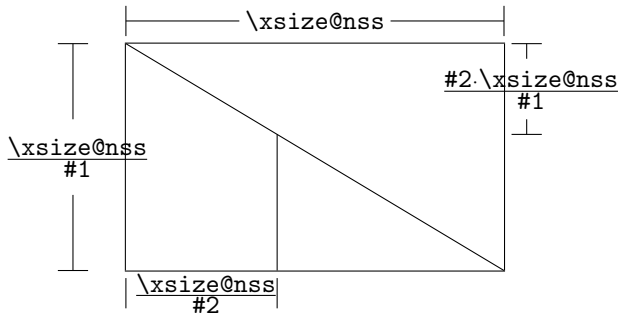
...

\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):



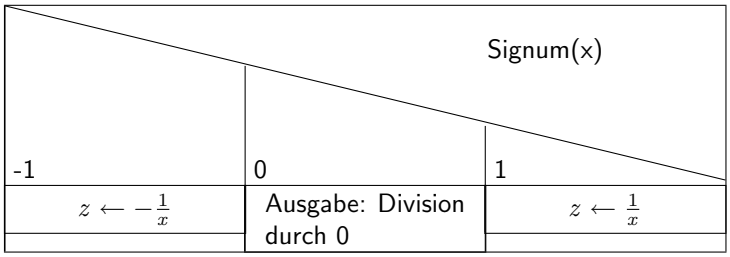
Der zweite Parameter ⟨*Anzahl der Fälle*⟩ gibt die Anzahl der zu zeichnenden Fälle an; alle Unterstruktogramme der einzelnen Fälle erhalten die gleiche Breite. Der ⟨*Text des 1. Falles*⟩ muss als Parameter des \case-Befehls angegeben werden, alle weiteren Fälle werden über den \switch-Befehl eingeleitet. 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 8

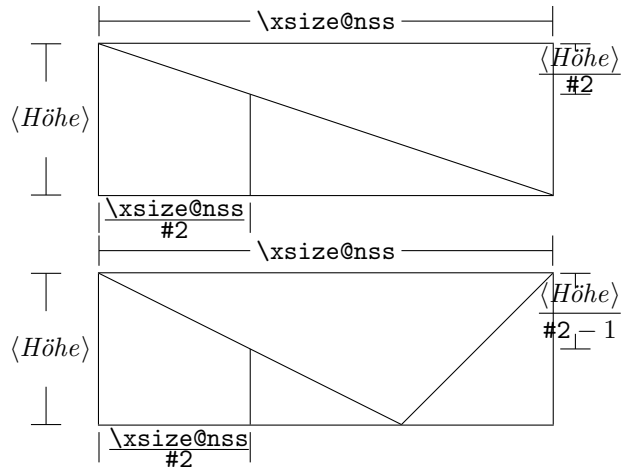
```
\begin{struktogramm}(95,30)
    \case{4}{3}{Signum(x)}{-1}
        \assign{$z \gets - \frac{1}{x}$}
```

```
    \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 `[`⟨*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.

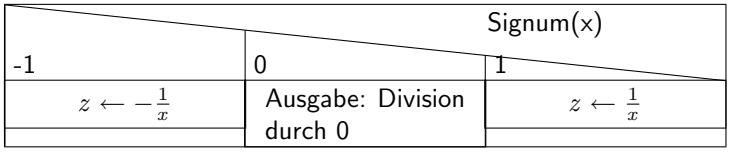


Beispiel 9

```
\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}
```

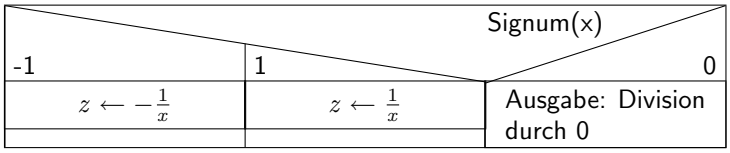
Diese Anweisungen führen zu folgendem Struktogramm:



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:

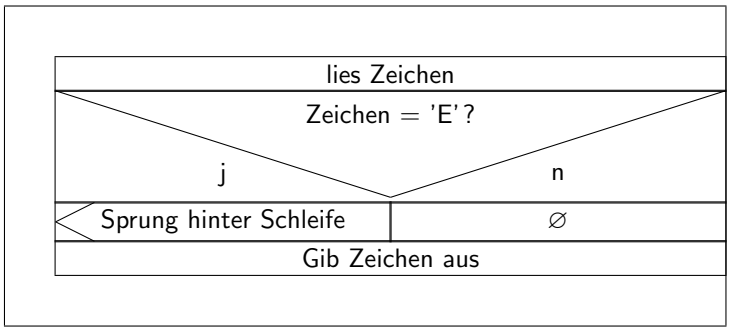


Das folgende Beispiel zeigt, wie mittels einfacher Verzweigung aus einer Endlosschleife gesprungen werden kann. Das Beispiel lässt sich ohne weiteres auf eine mehrfache Verzweigung übertragen.
Beispiel 11

```
\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:



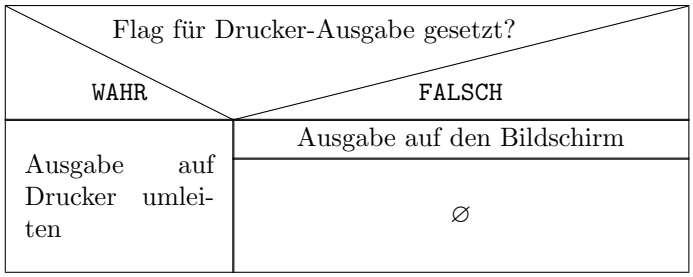
`centernss` Soll ein Struktogramm zentriert dargestellt werden, so wird dazu die Umgebung

```
\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:



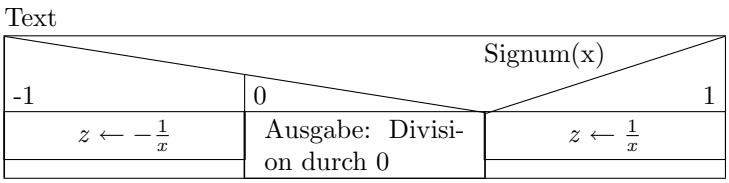
`\CenterNssFile` Häufig gibt es den Fall, dass Struktogramme in eigenen Dateien abgelegt werden, damit sie für sich allein auf Korrektheit getestet werden können oder in anderen Zusammenhängen genutzt werden können. Sollen sie zentriert eingebunden werden, kann nicht mit der folgenden Konstruktion gearbeitet 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` 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-test-0.nss` das in Abschnitt 4, Zeile 2–10 gezeigte Aussehen hat, so führt die Anweisung

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

zu folgendem Aussehen des formatierten Textes:



`\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`:

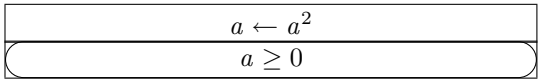
```
\assert[⟨Höhe⟩]{⟨Zusicherung⟩},
```

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



4 Beipieldatei zum Einbinden in die Dokumentation

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

```
\begin{struktogramm}(95,40)[Text]
    \case[10]{3}{3}{Signum(x)}{-1}
         \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
         \assign{Ausgabe: Division durch 0}
    \switch[r]{1}
         \assign{\(z \gets \frac{1}{x}\)} \caseend
\end{struktogramm}
```

5 Verschiedene Beispieldateien

5.1 Beispieldatei 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.

```
\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}
```

5.2 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.

```
\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}
```

5.3 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.

```
\documentclass{article}

\usepackage{struktxp,struktxf}

\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}
%%
%% End of file `struktex-test-2.tex'.
```

5.4 Beispieldatei zum Austesten der Makros des **struktxp.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 `lstinline` zu Fehlern führt.

```
\documentclass{article}

\usepackage{struktxp,struktxf}

\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{Methoden>drawImage(Image img,
                                 int dx1,
                                 int dy1,
                                 int dx2,
                                 int dy2,
                                 int sx1,
                                 int sy1,
                                 int sx2,
                                 int sy2,
                                 ImageObserver observer)=%
      \Expr{\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}
\end{document}
%%
%% End of file `struktex-test-5.tex'.
```

6 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.

```
⟨*strukdoc⟩
\RequirePackage{ifpdf}
\ProvidesPackage{strukdoc}
             [\filedate\space\fileversion\space (Jobst Hoffmann)]
\newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
\ifpdf \RequirePackage[colorlinks]{hyperref}\else
    \def\href#1{\texttt}\fi
\ifcolor \RequirePackage{color}\fi
\RequirePackage{nameref}
\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}}
\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}}
```

7 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.

```
679 #-----------------------------------------------------------------------
680 # Purpose: generation of the documentation of the struktex package
681 # Notice:  this file can be used only with dmake and the option "-B";
682 #          this option lets dmake interpret the leading spaces as
683 #          distinguishing characters for commands in the make rules.
684 #
685 # Rules:
686 #          - all-de:     generate all the files and the (basic) german
687 #                        documentation
688 #          - all-en:     generate all the files and the (basic) english
689 #                        documentation
690 #          - test:       format the examples
691 #          - history:    generate the documentation with revision
692 #                        history
693 #          - develop-de: generate the german documentation with revision
694 #                        history and source code
695 #          - develop-en: generate the english documentation with
696 #                        revision history and source code
697 #          - realclean
698 #          - clean
699 #          - clean-example
700 #
701 # Author:  Jobst Hoffmann, Fachhochschule Aachen, Abt. Juelich
702 # Date:    2003/04/18
703 #-----------------------------------------------------------------------
704
705 # The texmf-directory, where to install new stuff (see texmf.cnf)
706 # If you don't know what to do, search for directory texmf at /usr.
707 # With teTeX and linux often one of following is used:
708 #INSTALLTEXMF=/usr/TeX/texmf
709 #INSTALLTEXMF=/usr/local/TeX/texmf
710 #INSTALLTEXMF=/usr/share/texmf
711 #INSTALLTEXMF=/usr/local/share/texmf
712 # user tree:
713 #INSTALLTEXMF=$(HOME)/texmf
714 # Try to use user's tree known by kpsewhich:
715 INSTALLTEXMF=`kpsewhich --expand-var '$$TEXMFHOME'`
716 # Try to use the local tree known by kpsewhich:
717 #INSTALLTEXMF=`kpsewhich --expand-var '$$TEXMFLOCAL'`
718 # But you may set INSTALLTEXMF to every directory you want.
719 # Use following, if you only want to test the installation:
720 #INSTALLTEXMF=/tmp/texmf
721
```

```
# If texhash must run after installation, you can invoke this:
TEXHASH=texhash

######### Edit following only, if you want to change defaults!

