„Bafög"-Ausgabe

Frank Mittelbach
Michel Goossens

Der LaTeX Begleiter

st scientific tools

2. überarbeitete und erweiterte Auflage

Unter Mitarbeit von J. Braams, D. Carlisle und C.Rowley

PEARSON Studium

KAPITEL 3

Formatierungswerkzeuge

Die Art und Weise, in der Informationen visuell präsentiert werden, kann das Verständnis einer Aussage durch den Leser stark beeinflussen. Darum ist es wichtig, die besten verfügbaren Werkzeuge zu verwenden, um die genaue Bedeutung der eigenen Worte zu vermitteln. Gleichzeitig gilt jedoch, dass Formen der visuellen Darstellung dem Benutzer immer nur das Verständnis eines Textes erleichtern dürfen, nicht aber seine Aufmerksamkeit ablenken. Daher sind eine einheitliche Darstellung und einheitliche Konventionen für die visuellen Merkmale ein absolutes Muss. Ebenso sollte die Art und Weise, in der gegebene Strukturelemente hervorgehoben werden, im gesamten Dokument immer gleich sein. Diese Regeln lassen sich leicht festschreiben, indem man für jedes Dokumentenelement, das besonders behandelt werden soll, einen eigenen Befehl oder eine Umgebung definiert und diese Befehle und Umgebungen in einer Paketdatei oder in der Dokumentenpräambel zusammenfasst. Indem man dann ausschließlich diese Befehle verwendet, ist eine einheitliche Präsentationsform gewährleistet.

Das vorliegende Kapitel erklärt verschiedene Wege, Teile eines Dokumentes hervorzuheben. Der erste Abschnitt befasst sich mit der Hervorhebung kurzer Textfragmente oder Absätze und beschreibt Werkzeuge, mit denen diese verändert werden können.

Der zweite Abschnitt wirft einen Blick auf die verschiedenen Arten von Anmerkungen, wie z.B. Fußnoten, Marginalien und Endnoten, und erläutert, wie ihr Layout angepasst werden kann.

Der dritte Abschnitt behandelt dann das Formatieren von Listen. Zunächst werden die verschiedenen Parameter und Befehle untersucht, mit denen die LaTeX-Listenumgebungen `enumerate`, `itemize` und `description`

gesteuert werden. Anschließend werden die Erweiterungen, die durch das paralist-Paket bereitgestellt werden, sowie das Konzept der „theoremähnlichen Strukturen“ am Beispiel des amsthm-Paketes besprochen. Für die Struktur- und Layoutanforderungen der meisten Leser sind diese Erweiterungen ausreichend. Für alle anderen wird im restlichen Teil des Abschnitts die allgemeine Listenumgebung `list` vorgestellt. Es wird gezeigt, wie sich durch Variieren der Parameterwerte, mit denen die Umgebung `list` gesteuert wird, Layouts anpassen lassen.

Der vierte Teil beschreibt, wie die wortwörtliche Ausgabe von Text (englisch: verbatim) simuliert werden kann. In diesem Zusammenhang gilt ein spezielles Augenmerk den leistungsfähigen Paketen fancyvrb und listings.

Das Kapitel schließt mit der Betrachtung von Paketen, die sich mit der Zeilennummerierung, Handhabung von Spalten (etwa von parallelem Text in zwei Spalten) oder den Problemen beim mehrspaltigen Satz beschäftigen.

3.1 Textfragmente und Absätze

Dieser Abschnitt befasst sich mit Textfragmenten und erläutert, wie sie bearbeitet und hervorgehoben werden können, indem man ihnen ein anderes Erscheinungsbild zuweist als dem übrigen Text.

Zunächst wird besprochen, wie man Befehle definiert, die automatisch korrekten Leerraum nach ihrer Anwendung erzeugen. Anschließend wird gezeigt, wie professionelle Auslassungspunkte erzeugt werden.

Zur Hervorhebung von Text können Parameter wie Schriftform, -stärke und -grad angepasst werden (siehe Abschnitt 7.3.1 auf Seite 349). Genauso kann Text unterstrichen oder die Laufweite einer Schrift verändert werden. Mittel, mit denen sich diese Effekte erzielen lassen, werden durch die Pakete relsize, textcase, ulem und soul angeboten.

Der Rest des Abschnitts wendet sich absatzbezogenen Fragen zu, wie etwa der Erzeugung von großen Anfangsbuchstaben zu Beginn eines Absatzes, der Änderung der Absatzausrichtung oder Einstellung des Zeilenvorschubs innerhalb eines Absatzes sowie dem Einfügen von rechteckigen Löchern in einem Absatz, die unter anderem mit kleinen Bildern gefüllt werden können.

3.1.1 xspace – Korrekte Leerräume nach Makros

Das kleine Paket xspace (von David Carlisle) definiert den Befehl `\xspace`, der am Ende von Makros verwendet werden sollte, die hauptsächlich zur Textproduktion dienen. Er fügt ein Leerzeichen ein, sofern dem Makro nicht bestimmte Satzzeichen folgen.

Durch die Verwendung von `\xspace` müssen Befehle ohne Argument nicht mehr mit `\␣` oder `{}` abgeschlossen werden, damit nach ihnen ein Leerzeichen ausgegeben wird. Wenn jedoch eine dieser Konstruktionen auf `\xspace` folgt, fügt `\xspace` kein zusätzliches Leerzeichen ein. Das bedeutet, dass man `\xspace` ohne Risiko an das Ende eines bereits existierenden Makros anfügen kann, ohne dass man Änderungen in seinem Dokument vornehmen müsste. Mögliche Kandidaten für die Verwendung von `\xspace` sind

Befehle für Abkürzungen, wie „z.B.“ oder „d.h.“.

```
\newcommand\zB{}{z.B.,\xspace}
\newcommand\xdh{}{d.h.,\xspace}
\newcommand\usw{}{usw.\@\xspace}
```

Wichtig ist die richtige Verwendung des Befehls `\@` zur Erzeugung der richtigen Art von Leerraum: Rechts von einem Punkt verhindert er, dass zusätzlicher Abstand eingefügt wird, da der Punkt in diesem Fall nicht als ein Satzendezeichen gewertet wird. Links von einem Punkt bewirkt er, dass LaTeX den Punkt immer als Satzendezeichen interpretiert.

Unter Umständen kann es vorkommen, dass `\xspace` an der falschen Stelle einen Leerraum einfügt. In solchen Fällen kann man an das Makro einfach `{}` anschließen, wodurch der Leerraum unterdrückt wird.

Bsp. 3-1-1

Die Bundesrepublik Deutschland hat 16 Länder.
Großbritannien, die Bundesrepublik Deutschland und Frankreich haben enge kulturelle Verbindungen.

```
\usepackage{xspace}
\newcommand\BRD{Bundesrepublik Deutschland\xspace}
\newcommand\GB {Großbritannien\xspace}
Die \BRD hat 16 Länder.\\ \GB, die \BRD und
Frankreich haben enge kulturelle Verbindungen.
```

3.1.2 ellipsis, lips – Auslassungspunkte

Auslassungspunkte werden im Allgemeinen durch drei aufeinander folgende Punkte dargestellt (auch als *Ellipse* bekannt). Der Abstand der Punkte untereinander hängt von typographischen und verlagsspezifischen Konventionen ab. Hier sind deutliche Unterschiede zu beobachten. Im Französischen werden nach Hart [65] oder dem *Chicago Manual of Style* [38], „points de suspension“ dicht aneinander gesetzt und sie folgen dem vorangegangenen Wort mit einem Leerraum auf der rechten Seite:

C’est une chose… bien difficile.

Im Deutschen haben Auslassungspunkte gemäß Duden [45] einen Leerraum auf der linken *und* rechten Seite, es sei denn, sie stellen ausgelassene Buchstaben innerhalb eines Wortes dar oder ein Satzzeichen folgt:

Du E… du! Scher dich zum …!

In der britischen oder amerikanischen Typographie dagegen werden die Punkte mit vollem Wortabstand gesetzt und recht komplizierte Regeln legen fest, wie andere Satzzeichen an beiden Enden behandelt werden.

LaTeX bietet die Befehle `\dots` und `\textellipsis`, um Auslassungspunkte mit kleinem Abstand zu erzeugen. Leider produziert die Standarddefinition (die aus Plain-TeX stammt) einen ungleichmäßigen Abstand auf der linken und rechten Seite – dies ist für den korrekten Satz von einigen der obigen Beispiele völlig ungeeignet. Der zusätzliche kleine Abstand rechts der Auslassungspunkte ist in manchen Situationen richtig (z.B. wenn ein Satzzeichen folgt). Wenn den Auslassungspunkten ein Leerzeichen folgt, sieht dieser

Abstand jedoch recht unpassend aus und sollte am besten weggelassen werden, wie das Beispiel unten zeigt (auch wenn durch Auslassen des Abstandes im zweiten Fall das Ausrufezeichen ein wenig zu nah an die Auslassungspunkte rutscht).

```
\newcommand\lips{\dots\unkern}
Ein Beispiel zum Vergleich:\\
  Du E\dots\ du! Scher dich zum \dots!\\
  Du E\lips\ du! Scher dich zum \lips!
```

Bsp. 3-1-2

Ein Beispiel zum Vergleich:
Du E… du! Scher dich zum …!
Du E… du! Scher dich zum …!

Dieses Problem wird mit dem Paket ellipsis von Peter Heslin angepackt. Es definiert den Befehl `\dots` neu. Für die Entscheidung, ob ein Nachabstand gesetzt werden soll, wird das auf die Auslassungspunkte folgende Zeichen herangezogen. Ein zusätzlicher Leerraum wird nur eingefügt, wenn das folgende Zeichen im Befehl `\ellipsispunctuation` aufgelistet ist, der standardmäßig den Wert „`,.:;!?`“ hat. Bei einigen Sprachunterstützungspaketen werden bestimmte Zeichen aktiviert. Werden diese in der Liste verwendet, muss sie neu deklariert werden, damit das Paket die Zeichen weiterhin erkennt.

Der Abstand zwischen den Punkten sowie der hinter den Auslassungspunkten gegebenenfalls eingefügte Leerraum kann über den Befehl `\ellipsisgap` festgelegt werden. Automatische Anpassungen, die vom Schriftgrad abhängen, lassen sich über fontabhängige Einheiten, wie `em`, oder einen Bruchteil eines `\fontdimen`-Befehls steuern (siehe Seite 439).

```
\usepackage{ellipsis}
Ein Beispiel zum Vergleich:\\
  Du E\dots\ du! Scher dich zum \dots!\\
\renewcommand\ellipsisgap{1.5\fontdimen3\font}
  Du E\dots\ du! Scher dich zum \dots!\\
\renewcommand\ellipsisgap{0.3em}
  Du E\dots\ du! Scher dich zum \dots!
```

Bsp. 3-1-3

Ein Beispiel zum Vergleich:
Du E… du! Scher dich zum …!
Du E… du! Scher dich zum … !
Du E… du! Scher dich zum … !

Für den Spezialfall, dass Auslassungspunkte mitten im Wort benötigt werden (oder aus anderen Gründen nur ein kleiner Abstand auf beiden Seiten gewünscht ist), bietet das Paket den Befehl `\midwordellipsis`. Wenn das Paket mit der Option `mla` (Modern Language Association Style) geladen wurde, werden die Auslassungspunkte automatisch eingeklammert und hinter dem letzten Punkt folgt kein zusätzlicher Abstand.

Gemäß dem *Chicago Manual of Style* [38] werden Auslassungspunkte mit vollem Wortabstand zwischen den Punkten gesetzt. Dieses Erscheinungsbild erreicht man mit dem Paket lips[1] von Matt Swift. Es implementiert den Befehl `\lips`, der den Empfehlungen in diesem Referenzbuch folgt. Auslassungspunkte, die beispielsweise das Fehlen des Satzendes markieren, sollten nach [38, §10.48–63] aus vier Punkten bestehen, wobei der *erste* Punkt das Satzendezeichen darstellt.[2] Der Befehl `\lips` realisiert dies, indem er „`\lips.`“ als „`.\lips`“ interpretiert, wie das nächste Beispiel zeigt.

[1] lips ist, genaugenommen, Teil einer größeren Paketsammlung. Wenn nur dieses Paket verwendet werden soll, muss auch das Paket moredefs desselben Autors geladen werden.

[2] Nicht, dass die Autoren dieses Buches irgendeine Logik darin sehen könnten …

Bsp. 3-1-4

In . . . werden Auslassungspunkte normalerweise mit vollem Wortabstand gesetzt. . . . Ein Beispiel hierfür ist dieser Absatz.

```
\usepackage{moredefs,lips}
In \lips werden Auslassungspunkte
normalerweise mit vollem Wortabstand gesetzt
\lips. Ein Beispiel hierfür ist dieser Absatz.
```

Der Befehl `\lips` sucht nach Satzzeichen, die auf den Befehl folgen, und sorgt dafür, dass Auslassungspunkte und Satzzeichen im Fall von „`,:;?!)'`]/“ nicht durch einen Zeilenumbruch getrennt werden. In anderen Fällen (z.B. einer öffnenden Klammer) ist ein Zeilenumbruch zugelassen. Die obige Liste wird in `\LPNobreakList` gespeichert und kann bei Bedarf angepasst werden. Um einen umbruchgeschützten Leerraum nach `\lips` zu erzwingen, muss der Befehl durch eine Tilde ergänzt werden (`~`).