# The directory, where to install *.cls and *.sty
CLSDIR=$(INSTALLTEXMF)/tex/latex/jhf/$(PACKAGE)

# The directory, where to install documentation
DOCDIR=$(INSTALLTEXMF)/doc/latex/jhf/$(PACKAGE)

# The directory, where to install the sources
SRCDIR=$(INSTALLTEXMF)/source/latex/jhf/$(PACKAGE)

# The directory, where to install demo-files
# If we have some, we have to add following 2 lines to install rule:
#     $(MKDIR) $(DEMODIR); \
#     $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
DEMODIR=$(DOCDIR)/demo

# We need this, because the documentation needs the classes and packages
# It's not really a good solution, but it's a working solution.
TEXINPUTS := $(PWD):$(TEXINPUTS)

################################################################################
#   End of customization section
################################################################################

DVIPS = dvips
LATEX = latex
PDFLATEX = pdflatex

# postscript viewer
GV = gv

COMMON_OPTIONS = \OnlyDescription\CodelineNumbered
HISTORY_OPTIONS = \RecordChanges
DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex

PACKAGE = struktex

all-de: $(PACKAGE).de.pdf

all-en: $(PACKAGE).en.pdf

# strip off the comments from the package
$(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).dtx

$(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins
 +$(LATEX) $<

# generate the documentation
$(PACKAGE).dvi: $(PACKAGE).sty

```

```
$(PACKAGE).de.dvi: $(PACKAGE).dtx
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +mv $(<:.dtx=.dvi) $(<:.dtx=.de.dvi)

$(PACKAGE).de.pdf: $(PACKAGE).dtx
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)

$(PACKAGE).en.dvi: $(PACKAGE).dtx
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
 +mv $(<:.dtx=.dvi) $(<:.dtx=.en.dvi)

$(PACKAGE).en.pdf: $(PACKAGE).dtx
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)

# generate the documentation with revision history (only german)
history: $(PACKAGE).dtx
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
 +makeindex -s gind.ist                $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"

# generate the documentation for the developer (revision history always
# in german)
develop-de: $(PACKAGE).dtx
 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
 +makeindex -s gind.ist                $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
ifneq (,$(findstring pdf,$(LATEX)))
 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
else
 +mv $(<:.dtx=.dvi) $(<:.dtx=.de.dvi)
endif

develop-en: $(PACKAGE).dtx
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
 +makeindex -s gind.ist                $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
ifneq (,$(findstring pdf,$(LATEX)))
 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
else
 +mv $(<:.dtx=.dvi) $(<:.dtx=.en.dvi)
endif

```

```
# format the example/test files
test:
 for i in `seq 1 3`; do \
     f=$(PACKAGE)-test-$$i; \
     echo file: $$f; \
     $(LATEX) $$f; \
     $(DVIPS) -o $$f.ps $$f.dvi; \
     $(GV) $$f.ps \&; \
 done

install: $(PACKAGE).dtx $(PACKAGE).dvi
 [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
 [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
 cp $(PACKAGE).sty      $(CLSDIR)
 cp $(PACKAGE).dvi      $(DOCDIR)
 cp $(PACKAGE).ins      $(SRCDIR)
 cp $(PACKAGE).dtx      $(SRCDIR)
 cp $(PACKAGE)-test-*.tex   $(SRCDIR)
 cp LIESMICH        $(SRCDIR)
 cp README          $(SRCDIR)
 cp THIS-IS-VERSION-$(VERSION)  $(SRCDIR)

uninstall:
 rm -f  $(CLSDIR)/$(PACKAGE).sty
 rm -fr $(DOCDIR)
 rm -fr $(SRCDIR)

pack: $(PACKAGE).de.pdf $(PACKAGE).en.pdf  $(PACKAGE).dtx  $(PACKAGE).ins \
LIESMICH README
 + tar cfvz  $(PACKAGE).tgz $^

clean:
 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
 -rm *.mk *.makemake

realclean:  clean
 -rm -f *.sty *.cls *.ps *.dvi *.pdf
 -rm -f *test* getversion.* Makefile

clean-test:
 rm $(PACKAGE)-test-*.* # this $-sign is needed for font-locking in XEmacs only
```

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.

```
sed -e "`echo \"s/^ /@/g\" | tr '@' '\011'`" struktex.mk > Makefile
```

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

```
\documentclass[english]{ltxdoc}
\nofiles
\usepackage{struktex}
\GetFileInfo{struktex.sty}
\typeout{VERSION \fileversion}
\begin{document}
\end{document}
```

8 Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)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 StrukTEX 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 eingefgten `\switch` Kommandos sollte nicht fest gleich eins sein, sondern von der Anzahl der eingegebenen Flle abhngig sein.

```
;;; struktex.el --- AUCTeX style for ‘struktex.sty’

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

;; Author: J. Hoffmann <j.hoffmann@fh-aachen.de>
;; Maintainer: j.hoffmann@fh-aachen.de
;; Created: 2006/01/17
;; 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.
   (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"
    '("ifthenelse" TeX-mac-ifthenelse)
    "ifend"
    "change"
    "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.
These are optional height and the required arguments slope, number of cases,
```

```
condition, and the text of the first case"
  (let ((height (read-string "Height (optional): "))
        (slope (read-string "Slope: "))
        (number (read-string "Number of cases: "))
        (condition (read-string "Condition: "))
        (text (read-string "Text: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}{%s}{%s}"
                            slope number condition text)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\switch{ }")
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\caseend")))

(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)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\foreverend")))

(defun TeX-mac-ifthenelse (macro)
  "Insert \ifthenelse with all arguments.
These are optional height and the required arguments left slope, right slope,
condition, and the possible values of the condition"
  (let ((height (read-string "Height (optional): "))
        (lslope (read-string "Left slope: "))
        (rslope (read-string "Right slope: "))
        (condition (read-string "Condition: "))
        (conditionvl (read-string "Condition value left: "))
        (conditionvr (read-string "Condition value right: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}{%s}{%s}{%s}"
                            lslope rslope condition conditionvl conditionvr)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\change")
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\ifend")))

(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" "draft" "emlines" "final"
                                         "pict2e" "anygradient" "verification"
                                         "nofiller")
  "Package options for the struktex package.")

;;; struktex.el ends here.
```

9 Die Implementation

Der struktex.sty stellt eine einfache Implementation der beschriebenen Makros dar; er sollte prinzipiell mit anderen Makropaketen verträglich sein. Die Gültigkeit von Makros wurde im Regelfall auf einen möglichst kleinen Bereich beschränkt. Um Seiteneffekte mit anderen *packages* so weit wie möglich zu vermeiden, werden alle intern genutzten Makros mit dem Suffix `@nss` versehen.

9.1 Allgemeines

Als erstes wird getestet, ob das Paket struktex.sty noch nicht geladen wurde. Wenn das der Fall ist, wird weitergearbeitet, ansonsten wird das Lesen sofort abgebrochen.

```
⟨*struktex⟩
\@ifundefined{StrukTeX}{}{\endinput}
```

Dies ist ein Paket, das für LaTeX2e entwickelt wurde, es wird daher eine entsprechende Fehlermeldung ausgegeben, wenn ein anderes Format benutzt wird, was nicht zu Fehlern führen sollte; zusätzlich wird das Paket ifthen.sty geladen, falls es noch nicht anderweitig angefordert wurde. Nun wird das Einlesen des Makropakets mit der aktuellen Versionsnummer gemeldet.

```
\RequirePackage{ifthen}
```

StrukTeX benötigt zwei weitere Pakete, die zum Lieferumfang gehören. Diese werden hier geladen.

```
\RequirePackage{struktxf}
\RequirePackage{struktxp}
```

Ab der Version 5.3a kennt StrukTeX eine Option „`emlines`“, die es ermöglicht, Geraden beliebiger Steigung zu zeichnen; später kann auch das Paket curves zum Zeichnen beliebiger Steigungen eingesetzt werden, ab der Version 8.0a wird auch das Paket pict2e unterstützt, das dazugehörige Paket wird hier geladen.

```
\newboolean{curves}
\newboolean{emlines}
\newboolean{pictIIe}
\newboolean{anygradient}
\DeclareOption{curves}%
{%
    \setboolean{anygradient}{true}
    \setboolean{curves}{true}
    \setboolean{emlines}{false}
    \setboolean{pictIIe}{false}
}
\DeclareOption{emlines}%
{%
    \setboolean{anygradient}{true}
    \setboolean{curves}{false}
    \setboolean{emlines}{true}
    \setboolean{pictIIe}{false}
}
\DeclareOption{pict2e}%
{%
    \setboolean{anygradient}{true}
    \setboolean{curves}{false}
```

```
    \setboolean{emlines}{false}
    \setboolean{pictIIe}{true}
}
```

Ab der Version 7.0a kennt StrukTeX eine Option „`verification`“, die ein zusätzliches Element definiert: die `\assert`-Box, die in [Fut89] eingeführt wird.

```
\newboolean{verification}
\DeclareOption{verification}%
{%
    \setboolean{verification}{true}
}
```

Ab der Version 8.2a kennt StrukTeX eine Option „`nofiller`“, die die Darstellung von Leerraum beeinflusst: ohne diese Option wird in Alternativen ein ∅-Zeichen ausgegeben, wenn der Platz ausreichend ist. Dies führt zu einer uneinheitlichen Darstellung im Vergleich mit dem Leerraum, der von einem `\case`-Block erzeugt wird, da hier der Leerraum nicht in entsprechende Weise angezeigt wird – dies wäre nur mit einer aufwendigen Listenverwaltung möglich. Angabe der Option führt dazu, dass das ∅-Zeichen einheitlich weggelassen wird.

```
\newboolean{filler}
\setboolean{filler}{true}
\DeclareOption{nofiller}%
{%
    \setboolean{filler}{false}
}
```

Ab der Version 8.3 kennt StrukTeX die Optionen „`draft`“ und „`final`“, die zwischen dem Entwurf und der endgültigen Version des Struktogramms umschalten. Entwurfsmodus ist `\sProofOn` gesetzt, für die endgültige Version wird `\sProofOff` gesetzt.

```
\let\ifdraft@nss\iffalse
\let\iffinal@nss\iftrue
\DeclareOption{draft}%
{%
  \let\ifdraft@nss\iftrue
  \let\iffinal@nss\iffalse
}
\DeclareOption{final}%
{%
  \let\ifdraft@nss\iffalse
  \let\iffinal@nss\iftrue
}
```

Nun werden andere Pakete geladen und die Optionen verarbeitet.

```
\ProcessOptions
\ifcurves%
    \RequirePackage{curves}
\else\ifemlines
    \RequirePackage{emlines2}
\else\ifpictIIe
    \RequirePackage{pict2e}
\fi\fi\fi
```

`\StrukTeX` Das Logo des Paketes, anhand dessen späteres erneutes Einlesen unterbunden wird:

```
\def\StrukTeX{S\kern-.05emt\kern-.05em\raise.5ex\hbox{r}%
              \kern-.1667emu\kern-.05em\lower.5ex\hbox{k}%
              \kern-.2emT\kern-.1667em\lower.7ex\hbox{E}\kern-.125emX}
```

9.2 Diverse Makros zur Vorbereitung

Zunächst werden die zusätzlich benötigten Fonts bereitgestellt. Da diese Fonts auch in anderem Zusammenhang benötigt werden, wird hierfür ein eigenes Paket definiert.