Bei Verwendung der Option `mla` werden die erzeugten Auslassungspunkte automatisch eingeklammert und ein Punkt nach dem Befehl `\lips` wird nicht an den Anfang verschoben. Wenn erforderlich, erzeugt `\olips` die ursprüngliche Version ohne Klammern.

Bsp. 3-1-5

In . . . werden Auslassungspunkte mit vollem Wortabstand zwischen den Punkten gesetzt [. . .]. Ein Beispiel hierfür ist dieser Absatz.

```
\usepackage{moredefs}\usepackage[mla]{lips}
In \olips werden Auslassungspunkte mit
vollem Wortabstand zwischen den Punkten gesetzt
\lips. Ein Beispiel hierfür ist dieser Absatz.
```

3.1.3 amsmath – Geschützte Bindestriche

Das amsmath-Paket, das ausführlich in Kapitel 8 besprochen wird, bietet ebenfalls einen Befehl zur Verwendung in Absätzen. Der Befehl `\nobreakdash` unterdrückt jede Möglichkeit, nach dem Bindestrich einen Zeilenumbruch vorzunehmen. Eine sehr verbreitete Verwendung von `\nobreakdash` ist die Unterdrückung von unerwünschten Zeilenumbrüchen in Fällen wie „p-adic“, aber es gibt noch ein weiteres Beispiel: Wenn „Seite 3–9“ als `Seite 3\nobreakdash--9` kodiert ist, wird zwischen dem Bindestrich und der 9 kein Zeilenumbruch eingefügt.

Dieser Befehl muss *unmittelbar* vor dem Bindestrich (`-`, `--` oder `---`) angegeben werden. Das folgende Beispiel zeigt, wie ein Zeilenumbruch nach einem Bindestrich unterdrückt wird, aber die normalen Bindestriche im folgenden Wort zugelassen werden (es reicht, nach dem Bindestrich einen Leerraum ohne horizontale Ausdehnung einzufügen). Bei häufiger Verwendung empfiehlt es sich, Abkürzungen wie `\p` zu definieren. Danach wird das Wort „dimensionalen“ umbrochen, während ein Umbruch nach „p-“ verhindert (was im Beispiel zur überlaufenden Zeile führt) und dann „3–9“ auf die nächste Zeile verschoben wird.

Bsp. 3-1-6

Die Verallgemeinerung des n-dimensionalen Falls (mittels Standard p-adic-Topologie) kann man auf den Seiten 3–9 von Band IV finden.

```
\usepackage{amsmath}
\newcommand\p{$p$\nobreakdash}
\newcommand\Ndash{\nobreakdash--}
\newcommand\n[1]{$n$\nobreakdash-\hspace{0pt}}
Die Verallgemeinerung des \n-dimensionalen
Falls (mittels Standard \p-adic{}-Topologie) kann
man auf den Seiten 3\Ndash 9 von Band IV finden.
```

3.1.4 relsize – Relative Skalierung der Schriftgröße

Standard-LaTeX bietet zehn vordefinierte Befehle zur Änderung der Schriftgröße (siehe Tabelle 7.1 auf Seite 353). Die von ihnen erzeugten Schriftgrade werden jeweils von der Dokumentenklasse festgelegt, bleiben aber innerhalb eines Dokumentes vom Wert unverändert. Dies bedeutet, dass z.B. `\small` in einem Dokument immer denselben Wert wählt, ungeachtet der umgebenden Bedingungen.

In vielen Fällen ist es jedoch wünschenswert, die Schriftgröße relativ zur aktuellen Größe zu ändern. Dies lässt sich durch das Paket relsize bewerkstelligen, das ursprünglich von Bernie Cosell entwickelt und später für LaTeX 2ε von Donald Arseneau und Matt Swift aktualisiert und erweitert wurde.

Das Paket enthält den deklarativen Befehl `\relsize`, der als Argument eine Zahl erwartet. Diese gibt die Anzahl an Schritten an, um welche die Schriftgröße geändert werden soll. Wenn z.B. die aktuelle Schriftgröße `\Large` ist, ändert `\relsize{-2}` die Schriftgröße auf `\normalsize`. Wenn die gewünschte Anzahl an Schritten nicht verfügbar ist, wird die kleinste (d.h. `\tiny`) bzw. größte (d.h. `\Huge`) Schriftgröße gewählt. Eine Umkehr der Skalierung durch Negation des Argumentwertes muss also nicht notwendigerweise wieder zur Ausgangsgröße führen. Besser ist es, nur lokale Veränderungen vorzunehmen (etwa innerhalb einer Gruppe geschweifter Klammern) und LaTeX die Rückänderungen selbst zu überlassen.

Das Paket definiert außerdem `\smaller` und `\larger`, welche einfach Abkürzungen für den Befehl `\relsize` mit den Argumenten `-1` bzw. `1` sind. Weitere bequeme Alternativen sind `\textsmaller` und `\textlarger`, die als Argument den zu verkleinernden oder zu vergrößernden Text haben. Diesen vier Befehlen kann als optionales Argument die Anzahl der Skalierungsschritte mitgegeben werden, wenn ein anderer Wert als der voreingestellte Wert `1` benötigt wird.

Großer Text mit ein paar kleinen Worten integriert.

KAPITÄLCHEN (gefälscht)
KAPITÄLCHEN (echt; vgl. Lauflänge und Strichstärke zur vorherigen Zeile).

```
\usepackage{relsize}
\Large Großer Text mit ein paar \\
  {\relsize{-2}kleinen Worten} integriert.
\par\medskip
\normalsize\noindent
K\textsmaller[2]{APITÄLCHEN} (gefälscht)\\
\textsc{Kapitälchen} (echt; vgl.\ Lauf\/länge
  und \\ Strichstärke zur vorherigen Zeile).
```

Bsp. 3-1-7

Die obige Beschreibung für `\relsize` ist nicht ganz korrekt: Bei jedem Schritt versucht der Befehl, die Schriftgröße um 20% zu vergrößern oder zu verkleinern, und wählt dann den Schriftgrößenbefehl von LaTeX, welcher der gewünschten Zielgröße am nächsten kommt. Anschließend werden die gewünschte Schriftgröße und die Zielgröße verglichen. Wenn sie um mehr als den aktuellen Wert von `\RSpercentTolerance` (angegeben in Prozent) differieren, ruft das Paket den Befehl `\fontsize` mit der Zielgröße als einem der Parameter auf. Bei diesem Prozess versucht LaTeXs Zeichensatz-Auswahlschema, einen Font zu finden, welcher der Anforderung am nächsten kommt. Standardmäßig ist `\RSpercentTolerance` ein leeres Makro, das als 30 (Prozent) interpretiert wird, wenn die aktuelle Schriftformgruppe nur

aus separaten Größen besteht (siehe Abschnitt 7.10.3), und als 5, wenn die Schriftformdefinition einen Bereich von Schriftgrößen umfasst.

In einigen Fällen kann ein fester Faktor von `1.2` für jeden Schritt eine zu große Einschränkung darstellen. Aus diesem Grund bietet das Paket zusätzlich den allgemeineren deklarativen Befehl `\relscale{`*faktor*`}` und seine Variante `\textscale{`*faktor*`}{`*text*`}`, mit denen die Schriftgröße basierend auf dem gegebenen *faktor*, wie etwa `1.3` (Vergrößerung um 30%), ausgewählt werden kann.

Zwei weitere Befehle dienen zur Skalierung im mathematischen Modus: `\mathsmaller` und `\mathlarger`. LaTeX erkennt nur vier verschiedene mathematische Schriftgrößen, von denen zwei (`\displaystyle` und `\textstyle`) für die meisten Symbole nahezu identisch sind; dadurch ist das Anwendungsgebiet dieser Befehle etwas begrenzt. Die Situation verbessert sich geringfügig, wenn zusätzlich exscale geladen wird: Der Befehl `\mathlarger` schaltet dann, bei Verwendung in `\displaystyle`, intern auf eine größere Textschriftgröße um und wählt anschließend den zu dieser Schriftgröße gehörenden `\displaystyle`.

Bsp. 3-1-8

$$\sum \neq \sum$$

und $\frac{1}{2} \neq \frac{1}{2}$ aber $N = N$

```
\usepackage{exscale,relsize}
\[ \sum \neq \mathlarger{\sum} \]
und $\frac{1}{2} \neq \frac{\mathlarger 1}
{2}$ aber $N = \mathlarger {N}$
```

Diese Befehle versuchen, bei großen Operatoren die hoch- und tiefgestellten Indizes richtig zu positionieren. Zum Beispiel:

Bsp. 3-1-9

$$\sum_{i=1}^{n} \neq \sum_{i=1}^{n} \neq \sum_{i=1}^{n} \qquad \int_0^\infty \neq \int_0^\infty \neq \int_0^\infty$$

```
\usepackage{exscale,relsize}
\[
 \mathsmaller\sum_{i=1}^n \neq
 \sum_{i=1}^n \neq \mathlarger\sum_{i=1}^n
\qquad  \mathsmaller\int_0^\infty \neq
 \int_0^\infty \neq \mathlarger\int_0^\infty
\]
```

Zu beachten ist, dass die Verwendung dieser Befehle innerhalb von Formeln die wahre Natur der mathematischen Objekte in ihrem Argument verbergen, so dass die Abstände in der Formel ohne zusätzliche Hilfe missraten können. Wie das folgende Beispiel zeigt, muss man deshalb möglicherweise explizit `\mathrel`, `\mathbin` oder `\mathop` verwenden, um korrekte Abstände zu erhalten.

Bsp. 3-1-10

$$a \times b \neq a \times b \neq a \times b$$

```
\usepackage{exscale,relsize}
\[
   a \times b \neq  a \mathlarger{\times} b \neq
   a \mathbin{\mathlarger\times} b
\]
```

Aufgrund dieser Unwägbarkeiten sollten die Befehle `\mathlarger` und `\mathsmaller` nicht blindlings eingesetzt werden, sie sind nicht in jedem Fall empfehlenswert.

3.1.5 textcase – Intelligente Groß- und Kleinschreibung

Die LaTeX-Standardbefehle `\MakeUppercase` und `\MakeLowercase` ändern die Schreibweise der Zeichen ihrer Argumente in Groß- bzw. Kleinschreibung, wodurch die Makros nach Bedarf expandiert werden. Beispielsweise ergibt

```
\MakeUppercase{Am \today}
```

„AM 29. MÄRZ 2005“. Manchmal werden dadurch mehr Zeichen als gewünscht geändert. Wenn der Text z.B. eine mathematische Formel enthält, ist die Umsetzung in Großbuchstaben nicht so empfehlenswert, weil sich damit die Bedeutung ändert. Auf ähnliche Weise stellen die Argumente der Befehle `\label`, `\ref` und `\cite` semantische Informationen dar, die bei Änderung der Schreibweise zu falschen oder fehlenden Querverweisen führen, weil LaTeX nach den falschen Einträgen sucht.

`\MakeTextUppercase{`*text*`}`	`\MakeTextLowercase{`*text*`}`

Das Paket textcase von David Carlisle löst die oben angesprochenen Probleme, indem es `\MakeTextUppercase` und `\MakeTextLowercase` als Alternativbefehle bereitstellt, die mathematische Formeln und Querverweise erkennen und diese unberücksichtigt lassen.

1 Groß-/Kleinschreibung

TEXT IM ABSCHNITT 1, ÜBER $a = b$ UND $\alpha \neq a$

```
\usepackage{textcase}
\section{Groß-/Kleinschreibung}\label{exa}
\MakeTextUppercase{Text im Abschnitt~\ref{exa},
  über $a=b$ und \(\alpha \neq a \) }
```

Bsp. 3-1-11

Manchmal sollten Teile des Textes aus dem einen oder anderen Grund unverändert bleiben. Mit `\NoCaseChange` bietet das Paket eine allgemeine Vorgehensweise, um solche Teile zu markieren. Ein Beispiel:

EIN BISSCHEN TEXT Noch mehr TEXT

```
\usepackage{textcase}
\MakeTextUppercase{Ein bisschen Text
    \NoCaseChange{Noch mehr} Text}
```

Bsp. 3-1-12

Wenn notwendig, kann dieses Verfahren verwendet werden, um syntaktische Informationen zu verbergen, wie in

```
\NoCaseChange{\begin{tabular}{ll}}...\NoCaseChange{\end{tabular}}
```

Dadurch wird verhindert, dass `tabular` und `ll` fälschlicherweise großgeschrieben werden.