```
⟨/struktex⟩
⟨*struktxf⟩
\@ifundefined{nat}{}{\endinput}
\DeclareSymbolFont{italics}{\encodingdefault}{\rmdefault}{m}{it}%
\DeclareSymbolFont{AMSb}{U}{msb}{m}{n}
\DeclareSymbolFontAlphabet{\mathbb}{AMSb}
```

`\nat` Die folgenden Makros dienen dem vereinfachten Schreiben von Mengen:

`\integer` `\real` `\complex`

```
\def\nat{{\mathbb N}}
\def\integer{{\mathbb Z}}
\def\real{{\mathbb R}}
\def\complex{{\mathbb C}}
```

`\btt` Um fetten Schreibmaschinensatz erzeugen zu können, gibt es den Font `\btt`, der als Erweiterung der cmtt-Font-Familie definiert ist. Um auf diesen Font auch ohne zusätzliches Einbinden dieses *packages* zugreifen zu können, kann man die entsprechenden Deklarationen in den Dateien `ot1cmtt.fd` bzw. `t1cmtt.fd` durch die folgende ersetzen.[5]

```
\DeclareFontFamily{OT1}{cmbtt}{}
\DeclareFontShape{OT1}{cmtt}{bx}{n}{
   <-8> cmbtt8 <9> cmbtt9 <10-> cmbtt10
}{}

\def\btt%
{%
    \fontencoding{\encodingdefault}\fontfamily{cmbtt}\fontseries{bx}%
    \fontshape{n}\selectfont%
}
```

`\MathItalics` `\MathNormal` Die folgenden Makros dienen dem Umschalten zwischen verschiedenen Formen der *italics* im Mathematik-Modus. Sie wurden aus [Rah92] übernommen. Als Standard wird `\MathItalics` genommen, um eine schönere Darstellung von langen Variablennamen zu erzielen.

```
\def\@setmcodes#1#2#3{{\count0=#1 \count1=#3
    \loop \global\mathcode\count0=\count1 \ifnum \count0<#2
    \advance\count0 by1 \advance\count1 by1 \repeat}}
\def\MathItalics%
{%
    \@setmcodes{‘A}{‘Z}{"7\hexnumber@\symitalics41}
    \@setmcodes{‘a}{‘z}{"7\hexnumber@\symitalics61}
}
\def\MathNormal%
```

[5] Eine Alternative ist die Benutzung des Luxi Mono Fonts.

```
{%
    \@setmcodes{‘A}{‘Z}{"7141}
    \@setmcodes{‘a}{‘z}{"7161}
}
\MathItalics
⟨/struktxf⟩
⟨∗struktex⟩
```

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

Struktogramme enthalten manchmal direkt zu programmierenden Code. Um hier ein einheitliches Aussehen zu erreichen, sind die folgenden Makros definiert worden. Um diese Makros auch in anderem Zusammenhang nutzen zu können, sind sie zu einem eigenen *package* struktxp.sty zusammengefasst worden.

```
⟨/struktex⟩
⟨∗struktxp⟩
\@ifundefined{pVariable}{}{\endinput}
```

\pExpFont@nss
\pKeyFont@nss
\pCommentFont@nss

Die drei Makros \pExpFont@nss, \pKeyFonts@nss und \pCommentFont@nss dienen der Vorbesetzung der Fonts zur Formatierung von Variablen, Schlüsselwörtern und Kommentar. Diese kann mit dem Kommando \pFonts umdefiniert werden.

```
\newcommand{\pExpFont@nss}{\small\sffamily}
\newcommand{\pKeyFont@nss}{\small\sffamily\bfseries}
\newcommand{\pCommentFont@nss}{\small\sffamily\slshape}
```

\pFonts

Der Makro \pFonts dient dem Umsetzen der Vorbesetzungen der Fonts zum Setzen von Variablennamen, Schlüsselwörtern und Kommentar:

```
\newcommand{\pFonts}[3]%
{%
    \def\argi@nss{#1} \def\argii@nss{#2} \def\argiii@nss{#3}
    \ifx\argi@nss\empty\else%
        \renewcommand{\pExpFont@nss}{#1}%
    \fi%
    \ifx\argii@nss\empty\else%
        \renewcommand{\pKeyFont@nss}{#2}%
    \fi%
    \ifx\argiii@nss\empty\else%
        \renewcommand{\pCommentFont@nss}{#3}%
    \fi%
}
```

Zum Setzen von Programmtexten innerhalb von Struktogrammen wird unterschieden zwischen Ausdrücken und (Schlüssel-)Wörtern. Ein Ausdruck (zu verstehen im Sinne eines Ausdrucks einer Programmiersprache) wird in einem festen Font gesetzt, der durch \pExpFont@nss aktuell bestimmt wird. Dieses gilt in gleicher Weise für (Schlüssel-)Wörter, dabei werden aber die Schlüsselwörter, die in einer entsprechenden Liste definiert sind, fett gedruckt. Sowohl Ausdrücke als auch (Schlüssel-)wörter dürfen Sonderzeichen enthalten:

1. der Unterstrich „_“,
2. das Dach „^“,

3. das kaufmännische Und „`&`“ und

4. das Doppelkreuz „`#`“.

`\pExpression` `\pExp` Leerzeichen werden beachtet, so dass ganze Anweisungen geschrieben werden können. Das Argument kann erst nach der Umdefinition der *catcodes* von einem internen Makro ausgewertet werden.

Zur Erleichterung des Schreibens wird mit `\let` ein alternativer Zugriff auf `\pVariable` geschaffen.

```
\newcommand{\pExpression}%
{%
    \bgroup%
```

Zunächst werden die benötigten *catcodes* auf Sonstige – *other* – umgesetzt, zur Erinnerung:

catcode	Bedeutung
1	Gruppenanfang
2	Gruppenende
4	Tabulatorzeichen
8	Index
12	Sonstige

In einer Variablen sind die folgenden Sonderzeichen erlaubt:

1. der Unterstrich „`_`“,

2. das Dach „`^`“ und

3. das kaufmännische Und „`&`“.

```
        \catcode`\_=12 \catcode`\^=12 \catcode`\&=12 %
        \pUseExpFont@nss%
}
\let\pExp=\pExpression
```

`\pVariable` `\pVar` `\pKeyword` `\pKey` Die Implementation von `\pKeyword` entspricht der von `\pVariable`. Die hier neu benutzten Fonts (cmbtt..) werden durch den struktxf.sty bereitgestellt. Ansonsten wird wie bei `\pVariable` verfahren: Zusätzlich werden Makros `\pVar` sowie `\pKeyword` und `\pKey` mit der gleichen Bedeutung definiert. Diese dienen allein der Rückwärtskompatibilität.

```
\newcommand{\pVariable}%
{%
    \bgroup%
        \catcode`\_=13 \let\_\textunderscore
        \catcode`\#=12 \catcode`\^=12 \catcode`\&=12 %
        \pVariabl@%
}
\let\pVar=\pVariable
\let\pKeyword=\pVariable
\let\pKey=\pVariable
```

`\pVariabl@` Die Liste der Schlüsselwörter wird nach den verschiedenen Anwendungsgebieten unterschieden.[6]

[6] Die Python-Erweiterung erfolgte nach einem Hinweis von Ludger Humbert, Universität Dortmund, von dem auch die Liste der Schlüsselwörter stammt, vielen Dank!

```
\newcommand{\pVariabl@}[1]
{%
    \def\arg{#1}%
    \ifx\pLanguage@nss\Cee@nss%
        \CheckForKeyword@nss{auto,break,case,char,const,continue,default,do,double,%
      else,enum,extern,float,for,goto,if,int,long,register,return,%
      short,signed,sizeof,static,struct,switch,typedef,union,unsigned,%
      void,volatile,while}%
    \else\ifx\pLanguage@nss\Java@nss%
        \CheckForKeyword@nss{abstract,boolean,break,byte,case,catch,char,class,const,%
      continue,default,do,double,else,extends,final,finally,float,for,%
      goto,if,implements,import,instanceof,int,interface,long,native,%
      new,null,package,private,protected,public,return,short,static,%
      super,switch,synchronized,this,throw,throws,transient,try,void,%
      volatile,while,true,false}%
    \else\ifx\pLanguage@nss\Pascal@nss%
        \CheckForKeyword@nss{alfa,and,array,begin,boolean,byte,case,char,const,div,do,%
      downto,else,end,false,file,for,function,get,goto,if,in,integer,%
      label,maxint,mod,new,not,of,or,pack,packed,page,program,put,%
      procedure,read,readln,real,record,repeat,reset,rewrite,set,text,%
      then,to,true,type,unpack,until,var,while,with,write,writeln}%
    \else\ifx\pLanguage@nss\Python@nss%
        \CheckForKeyword@nss{and,assert,break,class,continue,def,del,elif,else,%
      except,exec,finally,for,from,global,if,import,in,is,lambda,not,or,%
      pass,print,raise,return,try,while,yield}%
    \else\ifx\pLanguage@nss\LaTeX@nss%
        \CheckForKeyword@nss{center,description,enumerate,figure,itemize,
      list,quote,tabbing,tabular,table,}%
    \else%
        \CheckForKeyword@nss{}%
    \fi\fi\fi\fi\fi%
    \egroup%
}
```

\pLanguage
\Cee@nss
\Java@nss
\Pascal@nss
\LaTeX@nss
\Python@nss

Die Liste der Schlüsselwörter wird nach den verschiedenen Anwendungsgebieten unterschieden.

```
\newcommand{\pLanguage}[1]{\gdef\pLanguage@nss{#1}}
\def\Cee@nss{C}
\def\Java@nss{Java}
\def\Pascal@nss{Pascal}
\def\LaTeX@nss{LaTeX}
\def\Python@nss{Python}
\def\OSII@nss{OS/2}
```

\CheckForKeyword@nss

Die Liste der Schlüsselwörter wird der Reihe nach durchlaufen und mit dem aktuellen Argument \arg verglichen.

```
\def\CheckForKeyword@nss#1%
{%
    \let\exec\pExpFont@nss%
    \def\endList{!}%
    \def\yyI##1,%
    {%
        \def\token{##1}%
        \ifx\token\endList%
```

```
            \ifmmode\else\let\mbox=\relax\fi%
            \mbox{\exec\selectfont\arg}%
        \else%
            \ifx\token\arg%
                \let\exec\pKeyFont@nss%
            \fi%
            \expandafter\yyI%
        \fi%
    }%
    \yyI#1,!,%
}
```

`\pComment` Zur Auszeichnung von Kommentar.

```
\def\pComment%
{%
    \bgroup%
        \pCommentFont@nss%
        \let\next=%
}
```

`\pUseKeyFont@nss` `\pUseExpFont@nss` Damit Unterstriche und ähnliche Sonderzeichen korrekt dargestellt werden können, ist vor der Benutzung eines bestimmten Fonts zu überprüfen, ob die gewünschten Zeichen in dem betreffenden Font überhaupt zur Verfügung stehen.

Es wird stets auf den Textmodus umgeschaltet, was unter Einsatz des amstex-Paketes noch besser durch das Kommando `\text` geschehen könnte. Es können Probleme mit dem Zeichen `^^T`[7] auftreten, in diesem Fall ist das Zeichen durch `^^Y`[8]. zu ersetzen

```
{
    \catcode`\#=6%
    \gdef\pUseKeyFont@nss#1%
    {%
        \ifmmode\else\let\mbox=\relax\fi%
        \mbox{\pKeyFont@nss{#1}}\egroup%
    }
    \gdef\pUseExpFont@nss#1%
    {%
        \ifmmode\else\let\mbox=\relax\fi%
        \mbox{\pExpFont@nss{#1}}\egroup%
    }
}
```

`\pTrue@nss` `\pFalse@nss` Die Vorbesetzungen der logischen Werte `\pTrue` und `\pFalse`:

```
\newcommand{\pTrue@nss}{\texttt{WAHR}}
\newcommand{\pFalse@nss}{\texttt{FALSCH}}
```

`\pTrue` `\pFalse` Um einheitliches Aussehen logischer Werte zu erzielen, werden entsprechende Makros mit den obigen Vorbesetzungen definiert.

```
\newcommand{\pTrue}{\pTrue@nss}
\newcommand{\pFalse}{\pFalse@nss}
```

[7] Dies funktioniert auf einem Linux-System

[8] getestet unter OS/2

\pBoolValue Mit \pBoolValue können die voreingestellten Werte von \pTrue und \pFalse verändert werden; \sBoolValue ist der alternative Aufruf zwecks Kompatibilität mit alten Struktogrammen.

```
\newcommand{\pBoolValue}[2]%
{%
    \renewcommand{\pTrue@nss}{#1}%
    \renewcommand{\pFalse@nss}{#2}%
}
\let\sBoolValue=\pBoolValue
```

\sVar \sKey \sTrue \sFalse Zur Darstellung von Variablen. Hier nur noch zwecks Kompatibilität zu früheren Versionen des struktex.styaufgeführt.

```
\let\sVar=\pVariable
\let\sKey=\pKeyword
\let\sTrue=\pTrue
\let\sFalse=\pFalse
⟨/struktxp⟩
⟨∗struktex⟩
```

9.4 Belegung von Registern

\savelength@nss \savelength@nss dient dazu, die vor dem Struktogramm gültige Längeneinheit zwischenzuspeichern, um nach dem Struktogramm den alten Wert wiederherstellen zu können.

```
\newdimen\savelength@nss
```

\gx@nss \gy@nss \gsize@nss \gx@nss, \gy@nss und \gsize@nss enthalten die *globalen* Werte für die x- und y-Koordinate und die Breite des Struktogramms; zur Vereinfachung der Anwendung wird in allen Fällen mit Zählregistern gearbeitet.

```
\newcount\gx@nss \newcount\gy@nss \newcount\gsize@nss
```

\param@nss \param@nss enthält den Wert, der als optionales Argument für die Breite oder Höhe eines Struktogramms angegeben wird.

```
\countdef\param@nss=199
```

\x@nss \y@nss \xsize@nss \xx@nss \yy@nss \tempx@nss \tempxx@nss \tempxxx@nss \tempy@nss \tempyy@nss \tempyyy@nss \ydepth@nss Es werden einige weitere Variablen benötigt, die alle zentral an dieser Stelle definiert werden: \x@nss, \y@nss und \xsize@nss als lokale Werte für die x- und y-Koordinate und die Breite eines (Unter-)Struktogramms; \ydepth@nss gibt die „Tiefe“ eines Struktogrammelementes an. Die Variablen \xx@nss, \yy@nss, \tempx@nss, \tempxx@nss, \tempxxx@nss, \tempy@nss, \tempyy@nss und \tempyyy@nss haben unterschiedliche temporäre Bedeutungen, im Regelfall deutet ein x im Namen auf horizontalen und ein y auf vertikalen Zusammenhang hin.

```
\countdef\x@nss=220     \countdef\y@nss=221      \countdef\xsize@nss=222
\countdef\ydepth@nss=223
\countdef\xx@nss=224    \countdef\yy@nss=225
\countdef\tempx@nss=226 \countdef\tempxx@nss=227
\countdef\tempy@nss=229 \countdef\tempyy@nss=230
\ifpictIIe%
    \countdef\tempxxx@nss=228  \countdef\tempyyy@nss=231
\fi
```

\TextBox@nss
\TextBoxHt@nss

\TextBox@nss ist die Box, die den Text in einem Struktogrammblock enthält, ihre tatsächliche Höhe, Überlänge und Unterlänge wird in \TextBoxHt@nss abgelegt.

```
\newbox\TextBox@nss \newdimen\TextBoxHt@nss
```

9.5 Die einzelnen Makros

\filler@nss

Beim Setzen eines Struktogrammes entsteht teilweise Leerraum, der mit \filler@nss aufgefüllt wird: „∅“. Eine Alternative wäre „∅“, was mit „\mathchar"023B“ erzeugt wird. Der Leerraum wird allerdings nicht gefüllt, wenn die Option nofiller gesetzt wird. Will man die globale Einstellung lokal umgehen, so kann man einfach die boolesche Variable filler auf true setzen: \fillertrue.

```
\DeclareMathSymbol\varnothing{\mathord}{AMSb}{"3F}
\let\emptyset\varnothing
\def\filler@nss%
{%
    \iffiller\ensuremath{\emptyset}\fi%
}
```

\c@nter@nss

Der folgende Makro setzt verschiedene Parameter, die zum Zentrieren eines Textes gebraucht werden.

```
\def\c@nter@nss%
{%
    \leftskip=0pt plus 2em \rightskip=\leftskip \parfillskip=0pt
    \spaceskip=.333em \xspaceskip=.5em
    \pretolerance=9999 \tolerance=9999
    \hyphenpenalty=9999 \exhyphenpenalty=9999%
}
```

\PositionNSS

Struktogramme werden intern als minipage verwaltet. Das Kommando \PositionNSS ermöglicht die Positionierung, es erwartet die Werte, die die minipage-Umgebung akzeptiert: t, b und keine Parameterangabe. Die Parameter werden nicht überprüft und gelten global ab Aufruf des Kommandos.

```
\def\Position@nss{}%
\newcommand{\PositionNSS}[1]%
{%
    \def\Position@nss{#1}%
}
```

\struktogramm

\struktogramm beginnt ein Struktogramm

```
\def\struktogramm(#1,#2)%
{%
```

Zunächst werden die internen Kommandos aktiviert:

```
    \let\ifthenelse=\ifthenelse@nss%
    \let\ifend=\ifend@nss%
```

Falls die Option draft gesetzt wurde, wird für jedes Struktogramm grundsätzlich auf den Entwurfsmodus umgeschaltet.

```
    \ifdraft@nss\let\struktogramm@nss=\struktogramm@Proof\fi%
```

Um den optionalen Parameter zu behandeln, wird in Abhängigkeit vom nächsten Zeichen ein internes Makro aufgerufen.

```
    \@ifnextchar [{\struktogramm@nss(#1,#2)}{\struktogramm@nss(#1,#2)[]}%
}
```

\endstruktogramm

\endstruktogramm beendet das Struktogramm. Offene Gruppen und Umgebungen werden geschlossen, lokale Definitionen globaler Werte werden rückgängig gemacht.

```
\def\endstruktogramm%
{%
            \endstr@kt%
        \end{picture}%
    \end{minipage}%
    \setlength{\unitlength}{\savelength@nss}%
}
```

\struktogramm@NoProof

\struktogramm@NoProof öffnet ein Struktogramm

```
\def\struktogramm@NoProof(#1,#2)[#3]%
{%
```

Zunächst wird die aktuelle \unitlength zwischengespeichert, da mit einer eigenen Größe gearbeitet wird.

```
    \def\next{#3}
    \setlength{\savelength@nss}{\unitlength}
    \setlength{\unitlength}{0.001mm}
```

Wenn der dritte Parameter nicht \relax ist, wird die Bezeichnung geschrieben und es muss ein bisschen zusätzlicher Platz geschaffen werden.

```
    \begin{minipage}[\Position@nss]{#1mm}
        \ifx\next\empty\else #3\\[\smallskipamount]\fi%
```

Um die Verwaltung des benötigten Raumes möglichst einfach zu machen, wird eine \picture-Umgebung geöffnet, in der eine weitere Gruppe durch den Aufruf von \str@kt geöffnet wird.

```
        \begin{picture}(#1000,#2000)%
            \str@kt{0}{#2000}{#1000}%
}
```

\struktogramm@Proof

\struktogramm@proof öffnet ein Struktogramm im *proof*-Modus. Der Ablauf entspricht im wesentlichen \struktogramm@NoProof.

```
\def\struktogramm@Proof(#1,#2)[#3]%
{%
    \def\next{#3}%
    \setlength{\savelength@nss}{\unitlength}%
    \setlength{\unitlength}{0.001mm}%
    \begin{minipage}[\Position@nss]{#1mm}%
        \ifx\next\empty\else #3\\[\smallskipamount]\fi%
        \begin{picture}(#1000,#2000)%
            \put(0,0){\makebox(0,0)[cc]{$\bullet$}}%
            \put(0,#2000){\makebox(0,0)[cc]{$\bullet$}}%
            \put(#1000,0){\makebox(0,0)[cc]{$\bullet$}}%
            \put(#1000,#2000){\makebox(0,0)[cc]{$\bullet$}}%
            \str@kt{0}{#2000}{#1000}%
}
```

\sProofOn
\sProofOff

Die Schalter zum Umschalten zwischen *proof*- und *non proof*-Modus. Abschließend wird in den *non proof*-Modus umgeschaltet.

```
\def\sProofOn%
{%
```

```
    \let\struktogramm@nss=\struktogramm@Proof%
}
\def\sProofOff%
{%
    \let\struktogramm@nss=\struktogramm@NoProof%
}
\let\struktogramm@nss=\struktogramm@NoProof%
```

Um Kompatibilität mit älteren Versionen von Struktogrammen zu erhalten, sind die Makros `\openstrukt` und `\closestrukt` weiterhin in diesem *package* erhalten. Diese sollten aber nicht weiter benutzt werden.

`\openstrukt` `\openstrukt` beginnt ein Struktogramm

```
\def\openstrukt#1#2%
{%
```

The actual `\unitlength` will be stored intermediately, since it is worked with an own entity.