Dies alles funktioniert nur, wenn sich der betreffende Text auf der obersten Ebene befindet. Alles, was in geschweiften Klammern steht (mit Ausnahme der Argumentklammern von `\label`, `\ref`, `\cite` oder `\NoCaseChange`), wird in Groß- bzw. Kleinschreibung umgewandelt, ungeachtet dessen, um was es sich handelt.

DIESE BEIDEN SCHLAGEN IMMER **FEHL** $A + B = C$, *LEIDER*

```
\usepackage{textcase}
\MakeTextUppercase{Diese beiden schlagen immer
  \textbf{fehl $a+b=c$}
  \emph{\NoCaseChange{, leider}}}
```

Bsp. 3-1-13

Im obigen Fall lässt sich diese Stolperfalle umgehen, indem man die Formel aus dem Argument von \textbf herausnimmt und \emph in das Argument von \NoCaseChange einsetzt. In anderen Fällen ist diese Art der Korrektur möglicherweise nicht machbar. Dann besteht die (etwas mühselige) Lösung darin, den problematischen Teil in einem gesonderten Makro zu verbergen und dieses beim Wechsel der Groß- und Kleinschreibung mit \protect zu schützen. Diese Methode funktioniert auch für die LaTeX-Standardbefehle, wie das nächste Beispiel zeigt.

Bsp. 3-1-14

DIES **FUNKTIONIERT** $a + b = c$ IMMER

```
\newcommand\mymath{$a+b=c$}
\MakeUppercase{Dies
  \textbf{funktioniert \protect\mymath} immer}
```

Einige Klassen und Pakete verwenden intern \MakeUppercase, zum Beispiel in Kopfzeilen. Wenn stattdessen \MakeTextUppercase verwendet werden soll, muss das textcase-Paket mit der Option overload geladen werden. Diese Option ersetzt die LaTeX-Standardbefehle durch die vom Paket definierten Varianten.

3.1.6 ulem – Betonen durch Unterstreichen

LaTeX ermutigt dazu, einzelne Elemente nicht durch explizite Befehle für den Fontwechsel, wie \bfseries oder \itshape, zu betonen, sondern eher mit dem Befehl \emph oder der Deklaration \em hervorzuheben. Das Paket ulem (von Donald Arseneau) definiert den Befehl \emph neu, so dass er anstelle von kursivem Text unterstrichenen Text erzeugt. Innerhalb des unterstrichenen Textes können Zeilenumbrüche und sogar eine einfache Silbentrennung vorgenommen werden.

Jedes Wort wird in einer unterstrichenen Box gesetzt, wodurch die automatische Silbentrennung ausgeschaltet wird. Weiche Trennstriche (\-) können aber trotzdem explizit eingesetzt werden. Die Unterstreichung setzt sich zwischen den Worten fort und wird wie normale Leerzeichen gedehnt. Da Leerzeichen normalerweise Wörter begrenzen, können bei syntaktischen Leerzeichen (z.B. „2.3 pt“) Schwierigkeiten auftreten. Es wird einiger Aufwand betrieben, um solche Leerzeichen zu handhaben. Wenn Probleme auftreten, kann man versuchen, den störenden Befehl in Klammern zu setzen, da alle Einträge in Klammern in eine \mbox gesetzt werden. Dadurch unterdrücken Klammern die Dehnung und den Umbruch des Textes, den sie einschließen. Konstruktionen, die in mehreren Abstufungen hervorgehoben werden sollen, werden von diesem Paket nicht immer korrekt behandelt. (Man beachte den Aufwand, der im folgenden Beispiel betrieben wurde, um korrekte Wortzwischenräume zu erhalten: Jedes einzelne weiter hervorgehobene Wort wurde mit einem eigenen \emph-Befehl versehen.)

Bsp. 3-1-15

Nein, ich habe nicht in dem Film Die Verfolgung und Ermordung des Jean-Paul Marat, vorgeführt von den Insassen des Asylum of Charenton unter Leitung des Marquis de Sade, mitgespielt! Aber ich habe ihn gesehen.

```
\usepackage{ulem}
Nein, ich habe \emph{nicht} in dem Film
\emph{\emph{Die} \emph{Verfolgung} \emph{und}
\emph{Ermordung} \emph{des} \emph{Jean-Paul}
\emph{Marat}, vorgeführt von den Insassen des Asylum
of Charenton unter Leitung des Marquis de~Sade,}
mitgespielt! Aber ich \emph{habe} ihn gesehen.
```

Alternativ kann eine Unterstreichung explizit über den Befehl \uline erzeugt werden. Außerdem gibt es einige weitere Varianten, die in der Satztechnik üblich sind. Diese sind im nächsten Beispiel zu sehen.

Bsp. 3-1-16

```
\usepackage{ulem}
Doppelt \uuline{unterstrichen}, mit
Wellenlinie \uwave{unterschlängelt}, einfach
(\sout{durchgestrichen}) oder vollständig
\xout{ausge-x-t}.
```

Doppelt unterstrichen, mit Wellenlinie unterschlängelt, einfach (~~durchgestrichen~~) oder vollständig ~~*ausge-x-t*~~.

Die Neudefinition von \emph kann mit \normalem und \ULforem ein- und ausgeschaltet werden. Alternativ kann das Paket mit der Option normalem geladen werden, wodurch diese Neudefinition unterdrückt wird. Eine weitere Paketoption ist UWforbf, die, wo immer möglich, \textbf und \bfseries durch \uwave ersetzt.

Die Position der Linie, die durch \uline erzeugt wird, kann explizit durch Angabe eines Wertes für die Länge \ULdepth gesetzt werden. Der Standardwert ist fontabhängig und wird durch den ansonsten unsinnigen (da viel zu großen) Wert \maxdimen ausgedrückt. Auf ähnliche Weise kann die Dicke der Linie über den Befehl \ULthickness gesteuert werden; er muss aus historischen Gründen mithilfe von \renewcommand geändert werden.

3.1.7 soul – Sperren oder Schafe stehlen

Frederic Goudy soll gesagt haben „Jeder, der Frakturschrift sperrt, stiehlt auch Schafe.“ Egal, ob die Aussage wahr oder erfunden ist – das Thema „Sperren“ ruft heftige Diskussionen unter Typographen hervor und wird in den meisten Fällen als Unart betrachtet, weil es den „Grauwert“ des Textes ändert und damit den Lesefluss stört. Trotzdem gibt es durchaus Gründe für eine Sperrung. Layoutschriften benötigen beispielsweise häufig einen lockeren Satz und in den meisten Schriftarten wird großgeschriebener Text auf diese Weise aufgewertet. Manchmal wird Sperrung auch zur Betonung eingesetzt, dabei tritt das Grauwertproblem jedoch besonders deutlich hervor.

TEX bietet nur wenige Möglichkeiten zum Sperren von Text. Theoretisch ist es am besten, speziell dafür vorgesehene Zeichensätze zu verwenden, anstatt zu versuchen, das Problem mithilfe eines Makropaketes zu lösen. Da dies jedoch voraussetzt, dass ein solcher Zeichensatz verfügbar ist, müssen sich die meisten Benutzer anders behelfen. In der Praxis ist daher eine makrobasierte Lösung viel einfacher umzusetzen, auch wenn dies bedeutet, dass man einige Beschränkungen hinnehmen muss. Einige Informationen über den Lösungsansatz mit Fonts sind in der Dokumentation des fontinst-Paketes [74,75] aufgeführt.

Das soul-Paket von Melchior Franz bietet Möglichkeiten zur Sperrung und zur Unterstreichung, erhält dabei aber die TEX-Funktion der automatischen Silbentrennung. Diese Funktion ist im ulem-Paket nicht verfügbar. Bei soul wird der Text Zeichen für Zeichen syntaktisch analysiert, was zu einer Reihe von Eigenheiten und Einschränkungen führt. Benutzer, die nur ein paar Wörter unterstreichen möchten und keine automatische Silbentrennung benötigen, sind daher besser mit ulem bedient, das bei der Eingabe weniger pingelig ist.

\caps{*text*}	\hl{*text*}	\so{*text*}	\st{*text*}	\ul{*text*}

Die Verwendung der fünf wichtigsten Anwenderbefehle des soul-Paketes zeigt das nächste Beispiel. In den Fällen, in denen es der Silbentrennungsalgorithmus von TEX nicht schafft, die richtige Trennstelle zu finden, kann man wie gewohnt die Stelle mit dem Befehl \- markieren. Wenn das color-Paket geladen ist, funktioniert \hl wie ein Textmarker, wobei er standardmäßig den Hintergrund gelb einfärbt; ansonsten verhält er sich wie \ul und unterstreicht sein Argument.

Bsp. 3-1-17

Mit dem soul-Paket können W ö r t e r und P h r a s e n g e s p e r r t werden. Großbuchstaben werden mit einem anderen Befehl GESPERRT. Auch zum Unterstreichen, ~~Durchstreichen~~ und Hervorheben gibt es eigene Befehle.

```
\usepackage{soul,color}
Mit dem \texttt{soul}-Paket können \so{Wörter
und Phrasen gesperrt} werden. Großbuchstaben
werden mit einem anderen Befehl \caps{GESPERRT}.
Auch zum \ul{Unterstreichen},
\st{Durchstreichen} und \hl{Hervorheben} gibt
es eigene Befehle.
```

Normalerweise interpretiert das soul-Paket den Text im Argument von \so, \st usw. Buchstabe für Buchstabe. Wenn Buchstaben jedoch durch mehr als ein Zeichen dargestellt werden (z.B. akzentuierte Zeichen), kann diese Vorgehensweise mit hässlichen TEX-Fehlermeldungen fehlschlagen. Glücklicherweise kennt das Paket bereits alle üblichen Akzentbefehle, so dass diese richtig behandelt werden. Alle anderen, z.B. diejenigen, die vom Paket textcomp bereitgestellt werden, können mithilfe einer \soulaccent-Deklaration dem soul-Paket bekannt gemacht werden. Eine andere Möglichkeit ist, die Zeichen einzuklammern.

Bsp. 3-1-18

ä ù Õ X̀ Y̆

```
\usepackage{soul} \usepackage{textcomp}
\soulaccent{\capitalgrave}
\Huge \st{\"a \`u \~O \capitalgrave X {\capitalbreve Y}}
```

Das soul-Paket weiß bereits, dass Anführungsstriche, Gedankenstriche und Geviertstriche aus mehreren Zeichen bestehen, und behandelt diese richtig. Andere syntaktische Ligaturen, wie das spanische Ausrufezeichen, müssen zur korrekten Bearbeitung in Klammern eingefasst werden.

Bsp. 3-1-19

„A l s o“, sagte er.
¡HOLA – MEIN **FREUND**!

```
\usepackage{soul}
\so{"`Also"'}, sagte er. \\
\caps{{!`}Hola -- mein \textbf{Freund}!}
```

Das soul-Paket erkennt auch Formeln, solange diese in $-Zeichen eingeschlossen sind (die Form \(...\) wird nicht unterstützt), sowie alle Fontbefehle mit Argument, wie \textbf. Ein selbstdefinierter Fontbefehl oder solche, die von einem Paket bereitgestellt werden, müssen allerdings mit der Deklaration \soulregister im soul-Paket registriert werden. Bei dieser Deklaration wird der Fontbefehl als erstes Argument und die Anzahl der Argumente (d.h. 0 oder 1) für diesen Befehl als zweites Argument angegeben. Innerhalb des soul-Paketes bietet keiner der Fontbefehle Kursivkorrekturen. Wenn eine solche erforderlich ist, muss sie manuell per \/ durchgeführt werden.

```
\newcommand\textsfbf[1]{\textsf{\bfseries#1}}
\usepackage{soul} \soulregister{\textsfbf}{1}
\so{Hier sehen wir \textsfbf{soul} in \emph{Aktion}:
    $x\neq y$ OK?}
```

Hier sehen wir **soul** in *Aktion*: $x \neq y$ OK?

Bsp. 3-1-20

Bei genauerer Betrachtung sieht man, dass die Fontbefehle eine direkt vorangehende und nachfolgende Sperrung unterdrücken, wie zwischen „Aktion“ und dem Doppelpunkt. Dies lässt sich korrigieren, indem man den Befehl \> hinzufügt, der einen größeren Abstand erzeugt.

```
\usepackage{soul}
\so{l\textbf{ei}dend  vs. l\>\textbf{ei}\>dend}
```

l**ei**d e n d v s . l **ei** d e n d

Bsp. 3-1-21

Eingeklammerter Text wird während der syntaktischen Analyse als ein Objekt betrachtet und daher nicht gesperrt. Dies ist ein sehr willkommener Effekt, wenn bestimmte Ligaturen innerhalb eines gesperrten Textes zusammengehalten werden sollen. Diese Methode funktioniert jedoch nur, wenn der Text innerhalb der Klammern keine Trennstellen zur Silbentrennung enthält. Wenn Trennstellen vorhanden sind, wird die Paketfehlermeldung „Reconstruction failed“ ausgegeben. Um die Trennstellen zu verbergen, muss der Text in eine \mbox eingefügt werden, wie die zweite Textzeile des nächsten Beispiels zeigt. (TEX würde hier in „Es-cher“ trennen – mitten im „sch“, das stets zusammengehalten werden sollte.) Dieser Effekt lässt sich auch durch \soulomit erreichen, aber dann lässt sich der Text nur kompilieren, wenn das soul-Paket geladen ist.

```
\usepackage{soul,yfonts}
\usepackage[ansinew]{inputenc}
\textfrak{\so{S{ch}u{tz}vorri{ch}tung}} \par
\so{Gödel, E\mbox{sch}er, Bach}         \par
\ul{Da lässt sich tre\soulomit{ffl}ich streiten!}
```

Schutzvorrichtung
Gödel, E s c h e r, B a c h
Da lässt sich trefflich streiten!

Bsp. 3-1-22

Eine der größten Einschränkungen der obigen Befehle ist, dass sie nicht verschachtelt werden können. Jegliche Versuche, soul-Befehle zu verschachteln, führen unweigerlich zu systemnahen TEX-Fehlern. Wenn wirklich eine Verschachtelung erforderlich ist, muss der innere Teil in einer Box platziert werden. Dies bedeutet aber, dass er nicht mehr am Zeilenende umbrochen werden kann.

```
\usepackage{soul}  \newsavebox\soulbox
\sbox\soulbox{\so{ eine Qual }}
\ul{Dies ist \mbox{\usebox{\soulbox}} für uns alle!}
```

Dies ist e i n e Q u a l für uns alle!

Bsp. 3-1-23

Einige andere Befehle zeigen innerhalb des Argumentes von \so und Co. ein besonderes Verhalten. Wie oben zu sehen war, kann eine Sperrung an bestimmten Punkten mit \< unterdrückt oder mit \> erzwungen werden. Wie bei LATEX üblich, erzeugt ~ ein geschütztes Leerzeichen. Der Befehl \\ wird zwar unterstützt, allerdings nur in seiner Grundform – d.h. ohne Stern und ohne optionales Argument. Mit \linebreak kann an bestimmten Punkten eine Zeile umbrochen werden, doch auch hier wird das optionale Argument nicht unterstützt. Andere LATEX-Befehle werden das Paket wahrscheinlich eher sprengen

– hier hilft nur auszuprobieren, um herauszufinden, was sicher funktioniert und was zur Katastrophe führt. Das nächste Beispiel zeigt ein paar Anwendungen für diese besonderen Fälle.

Bsp. 3-1-24

„A l s o“, s a g t e e r. D a n n w o l l e n w i r m a l e i n e k u r z e u n d e i n e g e s p e r r t e Z e i l e p r o d u z i e r e n, O K?

```
\usepackage{soul}
\so{{"‘}\<Also\<{"’}, sagte er.
    Dann wollen wir mal eine kurze und \\
    eine gesperrte Zeile\linebreak
    produzieren, OK?}
```

`\sodef{`*befehl*`}{`*font*`}{`*zwischenraum*`}{`*wortabstand*`}{`*endabstand*`}`

Mit der Deklaration `\sodef` kann man eigene Befehle zur Sperrung definieren. Außerdem bietet sie die Möglichkeit, die Standardwerte für den Befehl `\so` zu überschreiben.

Dieser Sperrungs-Algorithmus fügt zwischen Zeichen einen bestimmten *zwischenraum*, zwischen Wörtern einen *wortabstand* und am Anfang und Ende eines Textabschnitts einen *endabstand* ein. Letzterer wird nur eingefügt, wenn es an dieser Stelle passend ist. Die Standardwerte für diese Abstände sind auf den Satz von Texten in Frakturschriften ausgerichtet, aber sie lassen sich recht leicht mithilfe der Deklaration `\sodef` den eigenen Anforderungen anpassen. Im Argument *font* können Fontattribute angegeben werden; in den meisten Fällen bleibt es leer. Es empfiehlt sich, in den anderen Argumenten keine expliziten Maße anzugeben, sondern stattdessen auf `em`-Werte zurückzugreifen, damit die Definition vom aktuellen Font und seiner Größe abhängig wird.

Bsp. 3-1-25

Hier **w e r d e n e i n i g e W ö r t e r** betont.

```
\usepackage{soul}
\sodef\sobf{\bfseries}{.3em}{1em plus .1em}
                      {1.3em plus.1em minus.2em}
Hier \sobf{werden einige Wörter} betont.
```

Während bei `\so` oder jedem neuen, über `\sodef` definierten Befehl die gespeicherte Definition ausgelesen und ausgeführt wird, funktioniert der Befehl `\caps` anders. Er untersucht den aktuellen Font und versucht, diesen (oder einen ihm möglichst ähnlichen) in einer internen Datenbank zu finden. Danach wendet er die dort gespeicherten Werte zur Sperrung an. Die Datenbank ist erweiterbar. Mit der Deklaration `\capsdef` können Werte für einzelne Fonts oder Fontgruppen bereitgestellt werden. Auf diese Weise kann die Sperrung feinabgestimmt werden – zum Beispiel im Überschriftentext. Es ist sogar möglich, mehrere Datenbanken anzulegen und sie während der Bearbeitung im Dokument auszutauschen.

`\capsdef{`*muster*`}{`*font*`}{`*zwischenraum*`}{`*wortabstand*`}{`*endabstand*`}`

Bis auf das erste Argument, das sich von den anderen stark unterscheidet, sind die anderen Argumente von `\capsdef` mit jenen von `\sodef` identisch. Das erste Argument *muster* definiert den Font oder die Fonts, auf welche die aktuelle Deklaration angewendet werden soll.

Seine Syntax ist *schriftkodierung*, *-familie*, *-serie*, *-form* und *-grad*, getrennt durch Schrägstriche, wobei die Namenskonvention des NFSS verwendet wird. Wenn ein Wert leer ist, bedeutet dies, dass es für dieses Attribut keine Einschränkung gibt, d.h. bei `////` werden alle Fonts gesperrt, bei `/ptm///10` werden alle Times Fonts im Schriftgrad 10 Punkt gesperrt, und bei `OT1/cmr/m/n/` wird Computer Modern (`cmr`), Serie Medium (`m`), normale Form (`n`), kodiert in `OT1` in jeglichem Schriftgrad gesperrt. Darüber hinaus ist es möglich, Größenbereiche anzugeben. `5-14` bedeutet beispielsweise $5\,\text{pt} \leq \textit{grad} < 14\,\text{pt}$ und `14-` heißt, dass alle Grade von 14 pt und größer gesperrt werden. Einzelheiten über die Fontnamenskonventionen von NFSS sind in Kapitel 7 aufgeführt.

Wie bei `\sodef` ist das Argument *font* auch in den meisten Deklarationen leer. In einigen Fällen kann es sinnvoll sein, hier `\scshape` zu verwenden, zum Beispiel wenn die Schriftform vor der Sperrung in Kapitälchen umgewandelt werden soll.

`\caps` verwendet immer den ersten passenden Eintrag in seiner Datenbank, daher ist die Reihenfolge der `\capsdef`-Deklarationen wichtig. Nachfolgende Deklarationen werden zuerst untersucht, so dass bestehende Deklarationen überschrieben oder erweitert werden können.

EINE BEISPIEL-ÜBERSCHRIFT

Hier ist die Deklaration `\capsdef` anzuwenden, weil in der Definition für die Überschrift eine serifenlose Schrift spezifiziert ist und die Beispiele in diesem Buch in Times und Helvetica (phv) gesetzt sind.

```
\usepackage{titlesec,soul}
\newcommand\allcaps[1]{\MakeUppercase{\caps{#1}}}
\titleformat{\section}[block]{\centering\sffamily}
                             {\thesection.}{.5em}{\allcaps}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\section*{Eine Beispielüberschrift}
Hier ist die Deklaration \verb=\capsdef= anzuwenden, weil
in der Definition für die Überschrift eine serifenlose
Schrift spezifiziert ist und die Beispiele in diesem Buch
in Times und Helvetica (\texttt{phv}) gesetzt sind.
```

Bsp. 3-1-26

Das vorangegangene Beispiel enthielt eine interessante Kombination von `\caps` und `\MakeUppercase`: Der Befehl `\allcaps` wandelt den Text des Argumentes in Großbuchstaben um und sperrt ihn dann mit dem Befehl `\caps`.

`\capssave{`*name*`}`	`\capsselect{`*name*`}`	`\capsreset`

Benutzerdefinierte Sperrung für verschiedene Fälle

Mit `\capsreset` wird die Datenbank wieder in ihren ursprünglichen Zustand zurückgesetzt, der nur generische Standardwerte enthält. Mit `\capsdef` können anschließend neue Einträge hinzugefügt werden. Der aktuelle Status der `\caps`-Datenbank kann mit `\capssave` unter einem *name*n abgespeichert werden. Dieser Status kann über `\capsselect` später wieder hergestellt werden. Wenn beim Laden des Paketes die Option `capsdefault` verwendet wird, werden alle Verwendungen von `\caps`, zu denen es keine entsprechende Deklaration gibt, durch Unterstreichung des Textes markiert.

Bsp. 3-1-27

EINE BEISPIELÜBERSCHRIFT

Man beachte die unterschiedliche Laufweite der Sperrung im Überschriftentext und im LAUFENDEN TEXT. Für Times gibt es hier keine Definition, so dass die STANDARDEINSTELLUNG wirksam ist.

```
\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display}    \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}
            {\thesection.}{.5em}{\capsselect{display}\caps}
\section*{Eine Beispielüberschrift}
Man beachte die unterschiedliche Laufweite der Sperrung im
Überschriftentext und im \textsf{\caps{laufenden Text}}.
Für Times gibt es hier keine Definition, so dass die
\caps{Standardeinstellung} wirksam ist.
```

Benutzerdefinierte Unterstreichung

Mit `\setul` bzw. `\setuldepth` kann man die Position und die Dicke der Linie, die vom Befehl `\ul` erzeugt wird, nach eigenen Anforderungen definieren. Der Befehl `\setul` hat zwei Argumente: Im ersten wird die Position der Linie in Bezug auf die Grundlinie und im zweiten die Dicke der Linie angegeben. Alternativ kann man `\setuldepth` verwenden, um anzugeben, dass die Linie unter dem im Argument enthaltenen Text angeordnet werden soll. Mit `\resetul` können die Standardeinstellungen des Paketes wiederhergestellt werden.

Bsp. 3-1-28

Hier testen wir ~~eine Reihe von~~ verschiedenen Einstellungen. Zurück zur Normalität!

```
\usepackage{soul}
\setul{0pt}{.4pt}   \ul{Hier testen wir}              \par
\setul{-.6ex}{.3ex} \ul{eine Reihe von}               \par
\setuldepth{g}      \ul{verschiedenen Einstellungen.}\par
\resetul            \ul{Zurück zur Normalität!}
```

Sowohl `\ul` als auch `\st` verwenden standardmäßig eine schwarze Linie. Wenn zusätzlich das Paket color geladen wird, können stattdessen auch farbige Linien eingesetzt werden und, wenn gewünscht, kann die Hervorhebungsfarbe von `\hl` geändert werden, wie das Beispiel unten zeigt:

Bsp. 3-1-29

Linien können schwarz oder blau sein.

```
\usepackage{soul,color}
\sethlcolor{green} \setulcolor{blue} \setstcolor{red}
Linien \hl{können} \st{schwarz} oder \ul{blau} sein.
```

3.1.8 url – URLs, Pfadnamen und Ähnliches

E-Mail-Adressen, URLs, Pfad- oder Verzeichnisnamen und ähnliche Objekte zu formatieren ist in der Regel nicht ganz so einfach. Zum Einen enthalten sie oft Zeichen, die eine besondere Bedeutung für LaTeX haben, wie `~`, `#`, `&`, `{` oder `}`. Zum Anderen sollten sie möglichst nicht umbrochen werden oder, wenn nicht vermeidbar, zumindest mit Bedacht umbrochen werden. Es ist beispielsweise nicht empfehlenswert, an einem Bindestrich zu umbrechen, denn damit bleibt unklar, ob der Bindestrich aufgrund des Zeilenumbruchs eingefügt wurde (wie bei normalen Wörtern) oder Teil des Ausdrucks ist. Aus demselben Grund sollte man auch keinen Umbruch an einer Leerstelle vornehmen. Als kleine Hilfe entwickelte Donald Arseneau das Paket url, das versucht, die meisten dieser Probleme zu lösen.