```
    \setlength{\savelength@nss}{\unitlength}
    \setlength{\unitlength}{0.001mm}
```

Um die Verwaltung des benötigten Raumes möglichst einfach zu machen, wird eine `\picture`-Umgebung geöffnet, in der eine eigene weitere Gruppe durch den Aufruf von `\str@kt` geöffnet wird.

```
    \begin{picture}(#1000,#2000)
        \str@kt{0}{#2000}{#1000}%
}
```

Es ist zu beachten, dass es kein optionales Argument gibt.

`\closestrukt` `\closestrukt` beendet das Struktogramm. Offene Gruppen und Umgebungen werden geschlossen, lokale Definitionen globaler Werte werden rückgängig gemacht.

```
\def\closestrukt%
{%
        \endstr@kt
    \end{picture}
    \setlength{\unitlength}{\savelength@nss}%
}
```

`\getoption` `\getoption` überprüft, ob ein optionales Argument angegeben wurde. Dies geschieht, indem das nächste Zeichen getestet wird.

```
\def\getoption{\@ifnextchar [{\getnum}{\param@nss=0\next}}
```

`\getnum` `\getnum` weist den Parameter dem globalen Zähler `\param@nss` zu und ruft den auf `\next` gesetzten Makro auf.

```
\def\getnum[#1]{\param@nss=#1\next}
```

`\str@kt` `\str@kt` setzt globale Werte: `\gx@nss` als x-Koordinate (`#1`), `\gy@nss` als y-Koordinate (`#2`) und `\gsize@nss` als Breite des aktuellen Struktogramms (`#3`). Gleichzeitig wird eine Gruppe geöffnet, in der mit lokalen Kopien dieser Werte (`\x@nss`, `\y@nss` und `\xsize@nss`) gearbeitet werden kann.

```
\def\str@kt#1#2#3%
{%
    \global\gx@nss=#1\global\gy@nss=#2\global\gsize@nss=#3%
```

```
    \begingroup%
        \x@nss=\gx@nss\y@nss=\gy@nss\xsize@nss=\gsize@nss%
}
```

`\endstr@kt` schließt die oben geöffnete Gruppe, die aktuelle y-Koordinate wird global umgesetzt, um den (weiter unten liegenden) Ansatzpunkt für das nächste Element des Struktogramms zu erhalten.

```
\def\endstr@kt%
{%
        \global\gy@nss=\y@nss%
    \endgroup%
    \ydepth@nss=\gy@nss%
}
```

`\set@nss` `\set@nss` setzt den als ersten Parameter übergebenen Text in eine horizontale Box `\TextBox@nss`. Ist die Breite dieser Box größer als die im ersten Parameter übergebene Breite, wird der Text in eine vertikale Box gesetzt.

```
\def\set@nss#1#2%
{%
    \setbox\TextBox@nss=\hbox{#1}%
    \tempxx@nss=\xsize@nss\advance\tempxx@nss by-#2%
    \ifdim\wd\TextBox@nss>\tempxx@nss\unitlength%
        \setbox\TextBox@nss=\vbox{\hsize=\tempxx@nss\unitlength\noindent#1}%
    \fi%
}
```

`\block@nss` `\block@nss` setzt die Box `\TextBox@nss` entsprechend dem zweiten Parameter. Ist dieser 1, wird eine Box mit Rahmen gesetzt, ist dieser 0, wird eine Box ohne Rahmen gesetzt. Bei allen anderen Werten wird nur die Größe des Struktogramms, gegeben durch `\y@nss`, vergrößert.

```
\def\block@nss#1#2%
{%
    \tempy@nss=#1\TextBoxHt@nss=\ht\TextBox@nss%
    \advance\TextBoxHt@nss by\dp\TextBox@nss%
    \advance\TextBoxHt@nss by 2mm%
    \ifdim\TextBoxHt@nss<\tempy@nss\unitlength%
        \TextBoxHt@nss=\tempy@nss\unitlength%
    \fi%
    \TextBoxHt@nss=0.00536\TextBoxHt@nss%
    \ydepth@nss=\TextBoxHt@nss\advance\y@nss by-\ydepth@nss%
    \ifx#21%
        \put(\x@nss,\y@nss)%
        {%
            \framebox(\xsize@nss,\ydepth@nss){\box\TextBox@nss}%
        }%
    \else%
        \ifx#20%
            \put(\x@nss,\y@nss)%
            {%
                \makebox(\xsize@nss,\ydepth@nss){\box\TextBox@nss}%
            }%
        \else%
```

```
            \advance\y@nss by \ydepth@nss%
        \fi%
    \fi%
}
```

`\declarationtitle` Der Makro `\declarationtitle` legt den Inhalt der optionalen Überschrift bei Deklarationen fest. Er kann global vorgegeben werden, um eine einheitliche Darstellung zu erhalten.

```
\def\declarationtitle{\ifnum\language=\languageNGerman Speicherplatz bereitstellen:
                      \else providing memory space\fi}
```

`\descriptionindent` `\descriptionwidth` `\descriptionwidth@nss` `\descriptionsep` Die folgenden Variablen (*dimen-register*) wurden eingeführt, um das Layout der Variablenbeschreibung frei gestalten zu können.

```
\newlength{\descriptionindent} \setlength{\descriptionindent}{1.5em}
\newlength{\descriptionwidth}  \setlength{\descriptionwidth}{40pt}
\newlength{\descriptionsep}    \setlength{\descriptionsep}{\tabcolsep}
\newlength{\descriptionwidth@nss}
```

`\descriptionlabel@nss` Internes Kommando nach [GMS94, Abs. 3.3.5] zur Definition der Marke bei der Liste der Variablenbeschreibungen

```
\newcommand{\descriptionlabel@nss}[1]%
{%
    \settowidth{\descriptionwidth@nss}{#1}%
    \ifnum\descriptionwidth@nss>\descriptionwidth%  % term > labelwidth
        \parbox[b]{\descriptionwidth}%
        {%
            \makebox[0pt][l]{#1}\\%
        }%
    \else%                                         % term < labelwidth
        #1%
    \fi%
    \hfil\relax%
}
```

`declaration` Die Beschreibung von Variablen basiert nun auf der allgemeinen `list`-Umgebung, die entsprechend modifiziert wurde. Insbesondere wurden alle Abstände auf 0pt gesetzt.

```
\newenvironment{declaration}[1][\declarationtitle]%
{%
    \def\next{#1}%
    \ifx\next\empty\else #1\\\fi
    \let\description=\description@nss
    \tempxx@nss=\xsize@nss\advance\tempxx@nss by -\tempx@nss%
    \begin{minipage}[t]{\tempxx@nss\unitlength}
    \begin{list}{}%
    {%
        \renewcommand{\makelabel}{\descriptionlabel@nss}%
        \setlength{\labelwidth}{\descriptionwidth}%
        \setlength{\itemsep}{0pt} \setlength{\topsep}{0pt}%
        \setlength{\parsep}{0pt} \setlength{\partopsep}{0pt}
        \setlength{\leftmargin}{\descriptionwidth}%
        \addtolength{\leftmargin}{\descriptionsep}%
        \addtolength{\leftmargin}{\descriptionindent}%
```

```
    }%
}%
{%
    \end{list}%
    \end{minipage}
}
```

`\description@nss` Um keine Schwierigkeiten mit der `description`-Umgebung zu haben, wird das Kommando `\description@nss` mit der Erweiterung `@nss` eingeführt. Seine Gültigkeit wird mit einem `\let` auf die `declaration`-Umgebung beschränkt

```
\newcommand{\description@nss}[2]{\item[#1] \{#2\}}
```

`\assign` Einfache Anweisungen werden über `\assign` geschrieben:

```
\def\assign{\let\next=\@assign\getoption}%
\def\@assign#1%
{%
    \tempx@nss=\xsize@nss\divide\tempx@nss by 6%
    \set@nss{#1}{\tempx@nss}\tempy@nss=\number\param@nss000%
    \block@nss{\tempy@nss}{1}%
}
```

`\sub` Der Aufruf eines Unterprogrammes wird mit `\sub` gekennzeichnet:

```
\def\sub{\let\next=\@sub\getoption}
\def\@sub#1%
{%
    \@assign{#1}\tempx@nss=\xsize@nss\divide\tempx@nss by 20%
    \tempxx@nss=\x@nss\advance\tempxx@nss by \tempx@nss%
    \put(\tempxx@nss,\y@nss){\line(0,1){\ydepth@nss}}%
    \tempxx@nss=\x@nss\advance\tempxx@nss by \xsize@nss%
    \advance\tempxx@nss by -\tempx@nss%
    \put(\tempxx@nss,\y@nss){\line(0,1){\ydepth@nss}}%
}
```

`\exit` Verlassen einer Schleife an einer bestimmten Stelle.

```
\def\exit{\let\next=\@exit\getoption}
\def\@exit#1%
{%
    \tempx@nss=\xsize@nss\divide\tempx@nss by 6%
    \set@nss{#1}{\tempx@nss}%
    \tempy@nss=\number\param@nss000%
    \block@nss{\tempy@nss}{1}%
    \divide\ydepth@nss by 2%
    \advance\y@nss by \ydepth@nss%
    \ifnum\ydepth@nss<3500%
```

Eigene Gruppe bilden, um Änderung an `\ydepth@nss` lokal zu halten

```
    {%
       \multiply\ydepth@nss by 2%
       \put(\x@nss,\y@nss){\line(2,1){\ydepth@nss}}%
       \put(\x@nss,\y@nss){\line(2,-1){\ydepth@nss}}%
    }%
    \else%
       \put(\x@nss,\y@nss){\line(1,1){\ydepth@nss}}%
       \put(\x@nss,\y@nss){\line(1,-1){\ydepth@nss}}%
```

```
    \fi%
    \advance\y@nss by-\ydepth@nss%
}
```

\return Die Rückkkehr aus einem Unterprogramm wird mit \return gekennzeichnet:

```
\def\return{\let\next=\@return\getoption}
\def\@return#1%
{%
    \tempy@nss=\number\param@nss000%
    \ifnum\tempy@nss<7000 \tempy@nss=7000 \fi%
    \set@nss{#1}{\tempy@nss} \block@nss{\tempy@nss}{1}%
    \divide\ydepth@nss by 2%
    \advance\y@nss by \ydepth@nss%
    \put(\x@nss,\y@nss){\line(1,1){\ydepth@nss}}%
    \put(\x@nss,\y@nss){\line(1,-1){\ydepth@nss}}%
    \tempx@nss=\x@nss\advance\tempx@nss by \xsize@nss%
    \put(\tempx@nss,\y@nss){\line(-1,1){\ydepth@nss}}%
    \put(\tempx@nss,\y@nss){\line(-1,-1){\ydepth@nss}}%
    \advance\y@nss by-\ydepth@nss%
}
```

\condindent@nss
\index@nss
\anzzeilen@nss

\index@nss, \anzzeilen@nss und \condindent@nss werden benötigt, um den Umbruch der Bedingung in einer \if-Abfrage geeignet durchführen zu können:

\index@nss	ein Schleifenindex
\anzzeilen@nss	die Anzahl von Zeilen, in die der Bedingungstext maximal zerlegt wird
\condindent@nss	enthält den jeweiligen Wert des Einzugs, um den Bedingungstext an das Dreieck anzupassen

Die Höhe der Zeilen wird bei der Berechnung mit 5mm angenommen.