\url{*text*}	\url!*text*!	\path{*text*}	\path=*text*=

Der Grundbefehl dieses Paketes ist `\url`, den es in zwei Syntaxvarianten gibt: Das Argument *text* kann entweder in Klammern eingefasst werden (in diesem Fall muss der *text* immer genauso viele öffnende wie schließende Klammern enthalten) oder kann, wie `\verb`, durch willkürliche Zeichen (die nicht im *text* auftauchen) auf beiden Seiten begrenzt werden. (In der Syntaxbox oben werden `!` und `=` verwendet, aber das sind nur Beispiele.) Im zweiten Fall darf es auch eine ungleiche Anzahl an öffnenden und schließenden Klammern im Argument geben.

Der Befehl `\path` funktioniert genauso, außer dass er stets Typewriter Fonts (`\ttfamily`) verwendet, während `\url` individuell angepasst werden kann. Das Argument beider Befehle wird mehr oder weniger wortwörtlich ausgegeben. `\url{~}` erzeugt z.B. eine Tilde. Leerzeichen werden standardmäßig ignoriert, wie das folgende Beispiel zeigt.

Bsp. 3-1-30

```
\usepackage{url}
Die Webadresse des \LaTeX{}-Projektes lautet
\url{http://www . latex-project . org} und mein
Homeverzeichnis ist (manchmal) \path+~frank+.
```

Die Webadresse des LaTeX-Projektes lautet `http://www.latex-project.org` und mein Homeverzeichnis ist (manchmal) `~frank`.

Zeilenumbrüche können an bestimmten Sonderzeichen durchgeführt werden (standardmäßig nicht zwischen Buchstaben oder Bindestrichen), aber Bindestriche werden von den Befehlen an der Umbruchstelle nicht hinzugefügt. Wenn der *text* die Zeichen `%` oder `#` enthält oder auf `\` endet, kann er nicht im Argument eines anderen Befehls verwendet werden, ansonsten generiert dieser Befehl einen Fehler (genau wie der Befehl `\verb`). Eine weitere Zeichenfolge, die im Argument eines anderen Befehls nicht richtig funktioniert, sind zwei aufeinander folgende `^`-Zeichen. Dieser Fall ist sogar noch schlimmer, denn hierbei wird möglicherweise nicht mal eine Fehlermeldung ausgegeben, sondern einfach eine falsche Ausgabe erzeugt.[1] Dies zeigt auch das nächste Beispiel.

Bsp. 3-1-31

```
\usepackage{url}
\url{^frank}  und \mbox{\url{^frank}}  (OK)\par
\url{^^frank} aber \mbox{\url{^^frank}} (falsch)
```

`^frank` und `^frank` (OK)
`^^frank` aber `&rank` (falsch)

Selbst wenn der *text* keine kritischen Sonderzeichen enthält, ist es nie zulässig, solch einen Befehl in einem bewegten Argument zu verwenden – z.B. im Argument des `\section`-Befehls. Hier wird die Fehlermeldung

```
! Undefined control sequence.
\Url Error ->\url used in a moving argument.
```

ausgegeben, gefolgt von vielen seltsamen Fehlern. Selbst die Verwendung von `\protect` hilft in diesem Fall nicht. Was kann man also machen, wenn man einen Pfadnamen oder eine URL an einer solchen Stellen zitieren muss? Wenn man sorgfältig arbeitet und ausschließlich „sichere“ Zeichen im *text* verwendet, kann man die Befehle beim Laden des Paketes mithilfe der Option `allowmove` für die Verwendung in bewegten Argumenten freigeben. Dies ist

[1] Dies hängt vom nachfolgenden Buchstaben ab. Ein großes `F` erzeugt z.B. einen Fehler.

jedoch nicht wirklich hilfreich, wenn ein Zeichen wie „#“ verwendet werden soll. In diesem Fall muss die Information zunächst mit `\urldef` gespeichert werden, bevor sie später verwendet werden kann.

`\urldef{`*befehl*`}{`*url-befehl*`}{`*text*`}` `\urldef{`*befehl*`}{`*url-befehl*`}=`*text*`=`

Die Deklaration `\urldef` definiert einen neuen Befehl *befehl*, der den *url-befehl* (dies kann `\url`, `\path` oder ein neu definierter Befehl sein – siehe unten) so auf den *text* anwendet, dass er an jeder Stelle (einschließlich bewegter Argumente) eingesetzt werden kann. Der *url-befehl* wird an dieser Stelle nicht ausgeführt, was bedeutet, dass sich Änderungen des Schriftstils auf die Formatierung auswirken (siehe Beispiel 3-1-33). Technisch gesprochen werden hierbei die `\catcodes` der Zeichen im *text* während der Deklaration eingefroren, damit sie an Stellen wie in Argumenten nicht fehlinterpretiert werden können.

Bsp. 3-1-32

1 ^^frank~#$\ geht?

Ja, es geht – im Gegensatz zum vorherigen Beispiel.

```
\usepackage{url}
\urldef\test\path{^^frank~#$\}
\section{\test{} geht?}
Ja, es geht -- im Gegensatz zum vorherigen
Beispiel.
```

`\urlstyle{`*stil*`}`

Die Möglichkeit der Stiländerung wurde bereits erwähnt. Für diese Aufgabe bietet das url-Paket den Befehl `\urlstyle` an, der ein obligatorisches Argument hat: einen bestimmten *stil*. Vordefinierte Stile sind `rm`, `sf`, `tt` und `same`. Die ersten drei Werte selektieren die korrespondierende Schriftfamilie, während `same` die aktuelle Schrift verwendet und nur die Zeilenumbruchsregeln ändert.

Der Befehl `\url` verwendet den aktuellen *stil* (voreingestellt `tt`, d.h. Typewriter Font), während `\path` intern immer auf den Stil `tt` umschaltet. Im folgenden Beispiel wird eine URL, die in `\lproject` gespeichert ist, mehrere Male mit verschiedenen Stilen formatiert. Dieses Beispiel sieht immer noch fürchterlich aus, aber wie hätte es wohl ausgesehen, wenn die URL in diesem schmalen Satzspiegel nicht umbrochen werden dürfte?

Bsp. 3-1-33

Zapf Chancery! http://www.latex-project.org *(Standardeinstellung)* http://www.latex-project.org *(CM Roman)* http://www.latex-project.org *(CM Sans Serif)* http://www.latex-project.org *(CM Typewriter)* *http://www.latex-project.org (Zapf Chancery)*

```
\hyphenation{Stan-dard-ein-stellung}

\usepackage[hyphens]{url}
\urldef\lproject\url{http://www.latex-project.org}
\fontfamily{pzc}\selectfont Zapf Chancery!
            \lproject\   (Standardeinstellung) \quad
\urlstyle{rm}\lproject\   (CM Roman)       \quad
\urlstyle{sf}\lproject\   (CM Sans Serif)  \quad
\urlstyle{tt}\lproject\   (CM Typewriter)  \quad
\urlstyle{same}\lproject\ (Zapf Chancery)
```

Bei genauer Betrachtung kann man sehen, dass im obigen Beispiel die Option `hyphens` verwendet wurde. Diese Option ermöglicht einen expliziten

Zeilenumbruch an Bindestrichen, welches bei \url-ähnlichen Befehlen normalerweise deaktiviert ist. Ohne diese Option sind Umbrüche nur an den Punkten, nach dem Doppelpunkt oder nach „//“ zugelassen.

Leerzeichen im Argument

Wie bereits erwähnt, werden Leerzeichen im *text* standardmäßig ignoriert. Wenn dies nicht erwünscht ist, kann man die Option obeyspaces verwenden. Sie kann jedoch dazu führen, dass falsche Leerzeichen eingefügt werden, wenn der Befehl im Argument eines anderen Befehls verwendet wird und der *text* irgendwelche „\“-Zeichen enthält. In diesem Fall lässt sich das Problem mit \urldef lösen. Zeilenumbrüche und Leerzeichen werden nur zugelassen, wenn zusätzlich die Option spaces verwendet wird.

Das Paket erkennt automatisch, welche Schriftkodierung gerade verwendet wird. Bei T1-kodierten Schriften verwendet es auch die in dieser Kodierung zusätzlich verfügbaren Zeichen, um das Gesamtergebnis zu verbessern.

Text links oder rechts anhängen

Das Paket stellt die beiden Parameter \UrlLeft und \UrlRight bereit, die standardmäßig keine Auswirkung haben. Sie können aber so umdefiniert werden, dass sie Material links oder rechts vom *text* setzen. Das Material wird auf dieselbe Weise formatiert wie der *text*. Leerzeichen werden ignoriert, es sei denn, dass \␣ verwendet oder obeyspaces als Option angegeben wird. Wenn die Befehle auf der obersten Ebene neu definiert werden, wirken sie sich auf jeden \url-ähnlichen Befehl aus. Eine Möglichkeit zur Beschränkung ihres Gültigkeitsbereiches ist in Beispiel 3-1-34 dargestellt.

\DeclareUrlCommand{*befehl*}{*stil-information*}

Definition URL-ähnlicher Befehle

Manchmal ist es hilfreich, eigene Befehle zu definieren, die ähnlich wie \url oder \path funktionieren, aber ihre eigenen Schriften usw. verwenden. Mit dem Befehl \DeclareUrlCommand kann ein neuer \url-ähnlicher Befehl erzeugt oder ein bestehender modifiziert werden. Er hat zwei Argumente: den Befehl, der definiert oder geändert werden soll, und die *stil-information* (z.B. \urlstyle).

Im nächsten Beispiel wird \email so definiert, dass E-Mail-Adressen per \UrlLeft im rm-Stil gesetzt werden, und zwar beginnend mit der Zeichenfolge „E-Mail: “. Das Beispiel zeigt deutlich, dass der Definitionsbereich dieser Neudefinition auf den Befehl \email begrenzt ist. Bei genauerer Betrachtung entdeckt man, dass ein Leerzeichen in \UrlLeft (wie in der Definition auf der obersten Ebene) ohne Wirkung bleibt, während \␣ das gewünschte Ergebnis erzeugt.

```
\usepackage{url}
\renewcommand\UrlLeft{<url: }
\renewcommand\UrlRight{>}
\DeclareUrlCommand\email{\urlstyle{rm}%
   \renewcommand\UrlLeft{E-Mail:\ }%
   \renewcommand\UrlRight{}}
\url{http://www.latex-project.org}
\email{frank.mittelbach@latex-project.org}
\\      \path{$HOME/figures} uups, falsch!
```

<url:http://www.latex-project.org>
E-Mail: frank.mittelbach@latex-project.org
<url:$HOME/figures> uups, falsch!

Bsp. 3-1-34

Das url-Paket bietet eine Reihe weiterer Parameter, die den Zeilenumbruch beeinflussen, unter anderem \UrlBreaks, \UrlBigBreaks und

EUR	Europa	GRD	Griechenland
ATS	Österreich	IEP	Irland
BEF	Belgien	ITL	Italien
DEM	Deutschland	LUF	Luxemburg
ESP	Spanien	NLG	Niederlande
FIM	Finnland	PTE	Portugal
FRF	Frankreich		

Tabelle 3.1: ISO-Währungscodes des *Euros* und der 12 *Euro-Zonen*-Länder

`\UrlNoBreaks`. Diese Parameter können im Argument *stil-information* von `\DeclareUrlCommand` so umdefiniert werden, dass neue oder spezielle Konventionen verwendet werden. Einzelheiten hierzu enthält die Paketdokumentation, die am Ende der Datei `url.sty` angefügt ist.

3.1.9 euro – Konvertieren und Formatieren von Währungen

Um die Umrechnung zwischen länderspezifischen Währungen und dem Euro zu vereinfachen, entwickelte Melchior Franz das Paket euro. Dieses Paket berechnet beliebige Wechselkurse auf Basis des Euros. Die Berechnungen werden mithilfe des fp-Paketes von Michael Mehlich mit hoher Genauigkeit durchgeführt. Die Formatierung kann für jede Währung individuell verändert werden, so dass dieses Paket für alle Arten von Anwendungen eingesetzt werden kann, in denen Währungen vorkommen, ungeachtet dessen, ob Umrechnungen erforderlich sind oder nicht.

`\EURO{`*ausgangs-währung*`}[`*ziel-währung*`]{`*betrag*`}`

Der Hauptbefehl `\EURO` wandelt einen *betrag* in der *ausgangs-währung* in die *ziel-währung* um oder, wenn dieses optionale Argument fehlt, in Euro. Die Argumente *ausgangs-währung* und *ziel-währung* werden in ISO-Währungscodes angegeben, so wie sie in Tabelle 3.1 aufgelistet sind. Bei der Eingabe des *betrag*s werden die Dezimalstellen durch einen Punkt abgetrennt, selbst wenn die formatierte Zahl normalerweise anders dargestellt würde.

Bei der Standardeinstellung werden der *betrag* in der *ausgangs-währung* und dahinter der entsprechende Wert in der *ziel-währung* in Klammern angezeigt.

Bsp. 3-1-35

```
\usepackage{euro}
\EURO{DEM}[FRF]{7} \ \EURO{FRF}[DEM]{23.48}
\\    \EURO{EUR}[DEM]{1.00} \ \EURO{DEM}{2}
```

7 DM (23,48 FRF) 23,48 FRF (7 DM)
1 Euro (1,96 DM) 2 DM (1,02 Euro)

Die Paketoptionen

Das Paket bietet eine Anzahl von Optionen, die das allgemeine Layout der Ausgabe verändern können (sofern sie nicht von spezifischeren Formatierungsdeklarationen überschrieben werden, siehe unten). Mit `eco` wird dem Betrag nur der ISO-Code vorangestellt, individuelle Symbole werden nicht verwendet; bei `dots` wird zwischen Dreiergruppen von Ziffern jeweils ein Punkt eingefügt (standardmäßig wird ein kleiner Leerraum verwendet).

Standardmäßig werden ganze Beträge unverändert, d.h. ohne Dezimalpunkt und Dezimalnullen, ausgegeben. Wenn die Option `table` definiert ist, ändert sich diese Darstellung global, und es wird entweder ein — (Option `emdash`, Voreinstellung), ein – (Option `endash`) oder die richtige Anzahl von Nullen (Option `zeros`) ausgegeben.

```
\usepackage[eco,table,endash]{euro}
\EURO{DEM}[FRF]{7} \ \EURO{FRF}[DEM]{23.48}
\\     \EURO{EUR}[DEM]{1.00} \ \EURO{DEM}{2}
```

DEM 7,– (FRF 23,48) FRF 23,48 (DEM 7,–)
EUR 1,– (DEM 1,96) DEM 2,– (EUR 1,02)

Bsp. 3-1-36

Die spezifischeren Ausgabedefinitionen, die später noch erörtert werden, können überall im Dokument stehen. Es empfiehlt sich jedoch, sie in der Präambel zusammenzuhalten oder sie sogar in die Datei `euro.cfg` einzutragen, die beim Laden des Paketes gelesen wird.

Die Formatierung der Währungszeichen kann mit der Deklaration `\EUROSYM` eingestellt werden; standardmäßig verwendet das Paket für die meisten Währungen die ISO-Codes. Im Beispiel wird die Darstellung für Lira und Euro geändert, indem die Währungssymbole aus dem Paket textcomp verwendet werden. Außerdem wird zur einfacheren Darstellung der riesigen Lira-Beträge `dots` verwendet.

```
\usepackage{textcomp}\usepackage[dots]{euro}
\EUROSYM{ITL}{\textlira}
\EUROSYM{EUR}{\texteuro}
\EURO{ITL}{10000}\quad \EURO{DEM}[ITL]{1000}
```

10.000 £ (5,16 €) 1.000 DM (989.999 £)

Bsp. 3-1-37

Das Paket kann jederzeit Währungen neuer Länder aufnehmen, die der Euro-Zone beitreten. Es beherrscht die Umrechnung aus und in alle Währungen, solange ihr Umrechnungskurs zum Euro bekannt ist. Zum Hinzufügen einer neuer Währung wird die Deklaration `\EUROADD` verwendet, die drei Argumente hat: den ISO-Währungscode, das Symbol oder den Text, der für diese Währung angezeigt wird, sowie den Umrechnungskurs zum Euro. Das nächste Beispiel zeigt die Darstellung des Britischen Pfunds. Die Abkürzung `\GBP` vereinfacht die Eingabe ein wenig.

```
\usepackage{eurosans,euro}
\EUROADD{GBP}{\textsterling}{0.6648} % 2005/06/21
\newcommand*\GBP{\EURO{GBP}} \EUROSYM{EUR}{\euro}
\noindent \GBP{14.9}\\ \GBP[FRF]{10}\\ \EURO{EUR}[GBP]{10}
```

14,90 £ (22,41 €)
10 £ (98,67 FRF)
10 € (6,65 £)

Bsp. 3-1-38

Die Umrechnungskurse für die einzelnen Währungen der Euro-Zonenländer sind fest (und im Paket vordefiniert). Bei anderen Währungen können sich die Kurse stündlich ändern, man sollte sich daher auf häufige Änderungen einstellen.

Das Paket ermöglicht es, die Darstellung mithilfe von `\EUROFORMAT`-Deklarationen individuell zu gestalten. Man kann entweder neue Standardwerte bereitstellen oder die Formatierung einzelner Währungen anpassen. Das erste Argument gibt an, welcher Teil der Formatierung verändert werden soll, während das zweite Argument die Formatierung beschreibt.

Das Format `main` legt fest, wie die Ausgangs- und Zielwährungen angeordnet werden sollen. Dabei wird mit den reservierten Schlüsselwörtern `\in` und `\out` auf die Ausgangs- bzw. Zielwährungen verwiesen. Im Beispiel unten wird in der ersten Zeile ein Format definiert, das den Standardeinstellungen sehr ähnlich ist. Die zweite Zeile zeigt das Ergebnis der Umrechnung und in der dritten Zeile ist überhaupt keine Umrechnung zu sehen (obwohl diese hinter den Kulissen durchaus stattgefunden hat). Letzteres ist nützlich, wenn man die Formatierungsfunktion für die Währungen verwenden möchte, aber nicht an einer tatsächlichen Umrechnung interessiert ist.

Bsp. 3-1-39

1 000 DM (= 3 353,85 FRF)
3 353,85 FRF
1 000 DM

```
\usepackage{euro}
\EUROFORMAT{main}{\in\ (=\,\out)} \EURO{DEM}[FRF]{1000}\par
\EUROFORMAT{main}{\out}           \EURO{DEM}[FRF]{1000}\par
\EUROFORMAT{main}{\in}            \EURO{DEM}     {1000}
```

Die Formate `in` und `out` geben an, wie die Ausgangs- und Zielwährungen formatiert werden sollen. Dies geschieht unter Verwendung der reservierten Schlüsselwörter `\val` (monetärer Betrag), `\iso` (Währungscode) und `\sym` (Währungssymbol, sofern vorhanden, ansonsten der ISO-Code).

Bsp. 3-1-40

DM 1 000 (FRF 3 353,85)

```
\usepackage{euro}
\EUROFORMAT{in}{\sym~\val} \EUROFORMAT{out}{\iso~\val}
\EURO{DEM}[FRF]{1000}
```

Interessanter sind vielleicht die Möglichkeiten, die Formatierung der monetären Beträge zu verändern. Hierfür bietet das Paket fünf Deklarationen, die im zweiten Argument von `\EUROFORMAT` verwendet werden können. Die Deklaration `\round` gibt an, an welcher Stelle der monetäre Betrag gerundet wird: Bei positiven Werten wird der ganzzahlige Anteil gerundet, bei negativen Werten die Nachkommastellen. `\round{-3}` bedeutet beispielsweise, dass auf drei Nachkommastellen gerundet wird und auch nur diese drei angezeigt werden. Die Deklaration `\form` hat drei Argumente: das Trennzeichen für die Zifferngruppen des ganzzahligen Anteils (standardmäßig `\,`), das Dezimaltrennzeichen (standardmäßig ein Komma) und das Trennzeichen für die Zifferngruppen der Nachkommastellen (standardmäßig `\,`).

Das erste Argument kann entweder `all` lauten (dann wird die Standard-Zahlenformatierung definiert) oder ein ISO-Währungscode sein (dann wird die Formatierung einer einzelnen Währung geändert).

Bsp. 3-1-41

1,022·5838 Euro
−335·3855 FRF
9,900,000 Lit.

```
\usepackage{euro} \EUROFORMAT{main}{\out}
\EUROFORMAT{all}{\round{-4}\form{,}{\textperiodcentered}{}}
\EUROFORMAT{ITL}{\round{2}}
\noindent \EURO{DEM}{2000}\\ \EURO{DEM}[FRF]{-100}\\
\EURO{DEM}[ITL]{10000}
```

Die Deklaration `\minus` formatiert negative Werte, indem sie ihr erstes Argument vor der Zahl und das zweite Argument nach der Zahl ausführt (voreingestellt `\minus{$-$}{}`). Die Zahl selbst wird ohne Vorzeichen formatiert,

dieses wird von der Deklaration hinzugefügt. Die Deklaration \plus funktioniert analog für positive Zahlen (voreingestellt \plus{}{}).

```
\usepackage{color,euro} \EUROFORMAT{main}{\out}
\EUROFORMAT{all}{\plus{$+$}{}\minus{\color{blue}$-$}{}}
\EURO{DEM}{2000}\quad \EURO{DEM}[FRF]{-100}
```

+1 022,58 Euro −335,39 FRF

Bsp. 3-1-42

Die Deklaration \zero hat drei Argumente, die den Fall behandeln, wenn alles null ist, der ganzzahlige Anteil null ist oder die Nachkommastellen null sind. Im ersten und dritten Argument muss auch das Dezimaltrennzeichen eingegeben werden; dies sollte also dem Standard oder dem Wert des Befehls \form entsprechen.

```
\usepackage{eurosans,euro}
\EUROFORMAT{main}{\out} \EUROSYM{EUR}{\euro}
\EUROFORMAT{all}{\zero{0,00}{0}{,--}}
\EURO{DEM}{0}\quad \EURO{DEM}{1}\quad \EURO{EUR}{1}
```

0,00 € 0,51 € 1,– €

Bsp. 3-1-43

3.1.10 lettrine – Schmücken von Absätzen

Bei einigen Arten von Veröffentlichungen wird manchmal der Anfangsbuchstabe einiger Absätze durch einen sehr großen Buchstaben dargestellt, der häufig in den Absatz hineingesetzt wird (der um diesen Buchstaben herumfließt). Der restliche Teil des Satzes oder der Phrase folgt gewöhnlich in einer speziellen Schrift. Diese Art des Satzes wird nicht nur für Kapitelanfänge von Romanen, sondern auch zur Aufnahme neuer Gedankengänge bis hin zur reinen Dekoration und Auflockerung eines Zeitschriftentextes verwendet. Sie lässt sich bis zu den Anfangstagen des Buchdrucks zurückverfolgen, als solche Initiale häufig nach dem Druckvorgang von Hand koloriert wurden. Initiale finden sich sogar in Manuskripten des Mittelalters, also noch vor der Erfindung des Buchdrucks.

\lettrine[*schlüssel-wert-liste*]{*initial*}{*text*}

Das Paket lettrine von Daniel Flipo bietet die Möglichkeit, solche Initiale mit dem Befehl \lettrine zu erstellen. In der einfachsten Form hat er zwei Argumente: den Buchstaben, der als Initial gesetzt werden soll, und den Folgetext, der in einer besonderen Schrift gesetzt werden soll, standardmäßig ist dies \scshape.

```
\usepackage{lettrine} \usepackage[ansinew]{inputenc}
\usepackage[french]{babel}
\lettrine{L}{a moitié des passagers,} affaiblis,
expirants de ces angoisses inconcevables que le
roulis d'un vaisseau porte dans les nerfs et
dans toutes les humeurs du corps agitées en sens
contraire, \ldots
```

LA MOITIÉ DES PASSAGERS, affaiblis, expirants de ces angoisses inconcevables que le roulis d'un vaisseau porte dans les nerfs et dans toutes les humeurs du corps agitées en sens contraire, . . .

Bsp. 3-1-44

Das Initial unterscheidet sich normalerweise vom übrigen Text nur in der Schriftgröße. Wenn man den Befehl \LettrineFontHook umdefiniert und die

Standardbefehle des NFSS verwendet, kann man aber auch eine andere Schriftfamilie für das Initial definieren. Auf dieselbe Art kann auch für den Text im zweiten Argument durch Neudefinition von `\LettrineTextFont` die Schrift geändert werden.

Da der Befehl `\lettrine` die Größe des Initials so berechnet, dass es über mehrere Zeilen passt, sollten nur skalierbare Schriften eingesetzt werden, um optimale Ergebnisse zu erzielen. Die Beispiele in diesem Buch sind standardmäßig mit Adobe Times und Helvetica formatiert, hier ergeben sich also keine Probleme. Später gibt es Beispiele in Palatino, bei der es sich auch um eine skalierbare Type 1-Schrift handelt. Bei Verwendung einer Bitmap-Schrift, wie Computer Modern, müssen jedoch spezielle `.fd`-Dateien verwendet werden (siehe Kapitel 7, Seiten 430ff), um zufriedenstellende Ergebnisse zu erzielen.