```
\countdef\condindent@nss=232 \countdef\index@nss=233%
```

\gindhelp@nss
\indentmeasure@nss
\indentmeasureleft@nss
\thisindent@nss
\thisindentleft@nss
\thislength@nss

Um die Textform (\parshape) an beliebige Steigungen anpassen zu können, sind einige Rechenoperationen mit Hilfsgrößen erforderlich. Die folgenden dimen-Variablen wurden eingeführt, um die Textform (\parshape) an beliebige Steigungen anpassen zu können, die Bedeutungen sind im einzelnen:

\indentmeasure@nss:	Einheit des Einzugs (links)
\indentmeasureright@nss:	Einheit des Einzugs (rechts)
\thisindent@nss:	aktueller Einzug (links)
\thisindentright@nss:	aktueller Einzug (rechts)
\thislength@nss:	Länge der aktuellen Zeile, ergibt sich aus der Zeilenlänge \xsize@nss vermindert um \thisindent@nss und \thisindentright@nss

\gindhelp@nss ist eine einfache Hilfsvariable zur temporären Speicherung numerischer Werte.

```
\newdimen\indentmeasure@nss \newdimen\indentmeasureright@nss
\newdimen\thisindent@nss \newdimen\thisindentright@nss
\newdimen\thislength@nss
\countdef\gindhelp@nss=234
```

`\ifthenelse@nss` `\ifthenelse@nss` leitet eine einfache Verzweigung ein. Um Konflikte mit anderen Makropaketen zu vermeiden, wird eine Schreibweise benutzt, die auf Grund des Klammeraffens nicht direkt aufrufbar ist. Erst ein `\let` in der Definition der Umgebung `struktogramm` macht dieses Kommando aktiv.

```
\def\ifthenelse@nss{\let\next=\@ifthenelse@nss\getoption}
\def\@ifthenelse@nss#1#2#3#4#5%
```

Die 5 Parameter sind:

1. Die Steigung (negativ, invers) des linken Zweiges,
2. die Steigung (positiv, invers) des rechten Zweiges,
3. die Bedingung, die getestet wird,
4. Fall des linken Zweiges (üblicherweise „ja“) und
5. Fall des rechten Zweiges (üblicherweise „nein“).

```
{%
```

Als erstes werden die schrägen Linien gezeichnet. `\tempx@nss` ist die Breite des linken Zweiges, `\tempxx@nss` die des rechten Zweiges.

```
    %\ifthenelse{\boolean{anygradient}}%
    %{%
    \ifanygradient
```

Wenn `\anygradient` nicht wahr ist, werden schräge Linien mit `\line` erzeugt. `\xx@nss` ist die Anzahl von Einheiten, in die aktuelle Breite zerlegt wird.

```
        \xx@nss=#1 \advance\xx@nss by #2%
        \ifnum\number\param@nss000>0%
```

`\tempyy@nss` ist die Höhe des „Bedingungsvierecks“, `\tempx@nss` die linke und `\tempxx@nss` die rechte Breite.

```
            \tempyy@nss=\number\param@nss000%
            \tempx@nss=\xsize@nss%
            \multiply\tempx@nss by #1 \divide\tempx@nss by \xx@nss%
            \tempxx@nss=\xsize@nss%
            \multiply\tempxx@nss by #2 \divide\tempxx@nss by \xx@nss%
        \else%
            \tempyy@nss=\xsize@nss \divide\tempyy@nss by \xx@nss%
            \tempx@nss=\tempyy@nss \multiply\tempx@nss by #1%
            \tempxx@nss=\tempyy@nss \multiply\tempxx@nss by #2%
        \fi%
        \xx@nss=\tempx@nss \advance\xx@nss by \x@nss%
        \tempy@nss=\y@nss \advance\tempy@nss by -\tempyy@nss%
        \ifemlines%
            \emline{\x@nss}{\y@nss}{}{\xx@nss}{\tempy@nss}{}%
        \else\ifcurves%
            \curve(\x@nss, \y@nss, \xx@nss, \tempy@nss)%
        \else\ifpictIIe%
```

Die Steigungen dürfen jetzt fast beliebige Werte annehmen, Zähler und Nenner müssen jedoch im Intervall $[-1000, 1000]$ liegen, die Skalierung mit dem Faktor 1000 muss daher wieder rückgängig gemacht werden.

```
            \tempyyy@nss=\tempyy@nss
```

```
            \divide\tempyyy@nss by 1000
            \tempxxx@nss=\tempx@nss
            \divide\tempxxx@nss by 1000
            \put(\x@nss,\y@nss){\line(\tempxxx@nss, -\tempyyy@nss){\tempx@nss}}
        \fi\fi\fi%
        \advance\x@nss by \xsize@nss%
        \ifemlines%
            \emline{\x@nss}{\y@nss}{}{\xx@nss}{\tempy@nss}{}%
        \else\ifcurves%
            \curve(\x@nss, \y@nss, \xx@nss, \tempy@nss)
        \else\ifpictIIe%
            \tempxxx@nss=\tempxx@nss
            \divide\tempxxx@nss by 1000
            \put(\x@nss,\y@nss){\line(-\tempxxx@nss, -\tempyyy@nss){\tempxx@nss}}
        \fi\fi\fi%
```

Die Koordinaten werden auf die linke untere Ecke des „Bedingungsrechteckes“ gesetzt:

```
    \advance\x@nss by -\xsize@nss \advance\y@nss by -\tempyy@nss%
```

Nun wird die Form des Bedingungstextes berechnet (vgl. [Knu86, Kap. 14, S. 101–102]):

```
    \tempy@nss=\tempyy@nss\divide\tempy@nss by 5000%
```

Wenn die Höhe des „Bedingungsrechteckes“ zu gering ist, muss dennoch eine Zeile hineingeschrieben werden, ansonsten führt `\parshape` zu einem Fehler.

```
    \ifnum\tempy@nss<1 \tempy@nss=1\fi%
    \indentmeasure@nss=\xsize@nss\unitlength%
    \gindhelp@nss=#1\advance\gindhelp@nss by #2%
    \divide\indentmeasure@nss by \gindhelp@nss%
    \indentmeasureright@nss = \indentmeasure@nss%
    \multiply\indentmeasure@nss by #1%
    \multiply\indentmeasureright@nss by #2%
    \divide\indentmeasure@nss by \tempyy@nss%
    \divide\indentmeasureright@nss by \tempyy@nss%
    \phantom{\vbox{\hsize=0pt\global\gindhelp@nss=\the\baselineskip}}%
    \multiply\gindhelp@nss by 360%
    \multiply\indentmeasure@nss by \gindhelp@nss%
    \multiply\indentmeasureright@nss by \gindhelp@nss%
    \xdef\shape{}\index@nss=0%
    \loop%
        \advance\index@nss by 1% %Zahl der Zeile
        \thisindent@nss=\indentmeasure@nss%
        \multiply\thisindent@nss by \index@nss%
        \advance\thisindent@nss by 0.175cm%
        \xdef\shape{\shape\the\thisindent@nss}%
        \thisindentright@nss=\indentmeasureright@nss%
        \multiply\thisindentright@nss by \index@nss%
        \advance\thisindentright@nss by 0.175cm%
        \thislength@nss=\xsize@nss\unitlength%
        \advance\thislength@nss by -\thisindent@nss
        \advance\thislength@nss by -\thisindentright@nss
        \xdef\shape{\shape\the\thislength@nss}%
        \ifnum\index@nss<\tempy@nss%
    \repeat%
```

```
    %}%
    %{%
\else
    \ifnum\number\param@nss000>0%
        \typeout{struktex warning:}%
        \typeout{[...] not supplied without curves, emlines2, or pict2e}%
    \fi%
    \tempx@nss=#1 \advance\tempx@nss by #2%
    \tempyy@nss=\xsize@nss \divide\tempyy@nss by \tempx@nss%
    \tempx@nss=\tempyy@nss \multiply\tempx@nss by #1%
    \put(\x@nss,\y@nss){\line(#1,-1){\tempx@nss}}%
    \tempxx@nss=\tempyy@nss \multiply\tempxx@nss by #2%
    \advance\x@nss by \xsize@nss%
    \put(\x@nss,\y@nss){\line(-#2,-1){\tempxx@nss}}%
    \advance\x@nss by -\xsize@nss \advance\y@nss by -\tempyy@nss%
    \tempy@nss=\tempyy@nss\divide\tempy@nss by 5000%
    \ifnum\tempy@nss<1 \tempy@nss=1\fi%
    \xdef\shape{}\index@nss=0%
    \loop%
        \advance\index@nss by 1%
        \condindent@nss=#1\multiply\condindent@nss by\index@nss%
        \multiply\condindent@nss by 5000%
        \xdef\shape{\shape\the\condindent@nss\unitlength}%
        \condindent@nss=#1\advance\condindent@nss by #2%
        \multiply\condindent@nss by\index@nss%
        \multiply\condindent@nss by -5000%
        \advance\condindent@nss by \xsize@nss%
        \xdef\shape{\shape\the\condindent@nss\unitlength}%
        \ifnum\index@nss<\tempy@nss%
    \repeat%
    %}%
    \fi
    \put(\x@nss,\y@nss)%
    {%
        \framebox(\xsize@nss,\tempyy@nss)[tl]%
        {%
            \vbox%
            {%
                \hsize=\xsize@nss\unitlength%
                \parshape=\tempy@nss \shape \c@nter@nss%
                                                %dies sorgt f"ur den Einzug
                \noindent\vrule width0pt height12pt \relax #3%
                                               %hier wird der Text ausgegeben
            }%
        }%
    }%
    \divide\tempyy@nss by 2%
    \xx@nss=\xsize@nss\advance\xx@nss by-\tempx@nss%
    \advance\xx@nss by-\tempxx@nss%
    \ifodd\xx@nss\advance\tempxx@nss by 1\fi%
    \divide\xx@nss by 2\advance\tempxx@nss by \xx@nss%
    \advance\tempx@nss by \xx@nss%
    \put(\x@nss,\y@nss){\makebox(\tempx@nss,\tempyy@nss){#4}}%
    \xx@nss=\x@nss\advance\xx@nss by \tempx@nss%
```

```
    \put(\xx@nss,\y@nss){\makebox(\tempxx@nss,\tempyy@nss){#5}}%
    \str@kt{\x@nss}{\y@nss}{\tempx@nss}%
}
```

\change \change trennt die Zweige beim \ifthenelse@nss

```
\def\change%
{%
    \endstr@kt\str@kt{\xx@nss}{\y@nss}{\tempxx@nss}%
}
```

\ifend@nss \ifend@nss beendet \ifthenelse@nss, wird in struktogramm intern gemacht.