Bsp. 3-1-45

LA MOITIÉ DES PASSAGERS, affaiblis, expirants de ces angoisses inconcevables que le roulis d'un vaisseau porte dans les nerfs et dans toutes les humeurs du corps agitées en sens contraire, ...

```
\usepackage{lettrine} \usepackage[ansinew]{inputenc}
\usepackage[french]{babel}
\renewcommand\LettrineFontHook{\sffamily\bfseries}
\renewcommand\LettrineTextFont{\sffamily\scshape}
\lettrine{L}{a moitié des passagers,} affaiblis,
expirants de ces angoisses inconcevables que le
roulis d'un vaisseau porte dans les nerfs et
dans toutes les humeurs du corps agitées en sens
contraire, \ldots
```

Viele Bücher über Typographie geben Empfehlungen über die Art, wie große Initiale in Bezug auf den umgebenden Text am besten formatiert werden sollten. Für höchste Qualität ist es oft erforderlich, die Positionierung je nach Form des Initials manuell nachzukorrigieren. Bei Buchstaben mit einem nach links auslaufenden Balken wird z.B. häufig empfohlen, sie in den Rand hineinragen zu lassen. Der Befehl `\lettrine` berücksichtigt diese Anforderungen durch Bereitstellung eines optionalen Argumentes, in dem die Anpassungen in Form einer durch Kommas getrennten Liste eines Schlüssel-Wert-Paares angegeben werden können.

Die Größe des Initials wird standardmäßig so berechnet, dass sie zwei Textzeilen umfasst (gespeichert in `\DefaultLines`); mit dem Schlüsselwort `lines` kann die Anzahl an Zeilen geändert werden. Es gibt eine Ausnahme: Bei Angabe von `lines=1` wird das Initial trotzdem zwei Zeilen hoch formatiert, aber nicht in den Absatz eingefügt, sondern auf der Grundlinie der ersten Textzeile platziert.

Wenn das Initial sowohl in den Absatz eingebettet sein und über die erste Textzeile herausragen soll, ist das Schlüsselwort `loversize` von Nutzen. Der Wert `.2` vergrößert das Initial um 20%. Der Standardwert für dieses Schlüsselwort wird in `\DefaultLoversize` gespeichert. Dieses Schlüsselwort ist auch in Verbindung mit `lraise` hilfreich (voreingestellt 0 in `\DefaultLraise`). Bei einem Initial mit einer größeren Unterlänge, wie einem „Q“, muss der gesamte Buchstabe möglicherweise etwas angehoben werden, um das Überdrucken der Folgezeilen zu verhindern. In diesem Fall kann man mithilfe von `loversize` die Höhe reduzieren, um das Initial korrekt auszurichten.

Mit dem Schlüsselwort `lhang` kann man angeben, wie weit das Initial in den Rand hineinragen soll. Der Wert wird als Bruchteil angegeben, das heißt,

er liegt zwischen 0 und 1. Der Standardwert hierfür wird in \DefaultLhang gespeichert.

QUAND ILS FURENT revenus un peu à eux, ils marchèrent vers Lisbonne ; il leur restait quelque argent, avec lequel ils espéraient se sauver de la faim après avoir échappé à la tempête …

```
\usepackage{palatino,lettrine}
\usepackage[ansinew]{inputenc}
\usepackage[french]{babel}
\lettrine[lines=3, loversize=-0.1, lraise=0.1,
  lhang=.2]{Q}{uand ils furent} revenus un peu à
eux, ils marchèrent vers Lisbonne ; il leur restait
quelque argent, avec lequel ils espéraient se sauver
de la faim après avoir échappé à la tempête \ldots
```

Bsp. 3-1-46

Der Abstand zwischen dem Initial und dem Folgetext in der ersten Zeile wird über den Befehl \DefaultFindent gesteuert (Standardwert 0pt) und kann mithilfe des Schlüsselwortes findent überschrieben werden. Der Einzug der nachfolgenden Zeilen ist standardmäßig 0.5em groß (gespeichert in \DefaultNindent), kann aber über das Schlüsselwort nindent geändert werden. Ein geneigter Einzug ist mithilfe des Schlüsselwortes slope möglich, er beginnt ab der dritten Zeile. Auch hier kann der Standardwert, der im Befehl \DefaultSlope gespeichert ist, geändert werden. Es ist jedoch fraglich, ob jemals etwas anderes als 0pt benötigt wird, da eine Neigung normalerweise nur für Buchstaben wie „A“ oder „V“ benutzt wird.

ÀPEINE ONT-ILS MIS le pied dans la ville en pleurant la mort de leur bienfaiteur, qu'ils sentent la terre trembler sous leurs pas ; …

```
\usepackage{palatino,lettrine}
\usepackage[ansinew]{inputenc}
\usepackage[french]{babel}
\lettrine[lines=4, slope=0.6em, findent=-1em,
   nindent=0.6em]{À} { peine ont-ils mis} le pied
dans la ville en pleurant la mort de leur
bienfaiteur, qu'ils sentent la terre trembler sous
leurs pas; \ldots
```

Bsp. 3-1-47

Das Beispiel oben zeigt deutlich, dass die Größenberechnung für das Initial Akzente unberücksichtigt lässt. So sollte es normalerweise auch sein. Natürlich kann die Größe auch hier manuell mit loversize angepasst werden.

Wenn links neben dem Initial noch Text eingefügt werden soll (z. B. ein einleitendes Zitat), ist dies mit dem Schlüsselwort ante möglich. Dieses Schlüsselwort ist das einzige, für das es keinen Befehl zum Setzen der Standardeinstellung gibt.

Durch Modifikation der Standardeinstellungen kann das Paket so umdefiniert werden, dass die Initiale nach ganz spezifischen Anforderungen formatiert werden. Die Definitionen können entweder in der Präambel oder in einer Datei namens lettrine.cfg durchgeführt werden, die beim Auffinden geladen wird.

3.1.11 Randausgleich in LaTeX

Zur Formatierung von Absätzen setzt LaTeX den bereits im TeX-Programm implementierten Algorithmus ein, der standardmäßig bündige Absätze erzeugt. Mit anderen Worten: Die Abstände zwischen Wörtern werden leicht gedehnt

oder gestaucht, um Zeilen gleicher Länge zu erzeugen. TEX erreicht dies mit einem Algorithmus, der versucht, für einen ganzen Absatz eine optimale Lösung zu finden. Dabei benutzt es die aktuellen Einstellungen von etwa 20 internen Parametern. Hierzu gehören Aspekte wie optisch übereinstimmende Zeilen, d.h. dass eine sehr locker gesetzte Zeile nicht direkt unter einer sehr eng beschriebenen Zeile steht und dass mehrere Trennstriche in aufeinander folgenden Zeilen als Zeichen schlechter Qualität betrachtet werden. Das Zusammenspiel dieser Parameter ist sehr kompliziert und selbst Experten können oft nicht voraussagen, wie das Ergebnis ausfallen wird. Da die Standardeinstellungen für fast alle Anwendungen geeignet sind, werden in diesem Buch nur einige dieser Parameter beschrieben. Anhang B.3.3 erörtert, wie die Arbeitsweise des Algorithmus überwacht werden kann. Tiefere Einblicke in die Thematik des automatischen Zeilenumbruchs in Absätzen gibt *The TEXbook* [87, Kapitel 14], das den Algorithmus im Detail beschreibt, oder auch der sehr interessante Artikel von Michael Plass und Donald Knuth über dieses Thema, der in *Digital Typography* [83] abgedruckt ist.

Der global optimierende Ansatz von TEX hat einen Nachteil, dem jeder früher oder später begegnen wird: Kleine Änderungen, wie die Korrektur eines Tippfehlers am Ende eines Absatzes, können drastische und überraschende Effekte haben, da sie den Zeilenumbruch des gesamten Absatzes betreffen können. Möglich und absolut nicht unwahrscheinlich ist beispielsweise, dass beim *Löschen* eines Wortes der Absatz um eine Zeile *länger* wird. Dieses Verhalten kann sehr ärgerlich sein, wenn man sich am Ende eines wichtigen Projektes befindet (wie der zweiten Ausgabe dieses Buches) und eine Korrektur bei den bereits manuell überarbeiteten Seitenumbrüchen ein heilloses Chaos anrichtet. In solch einer Situation ist es das Beste, an strategisch günstigen Stellen `\linebreak`- oder `\pagebreak`-Befehle einzufügen, um TEX zu zwingen, eine Lösung zu wählen, die es normalerweise als minderwertig betrachtet. Diese manuellen Umbrüche können später leicht entfernt werden, indem man sich eigene Befehle definiert, z.B.

```
\newcommand\finallinebreak{\linebreak}
```

anstatt direkt Standardbefehle von LATEX zu benutzen. Dadurch lassen sich Layoutänderungen für eine bestimmte Version leicht von anderen Verwendungen der ursprünglichen Befehle unterscheiden – diese Methode hat sich auch bei der Erstellung dieses Buches bewährt.

Die Wortzwischenräume in bündig formatierten Absätzen (die Leerräume zwischen einzelnen Wörtern) werden durch verschiedene TEX-Parameter gesteuert. Die wichtigsten davon sind `\tolerance` und `\emergencystretch`. Wenn diese Parameter richtig eingestellt sind, kann man alle oder zumindest fast alle „Overfull box“-Warnungen vermeiden, ohne dass Zeilen manuell umbrochen werden müssen. Der Parameter `\tolerance` gibt an, wie stark Wortzwischenräume in einem Absatz von ihrer optimalen Größe abweichen dürfen.[1] Dieser Befehl ist ein TEX-Zähler (kein LATEX-Zähler) und hat daher eine etwas ungewöhnliche Syntax für die Wertzuweisung, z.B. `\tolerance=500`. Niedrigere Werte führen dazu, dass TEX nur Lösungen in der Nähe des Optimums akzeptiert, höhere Werte erlauben größere Abweichungen beim

[1] Die optimale Größe ist fontabhängig; siehe Abschnitt 7.10.3 auf Seite 439.

Setzen. Voreingestellt ist häufig ein Wert von 200. Wenn TEX nicht innerhalb der vorgegebenen Toleranz bleiben kann, erhält man in der Ausgabe überlaufende Zeilen (d.h. Zeilen die, wie die vorherige, über den Rand hinaus ragen). Wenn der Wert für `\tolerance` erhöht wird, zieht TEX auch ungünstigere Zeilenumbrüche in Betracht (wie in diesem Absatz), bevor es das Problem für eine manuelle Lösung an den Benutzer übergibt. Werte zwischen 50 und 9999 sind vernünftige Einstellungen, 10000 oder höhere Werte sollten nicht eingesetzt werden, denn diese erlauben es TEX, beliebig schlechte Zeilenumbrüche vorzunehmen (so wie in dieser Zeile).

Vorsicht bei TEXs Vorstellung von „unendlich schlecht“ (infinitely bad)

Wenn man wirklich vollautomatische Zeilenumbrüche benötigt, z.B. bei Anwendungen, bei denen der Text automatisch aus einer Datenbank zusammengestellt wird, ist es besser, den Längenparameter `\emergencystretch` auf einen positiven Wert zu setzen. Wenn TEX (aufgrund der Einstellungen für `\tolerance`) einen Absatz nicht umbrechen kann, ohne dabei Zeilenüberläufe zu produzieren, und `\emergencystretch` positiv ist, fügt es diese Länge als dehnbaren Leerraum jeder Zeile hinzu. Dadurch werden Zeilenumbrüche akzeptabel, die zuvor verworfen wurden. Das kann zu einigen „Underfull Box“-Warnungen (nicht ganz ausgefüllten Zeilen) führen, da nun alle Zeilen nach einem lockereren Maßstab gesetzt werden. Dieses Ergebnis sieht aber immer noch besser aus als eine einzelne, hässliche Zeile inmitten eines ansonsten perfekt gesetzten Absatzes.

Die oben beschriebenen Parameter werden in LATEX durch zwei vordefinierte Befehle beeinflusst: durch den voreingestellten Befehl `\fussy` und den Befehl `\sloppy`, der relativ ungünstige Zeilenumbrüche zulässt. Der Befehl `\sloppy` wird von LATEX automatisch immer dann verwendet, wenn perfekte Zeilenumbrüche aufgrund des engen Zeilenmaßes unwahrscheinlich sind (z.B. beim Formatieren von `\marginpar`-Befehlen oder von p-Spalten in der Umgebung `tabular`).

Text ohne Randausgleich

Wie man theoretisch hochwertigen, bündigen Text erzeugt, weiß im Prinzip jeder (auch wenn überraschender Weise nur wenige Satzsysteme außer TEX Algorithmen verwenden, die tatsächlich beabsichtigt eine hohe Qualität produzieren), aber bei nicht bündigem Text sieht das Ganze etwas anders aus. Auf den ersten Blick erscheint das etwas seltsam. Warum sollte es schwierig sein, einen Absatz in Zeilen verschiedener Länge aufzuteilen? Die Antwort ist, dass es keine quantitativ messbaren Qualitätsmaßstäbe gibt, anhand derer man einfach sagen könnte, ob ein bestimmter Umbruch gut oder schlecht ist. Im Vergleich zu dem Ergebnis bei bündigem Text erzielt TEX beim Erzeugen nicht bündiger Texte sehr schlechte Ergebnisse. Um höchste Qualität zu erhalten, ist deshalb in vielen Fällen ein Eingreifen des Benutzers erforderlich, der an strategischen Punkten explizite Zeilenumbrüche einfügt. Eine gute Einführung in diese Thematik bietet der Artikel von Paul Stiff [155].