```
\def\ifend@nss%
{%
    \global\gy@nss=\y@nss\endgroup\tempy@nss=\gy@nss%
    \ifnum\tempy@nss<\ydepth@nss%
        \y@nss=\tempy@nss\advance\ydepth@nss by-\tempy@nss%
        \put(\x@nss,\y@nss)%
        {%
            \framebox(\tempx@nss,\ydepth@nss)%
            {%
                \ifnum\ydepth@nss>5000\filler@nss\fi%
            }%
        }%
    \else%
        \ifnum\tempy@nss>\ydepth@nss%
            \y@nss=\ydepth@nss\advance\tempy@nss by-\ydepth@nss%
            \put(\xx@nss,\y@nss)%
            {%
                \framebox(\tempxx@nss,\tempy@nss)%
                {%
                    \ifnum\tempy@nss>5000\filler@nss\fi%
                }%
            }%
        \else%
            \y@nss=\ydepth@nss%
        \fi%
    \fi%
}
```

\forever \foreverend Beginn und Ende einer Endlosschleife.

```
\def\forever{\let\next=\@forever\getoption}
\def\@forever{
    \tempx@nss=\xsize@nss
    \divide\tempx@nss by 6
    \set@nss{\mbox{\rule{0mm}{3ex}}}{\tempx@nss}
    \tempy@nss=\number\param@nss000
    \block@nss{\tempy@nss}{2}
    \advance\x@nss by \ydepth@nss
    \advance\y@nss by -\ydepth@nss
    \tempxx@nss=\xsize@nss
    \advance\tempxx@nss by -\ydepth@nss
    \tempy@nss=\ydepth@nss
    \str@kt{\x@nss}{\y@nss}{\tempxx@nss}
}
```

```
\def\foreverend{
    \endstr@kt
    \advance\x@nss by-\tempy@nss
    \advance\ydepth@nss by -\tempy@nss
    \tempyy@nss=\ydepth@nss
    \ydepth@nss=-\ydepth@nss
    \advance\ydepth@nss by \y@nss
    \advance\ydepth@nss by \tempy@nss
    \y@nss=\tempyy@nss
    \put(\x@nss,\y@nss){\framebox(\xsize@nss,\ydepth@nss){}}
    \put(\x@nss,\y@nss){\makebox(\xsize@nss,\tempy@nss){\box\TextBox@nss}}
}
```

`\dfr` `\dfrend` Aus Kompatibilitätsgründen werden für `\forever` und `\foreverend` noch die folgenden Abkürzungen bereitgestellt.

```
\let\dfr\forever
\let\dfrend\foreverend
```

`\while` `\whileend` Beginn und Ende einer kopfgesteuerten Schleife.

```
\def\while{\let\next=\@while\getoption}
\def\@while#1{\tempx@nss=\xsize@nss\divide\tempx@nss by 6
    \set@nss{#1\hskip\xsize@nss\unitlength}{\tempx@nss}
    \tempy@nss=\number\param@nss000 \block@nss{\tempy@nss}{0}
    \tempy@nss=\y@nss\advance\y@nss by \ydepth@nss
    \tempx@nss=\x@nss\advance\tempx@nss by \ydepth@nss
    \tempxx@nss=\xsize@nss\advance\tempxx@nss by -\ydepth@nss
    \str@kt{\tempx@nss}{\tempy@nss}{\tempxx@nss} }
\def\whileend{\endstr@kt\tempy@nss=\y@nss\advance\tempy@nss by-\ydepth@nss
    \y@nss=\ydepth@nss
    \put(\x@nss,\y@nss){\framebox(\xsize@nss,\tempy@nss){}} }
```

`\until` `\untilend` Beginn und Ende einer fußgesteuerten Schleife.

```
\def\until{\let\next=\@until\getoption}
\def\@until#1{\tempx@nss=\xsize@nss\divide\tempx@nss by 6
    \set@nss{#1\hskip\xsize@nss\unitlength}{\tempx@nss}
    \tempy@nss=\number\param@nss000 \block@nss{\tempy@nss}{2}
    \advance\x@nss by \ydepth@nss\tempxx@nss=\xsize@nss
    \advance\tempxx@nss by -\ydepth@nss \tempy@nss=\ydepth@nss
    \str@kt{\x@nss}{\y@nss}{\tempxx@nss} }
\def\untilend{\endstr@kt\advance\x@nss by-\tempy@nss
    \advance\ydepth@nss by -\tempy@nss
    \tempyy@nss=\ydepth@nss\ydepth@nss=-\ydepth@nss%
    \advance\ydepth@nss by \y@nss
    \y@nss=\tempyy@nss
    \put(\x@nss,\y@nss){\framebox(\xsize@nss,\ydepth@nss){}}
    \put(\x@nss,\y@nss){\makebox(\xsize@nss,\tempy@nss)
    {\box\TextBox@nss}}}
```

`\cases@nss` `\cases@nss` wird benötigt, um die Anzahl der Fälle im linken Zweig des `\case`-Konstruktes berechnen zu können. `\cases@nss` nimmt entweder die Anzahl der Fälle oder die Anzahl der Fälle -1 an.

\case
\caseend
\switch

\case, \caseend und \switch stellen eine Mehrfachverzweigung dar. Es sind derzeit noch die alten Versionen \caseold, \caseoldend und \switchold zusätzlich enthalten, diese sollten in einer späteren Version gestrichen werden.

Der Makro hat vier Parameter:

1. Die Steigung (negativ, invers) der schrägen Linie,
2. die Anzahl der Fälle,
3. den Text der Bedingung und
4. den des ersten Falles

Zusätzlich gibt es einen optionalen vorangestellten Parameter, der es im Falle der Benutzung des emlines2.sty ermöglicht, die schräge Linie mit einer beliebigen Steigung zu zeichnen. Wird dieser Parameter genutzt, hat der erste Parameter eine geänderte Bedeutung: ist der Parameter eine gerade Zahl, wird die Schräge gerade durchgezogen (kein *default*-Zweig), ansonsten wird der letzte Zweig als Standardfall angesehen und somit werden die Schrägen entsprechend gezeichnet: die ersten $n-1$ Fälle werden im linken Zweig untergebracht, der n-te Fall wird im rechten Zweig abgehandelt, vgl. dazu auch die Abbildungen auf Seite 16 und 17.

```
\def\case{\let\next=\@case\getoption}
\def\@case#1#2#3#4%
{
```

Merken der Anzahl der Fälle

```
    \tempxx@nss=\number#2
```

Zunächst die schräge Linie zeichnen

```
    \ifanygradient%
    %{%
```

– wenn \anygradient nicht wahr ist, werden schräge Linien mit \line erzeugt,

wobei die Höhe des Rahmens für die Fallunterscheidungen aus seiner Breite und der vorgegebenen Steigung berechnet wird –,

```
        \ifnum\number\param@nss000>0
```

Wenn die Höhe des Rahmens vorgegeben ist, nimm diese, ansonsten berechne sie wie oben.

```
            \tempyy@nss=\number\param@nss000%
            \ifodd\number#1
                \advance\tempxx@nss by -1
            \fi
        \else
            \tempyy@nss=\xsize@nss
            \divide\tempyy@nss by #1
        \fi
        \ifnum\number#2=\tempxx@nss
```

\tempxx@nss ist unverändert, es wird also die Diagonale (gerade Linie) gezeichnet. Die Berechnung der Koordinaten basiert auf:

\x@nss: linke obere x-Koordinate des Kastens
\y@nss: linke obere y-Koordinate des Kastens
\xsize@nss: Breite des Kastens
\tempyy@nss: Höhe des Kastens
\tempxx@nss: Anzahl der Fälle (evtl. reduziert um 1)

```
            \xx@nss=\x@nss \advance\xx@nss by \xsize@nss
            \tempy@nss=\y@nss \advance\tempy@nss by -\tempyy@nss
            \ifemlines%
                \emline{\x@nss}{\y@nss}{}{\xx@nss}{\tempy@nss}{}%
            \else\ifcurves%
                \curve(\x@nss, \y@nss, \xx@nss, \tempy@nss)%
            \else\ifpictIIe%
                \tempxxx@nss=\xx@nss
```

Die Argumente von `\line` für die Steigung müssen im Intervall $[-1000, 1000]$ liegen, daher wird hier der Faktor 1000 wieder wegdividiert. Abschließend wird die Länge der Strecke berechnet.

```
                \advance\tempxxx@nss by -\x@nss
                \divide\tempxxx@nss by 1000
                \tempyyy@nss=\tempy@nss
                \advance\tempyyy@nss by -\y@nss
                \divide\tempyyy@nss by 1000
                \advance\xx@nss by -\x@nss
                \put(\x@nss,\y@nss){\line(\tempxxx@nss, \tempyyy@nss){\xx@nss}}
            \fi\fi\fi%
        \else
            \tempx@nss=\xsize@nss
            \multiply\tempx@nss by \tempxx@nss \divide\tempx@nss by #2
            \xx@nss=\x@nss \advance\xx@nss by \tempx@nss
            \tempy@nss=\y@nss \advance\tempy@nss by -\tempyy@nss
            \ifemlines%
                \emline{\x@nss}{\y@nss}{}{\xx@nss}{\tempy@nss}{}%
            \else\ifcurves%
                \curve(\x@nss, \y@nss, \xx@nss, \tempy@nss)%
            \else\ifpictIIe%
```

Die Berechnung der Länge muss lokal sein, darum wird hier eine eigene Gruppe eingeführt anstatt die letzte Subtraktion rückgängig zu machen.

```
                \begingroup
                    \tempxxx@nss=\xx@nss
                    \advance\tempxxx@nss by -\x@nss
                    \divide\tempxxx@nss by 1000
                    \tempyyy@nss=\tempy@nss
                    \advance\tempyyy@nss by -\y@nss
                    \divide\tempyyy@nss by 1000
                    \advance\xx@nss by -\x@nss
                    \put(\x@nss,\y@nss){\line(\tempxxx@nss,
                        \tempyyy@nss){\xx@nss}}
                \endgroup
            \fi\fi\fi%
            \advance\x@nss by \xsize@nss
            \ifemlines%
                \emline{\x@nss}{\y@nss}{}{\xx@nss}{\tempy@nss}{}%
            \else\ifcurves%
                \curve(\x@nss, \y@nss, \xx@nss, \tempy@nss)%
            \else\ifpictIIe%
                \tempxxx@nss=\x@nss
                \advance\tempxxx@nss by -\xx@nss
                \divide\tempxxx@nss by 1000
                \tempyyy@nss=\tempy@nss
```

```
                \advance\tempyyy@nss by -\y@nss
                \divide\tempyyy@nss by 1000
                \xx@nss=-\xx@nss
                \advance\xx@nss by \x@nss
                \put(\x@nss,\y@nss){\line(-\tempxxx@nss, \tempyyy@nss){\xx@nss}}
            \fi\fi\fi%
            \advance\x@nss by -\xsize@nss
        \fi
    %}{
    \else% \ifanygradient
        \ifnum\number\param@nss000>0
            \typeout{struktex warning:}%
            \typeout{[...] not supplied without curves, emlines2, or pict2e}%
        \fi
        \put(\x@nss,\y@nss){\line(#1,-1){\xsize@nss}}
        \tempyy@nss=\xsize@nss
        \divide\tempyy@nss by #1
    %}
    \fi% \ifanygradient
```

dann den Bedingungstext in die Mitte des rechten oberen Viertels des CASE-Rechtecks schreiben,

```
    \tempx@nss=\xsize@nss
    \divide\tempx@nss by 2
    \divide\tempyy@nss by 2
    \advance\x@nss by \tempx@nss
    \advance\y@nss by -\tempyy@nss
    \put(\x@nss,\y@nss){\makebox(\tempx@nss,\tempyy@nss){#3}}
```

schließlich `\x@nss` und `\y@nss` auf den alten Wert zuruecksetzen und zur späteren Verwendung merken.

```
    \advance\x@nss by -\tempx@nss
    \advance\y@nss by \tempyy@nss
    \xx@nss=\x@nss
    \yy@nss=\y@nss
```

`\tempx@nss` ist die Breite eines Zweiges, die für alle Zweige gleich ist – Gesamtbreite geteilt durch Anzahl der Fälle –, `\tempy@nss` die Höhe des Falltextes, die von Zweig zu Zweig kleiner wird, `\condindent@nss` ist die Schrittweite, um die sich die Höhe des Falltextes bei jedem `\switch` verringert. Höhe und Schrittweite hängen davon ab, ob ein *default*-Zweig eingeführt wurde oder nicht, sie werden demzufolge durch `\tempxx@nss` gesteuert.

```
    \tempx@nss=\xsize@nss
    \divide\tempx@nss by #2
    \ifnum\number\param@nss000>0
        \tempy@nss=\number\param@nss000%
    \else
        \tempy@nss=\xsize@nss
        \divide\tempy@nss by #1
    \fi
    \condindent@nss=\tempy@nss
    \divide \condindent@nss by \tempxx@nss
```

Links oben im linken Zweig den Text des Falles eintragen

```
    \advance\y@nss by -\tempy@nss
```

```
    \put(\x@nss,\y@nss)%
    {%
        \makebox(\tempx@nss,\tempy@nss)[lb]{\raise3pt\hbox{~#4}}%
    }%
    \ydepth@nss=1000000
    \str@kt{\x@nss}{\y@nss}{\tempx@nss}
}
```

Der Makro `\switch` für die einzelnen Fälle hat zwei Parameter:

1. einen optionalen Parameter, der angibt, wo der Text der Bedingung gesetzt wird: `l` oder `r` – der Standardwert ist `l` –, und
2. den Bedingungstext

```
\def\switch%
{%
    \@ifnextchar [{\@switch}{\@switch[l]}%
}
\def\@switch[#1]#2%
{
    \global\gy@nss=\y@nss\endgroup
    \tempyy@nss=\gy@nss
    \ifnum\tempyy@nss<\ydepth@nss
        \ydepth@nss=\tempyy@nss
    \fi
```

Nun in den nächsten Zweig gehen und die Falltexthöhe vermindern.

```
    \advance\x@nss by \tempx@nss
    \advance\tempy@nss by -\condindent@nss
    \put(\x@nss,\y@nss)%
            {\makebox(\tempx@nss,\tempy@nss)[#1b]{\raise3pt\hbox{~#2~}}}
    \str@kt{\x@nss}{\y@nss}{\tempx@nss}
}
\def\caseend%
{
    \global\gy@nss=\y@nss\endgroup
    \tempyy@nss=\gy@nss
    \ifnum\tempyy@nss<\ydepth@nss
        \ydepth@nss=\tempyy@nss
    \fi
    \x@nss=\xx@nss
    \y@nss=\ydepth@nss
    \tempyy@nss=\yy@nss
    \advance\tempyy@nss by -\ydepth@nss
    \put(\x@nss,\y@nss){\framebox(\xsize@nss,\tempyy@nss){}}
    \tempxx@nss=\x@nss
```

Die folgende Schleife ist dafür zuständig, die Trennlinien zwischen den Falltexten zu ziehen.

```
    \loop
        \put(\tempxx@nss,\y@nss){\line(0,1){\tempyy@nss}}
        \advance\tempyy@nss by -\condindent@nss
        \advance\tempxx@nss by \tempx@nss
```

Die Bedeutung der folgenden Zeilen ist noch unklar, was soll die 100?

```
        \xx@nss=\tempxx@nss
        \advance\xx@nss by -\x@nss
        \advance\xx@nss by 100
        \ifnum \xx@nss<\xsize@nss
    \repeat
}
```

\centerNss
\endcenterNss

Makro zur Zentrierung von Struktogrammen: da einfaches Zentrieren mit `\centering` nicht funktioniert, wird mit `\centerline` gearbeitet:

```
\newbox\CenterBox@nss%
\def\centernss%
{%
    \begin{trivlist}%
    \item[] \strut%
        \setbox\CenterBox@nss=\hbox%
                                \bgroup%
}
\def\endcenternss%
{%
                                \egroup%
        \strut\hfill\box\CenterBox@nss\hfill\strut%
    \end{trivlist}%
}
```

\CenterNssFile

Makro zur Zentrierung von Struktogrammen: da einfaches Zentrieren mit `\centering` nicht funktioniert, wird mit `\centerline` gearbeitet:

```
\def\CenterNssFile#1%
{%
    \begin{trivlist}%
        \item[] \setbox\CenterBox@nss=\hbox{\input{#1.nss}}%
        \strut\hfill\box\CenterBox@nss\hfill\strut%
    \end{trivlist}%
}
\let\centernssfile=\CenterNssFile
```

9.6 Zusätzliche Makros (Verifikation)

\assert
\assert@nss

`\assert` wird wie `\assign` definiert, führt ohne die Option `verification` aber nicht zu sichtbaren Ergebnissen:

```
  \newcommand\assert{\let\next=\assert@nss\getoption}%
  \newcommand\assert@nss[1]{}%
```

Der folgende Code ist nur gültig, wenn die Option `verification` gesetzt wurde:

```
\ifthenelse{\boolean{verification}}
{
```

\xsizeo@nss
\ydeptho@nss

Da `\oval` einen anderen Referenzpunkt als `\framebox` hat, brauchen wir zusätzliche Zähler, um den anderen Referenzpunkt ansprechen zu können.

```
    \countdef\xsizeo@nss=232%
    \countdef\ydeptho@nss=233%
```

Für den Fall, dass die Option `verification` gesetzt wurde, wird der intern genutzte Makro `\assert@nss` neu definiert.

```
\renewcommand{\assert@nss}[1]%
{%
  \tempx@nss=\xsize@nss\divide\tempx@nss by 6%
  \set@nss{#1}{\tempx@nss}\tempy@nss=\number\param@nss000%
  \assertblock@nss{\tempy@nss}{1}%
}
```

`\assertblock@nss` `\assertblock@nss` setzt die Box `\TextBox@nss` entsprechend dem zweiten Parameter. Ist dieser 1, wird eine Box mit Rahmen gesetzt, ist dieser 0, wird eine Box ohne Rahmen gesetzt. Bei allen anderen Werten wird nur die Größe des Struktogramms, gegeben durch `\y@nss`, vergrößert.

```
\newcommand{\assertblock@nss}[2]%
{%
  \tempy@nss=#1\TextBoxHt@nss=\ht\TextBox@nss%
  \advance\TextBoxHt@nss by\dp\TextBox@nss%
  \advance\TextBoxHt@nss by 2mm%
  \ifdim\TextBoxHt@nss<\tempy@nss\unitlength%
      \TextBoxHt@nss=\tempy@nss\unitlength%
  \fi%
  \TextBoxHt@nss=0.00536\TextBoxHt@nss%
  \ydepth@nss=\TextBoxHt@nss\advance\y@nss by-\ydepth@nss%
  \xsizeo@nss=\xsize@nss \tempx@nss=\x@nss
  \divide\xsizeo@nss by 2%
  \advance\tempx@nss by+\xsizeo@nss%
  \ydeptho@nss=\ydepth@nss \tempy@nss=\y@nss
  \divide\ydeptho@nss by 2%
  \advance\tempy@nss by+\ydeptho@nss%
  \ifx#21%
      \put(\x@nss,\y@nss)%
      {%
          \framebox(\xsize@nss,\ydepth@nss){\box\TextBox@nss}%
      }%
      \put(\tempx@nss,\tempy@nss){\oval(\xsize@nss,\ydepth@nss)}%
  \else%
      \ifx#20%
          \put(\x@nss,\y@nss)%
          {%
              \makebox(\xsize@nss,\ydepth@nss){\box\TextBox@nss}%
          }%
          \put(\tempx@nss,\tempy@nss){\oval(\xsize@nss,\ydeptho@nss)}%
      \else%
          \advance\y@nss by \ydepth@nss%
      \fi%
  \fi%
}
}
{} % end of \ifthenelse{\boolean{verification}}
```

Literatur

[Fut89] Gerald Futschek. *Programmentwicklung und Verifikation*. Springer Verlag, Wien – New York, 1989. 2, 43

[GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LaTeX-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994. 2, 54

[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. 58

[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 TeXnische Komödie*, 8(4):23–40, Februar 1996. 29

[Rah92] Sebastian Rahtz. *The* oz *package*, 1992. 44

Index

Die kursiven Zahlen bezeichnen die Seite, auf der der jeweilige Eintrag beschrieben ist, unterstrichene Zahlen verweisen auf seine Definition; alle anderen Zahlen geben Stellen an, an denen der Eintrag auftritt.

R

S

T

U

V

W

X

Y

Z

Revisionsgeschichte

6.0

General: Einsatz von curves.sty zum Zeichnen beliebiger Steigungen implementiert 14

v1.41

General: Andreas Wagener, Prozeßrechner CYBER 815 1

v1.5

General: StrukTeX Makros nun als .sty-Datei, J. Hoffmann 3