Die wichtigste Variante des nicht bündigen Textes ist die, in der die Zeilen linksbündig formatiert sind und rechts flattern. Für diese Art des Satzes bietet LATEX die Umgebung `flushleft`. Sie setzt den gesamten Text in seinem Gültigkeitsbereich „linksbündig“, indem auf der rechten Seite jeder Zeile sehr

dehnbarer Leerraum eingefügt wird; d.h. der interne Parameter `\rightskip` erhält den Wert `0pt plus 1fil`. Diese Einstellung erzeugt häufig sehr „zerrupft“ aussehende Absätze, weil alle Zeilen gleich bewertet werden, ungeachtet dessen, wie viel Text sie enthalten. Außerdem wird die Silbentrennung im Wesentlichen unterdrückt, weil Binde- oder Trennstriche zur schlechten Bewertung („badnesss“) einer Zeile beitragen. Da es kein Mittel gibt, diesem Verhalten gegenzusteuern, wählt der Algorithmus von TEX vorzugsweise Zeilenumbrüche, bei denen eine Silbentrennung vermieden wird.

Bsp. 3-1-48

„Das Schriftsatzsystem LATEX ist eine spezielle Form des TEX-Programms von Donald Knuth. TEX ist ein anspruchsvolles Programm zur Erzeugung hochwertig gesetzten Textes, insbesondere mathematischer Texte.“

```
\begin{flushleft}
"‘Das Schriftsatzsystem \LaTeX{} ist eine spezielle
Form des \TeX{}-Pro\-gramms von Donald Knuth.
\TeX{} ist ein anspruchsvolles Programm zur
Erzeugung hochwertig gesetzten Textes, insbesondere
mathematischer Texte."’
\end{flushleft}
```

Alles in allem ist die LATEX-Umgebung `flushleft` nicht sonderlich gut für längere nicht bündige Texte geeignet, die an der rechten Grenze lediglich um ein gewisses Maß variieren sollen und bei denen an geeigneter Stelle Trennstriche eingefügt werden sollen (siehe nächster Abschnitt für Alternativen). Nichtsdestotrotz kann sie nützlich sein, um einzelne Objekte, wie eine Graphik, linksbündig an den Rand zu setzen, insbesondere da diese Umgebung, genauso wie Listenumgebungen, einen Abstand über und unter sich einfügt.

Eine weitere, bedeutende Einschränkung ergibt sich aus der Tatsache, dass die von dieser Umgebung gewählten Einstellungen nicht allgemeingültig sind. Einige Umgebungen (wie `minipage` oder `tabular`) und Befehle (wie `\parbox`, `\footnote` und `\caption`) setzen die Ausrichtung für Absätze wieder auf Blocksatz zurück, d.h. sie setzen den Abstand zum rechten Rand (`\rightskip`) auf `0pt`. Damit wird der dehnbare Leerraum am rechten Zeilenrand gelöscht. Eine Möglichkeit, dieses Problem automatisch zu lösen, bietet das Paket `ragged2e` (siehe nächster Abschnitt).

Weitere Formate zum Setzen von Absätzen sind: rechtsbündig mithilfe der Umgebung `flushright` und zentriert mithilfe der Umgebung `center`. In diesen beiden Umgebungen werden Zeilenumbrüche normalerweise durch den Befehl `\\` erzeugt, während man beim Flattersatz (mit der oben erörterten Umgebung `flushleft`) LATEX die Zeilenumbrüche überlassen kann (sofern man mit der Qualität des Ergebnisses zufrieden ist).

Die drei Umgebungen, die in diesem Abschnitt erklärt wurden, ändern das Absatzformat, indem sie die Parameter ändern, mit deren Hilfe TEX Absätze formatiert. Wie man aus der folgenden Zuordnungstabelle ersehen kann, sind sie auch als Deklarationen verfügbar:

Umgebung:	`center`	`flushleft`	`flushright`
Befehl:	`\centering`	`\raggedright`	`\raggedleft`

Anders als die entsprechenden Umgebungen beginnen diese Befehle keinen neuen Absatz und fügen keinen vertikalen Leerraum hinzu. Dadurch können sie innerhalb anderer Umgebungen, insbesondere in einer `\parbox`, verwendet werden, um die Ausrichtung in p-Spalten einer `array`- oder `tabular`-

Umgebung zu steuern. Zu beachten ist jedoch, dass das Ende einer Tabellenzeile nicht mit dem Befehl \\ angegeben werden kann, wenn diese Befehle in der letzten Spalte einer tabular- oder array-Umgebung verwendet werden. Für diesen Zweck kann stattdessen der Befehl \tabularnewline verwendet werden (siehe auch Abschnitt 5.2.1).

3.1.12 ragged2e – Verbessern des Randausgleichs

Im vorangegangenen Abschnitt wurden die Mängel der LaTeX-Umgebungen flushleft und flushright angesprochen. Das Paket ragged2e von Martin Schröder sucht Alternativen, die nicht solch extrem flatternde Texte erzeugen. Diese Aufgabe ist nicht so trivial, wie es sich anhört, denn es reicht nicht aus, \rightskip einen Wert wie 0pt plus 2em zuzuweisen. Abgesehen davon, dass diese Einstellung dazu führen würde, dass TeX versucht, die Zeilenenden innerhalb der Grenze von 2em zu halten, bleibt ein kleines Problem: Bei den meisten Zeichensätzen ist auch der Wortzwischenraum dehnbar. Wenn \rightskip nur eine endliche Dehnbarkeit hat, wird TeX an allen Leerräumen gleichmäßig zusätzlichen Leerraum einfügen. Je nach Dichte des Textes entstehen damit unterschiedlich große Wortzwischenräume. Hier muss der Wortzwischenraum so umdefiniert werden, dass er nicht mehr größer oder kleiner werden kann. Dies geschieht über die Angabe eines geeigneten (fontabhängigen) Wertes für \spaceskip. Dieser interne TeX-Parameter gibt, sofern er nicht null ist, den aktuellen Wortzwischenraum wieder, wodurch der Standardwert überschrieben wird, der vom aktuellen Zeichensatz definiert wurde.

Standardmäßig ändert das Paket nicht die LaTeX-Standardbefehle und -umgebungen, die im vorigen Abschnitt erörtert wurden. Stattdessen definiert es eigene Befehle und Umgebungen mit identischen Namen, aber unterschiedlicher Groß-/Kleinschreibung.[1] Die neuen Umgebungen und Befehle sind in der folgenden Zuordnungstabelle aufgeführt:

Umgebung:	Center	FlushLeft	FlushRight
Befehl:	\Centering	\RaggedRight	\RaggedLeft

Sie unterscheiden sich von ihren Gegenstücken aus dem vorigen Abschnitt nicht nur dadurch, dass sie versuchen, weniger flatternde Absätze zu produzieren, sondern auch dadurch, dass die meisten ihrer Formatierungsparameter geändert werden können. Dies erhöht die Flexibilität.

Überladen der ursprünglichen Befehle

Da es etwas anstrengend ist, ständig die Befehle und Umgebungen mit gemischter Groß-/Kleinschreibung einzugeben, gibt es die Möglichkeit, die ursprünglichen Befehle und Umgebungen, wie \raggedright, mit den neuen Definitionen zu überschreiben. Hierzu gibt man beim Laden des Paketes die Option newcommands an.

Das Paket bietet eine Vielzahl an Parametern, die das genaue Verhalten der neuen Befehle und Umgebungen definieren (siehe Tabelle 3.2 auf der gegenüberliegenden Seite). Bei \RaggedRight oder FlushLeft kann der Leerraum, der an der rechten Seite jeder Zeile hinzugefügt wird, über den Parame-

[1] Genau genommen widerspricht dies den Standard-Namenskonventionen. In den meisten Paketen weisen Befehle mit gemischter Groß-/Kleinschreibung auf Schnittstellenbefehle hin, die von Entwicklern in Klassendateien oder in der Präambel verwendet werden, jedoch nicht auf Dokumentbefehle.

\RaggedLeftParindent	0pt	\RaggedLeftLeftskip	0pt plus 2em
\RaggedLeftRightskip	0pt	\RaggedLeftParfillskip	0pt
\CenteringParindent	0pt	\CenteringLeftskip	0pt plus 2em
\CenteringRightskip	0pt plus 2em	\CenteringParfillskip	0pt
\RaggedRightParindent	0pt	\RaggedRightLeftskip	0pt
\RaggedRightRightskip	0pt plus 2em	\RaggedRightParfillskip	0pt plus 1fil
\JustifyingParindent	1 em	\JustifyingParfillskip	0pt plus 1fil

Tabelle 3.2: Von ragged2e verwendete Parameter

ter \RaggedRightRightskip eingestellt werden und der Leerraum auf der linken Seite mittels \RaggedRightLeftskip. Der zu verwendende Absatzeinzug kann über \RaggedRightParindent abgerufen werden und der Leerraum zum Auffüllen der letzten Zeile ist über \RaggedRightParfillskip verfügbar. Auf ähnliche Weise können auch die Einstellungen für \Centering und \RaggedLeft geändert werden; dazu muss lediglich RaggedRight in den Parameternamen durch Centering oder RaggedLeft ersetzt werden.

Nicht bündiger Satz als Standard

Um ein komplettes Dokument im Flattersatz zu formatieren, muss das ragged2e-Paket mit der Option document geladen werden. Für den bündigen Satz einzelner Absätze bietet das Paket den Befehl \justifying und die Umgebung justify. Beide können über die Längenparameter \JustifyingParindent und \JustifyingParfillskip individuell angepasst werden.

Ein Dokument mit mäßig flatternden Absätzen und einem Texteinzug von 12pt ließe sich also wie im folgenden Beispiel erzeugen (vgl. Beispiel 3-1-48 auf Seite 111).

Bsp. 3-1-49

„Das Schriftsatzsystem LaTeX ist eine spezielle Form des TeX-Programms von Donald Knuth. TeX ist ein anspruchsvolles Programm zur Erzeugung hochwertig gesetzten Textes, insbesondere mathematischer Texte.“

```
\usepackage[document]{ragged2e}
\setlength\RaggedRightRightskip{0pt plus 1cm}
\setlength\RaggedRightParindent{12pt} % Einzug
"‘Das Schriftsatzsystem \LaTeX{} ist eine spezielle
Form des \TeX{}-Pro\-gramms von Donald Knuth.
\TeX{} ist ein anspruchsvolles Programm zur
Erzeugung hochwertig gesetzten Textes, insbesondere
mathematischer Texte."’
```

Nicht bündige Formatierung in engen Spalten

Bei geringer Zeilenlänge (z.B. in \marginpars, \parboxen, minipage-Umgebungen oder p-Spalten von tabular-Umgebungen) führt Blocksatz zu eher dürftigen Ergebnissen. Mit der Option raggedrightboxes werden Absätze an solchen Stellen automatisch mit \RaggedRight formatiert. Bei Bedarf kann dann mit \justifying bündiger Text erzeugt werden.

Die Standardwerte

Wenn in den Voreinstellungen em-Werte verwendet werden (siehe Tabelle 3.2), ist besondere Vorsicht geboten, weil em beim Laden des Paketes in ein reales Maß umgewandelt wird. Daher sollte das Paket geladen werden, *nachdem* die Grundschrift und die Größe eingerichtet wurden – z.B. nach dem Laden der Fontpakete.

Anstatt die in der Tabelle 3.2 aufgeführten Voreinstellungen zu verwenden, kann man das Paket auch anweisen, zunächst LaTeXs Standardwerte zu

verwenden, indem man es mit der Option `originalparameters` lädt und dann nach Bedarf die Parameterwerte wie gewünscht ändert.

3.1.13 setspace – Ändern des Zeilenvorschubs

Der Befehl `\baselineskip` ist ein TEX-Parameter, mit dem der *Zeilenvorschub* festgelegt wird. Dies ist der Abstand zweier aufeinander folgender Grundlinien, manchmal auch als Zeilenabstand oder als Durchschuss bezeichnet, siehe aber die Diskussion der Begriffe auf Seite 426. Standard-LATEX definiert den Zeilenvorschub um fast 20% größer als die Entwurfsgröße des verwendeten Zeichensatzes. Da es nicht empfehlenswert ist, die Einstellung von `\baselineskip` direkt zu ändern, verfügt LATEX über den Befehl `\linespread`, mit dem sich `\baselineskip` global für alle Größen ändern lässt. Nach einer Anweisung wie etwa `\linespread{1.5}\selectfont` tritt der neue Wert sofort in Kraft.[1]

Das Paket `setspace` (von Geoffrey Tobin und anderen) definiert Umgebungen und Befehle zum Einfügen variabler Abstände (hauptsächlich doppelt und anderthalb). Um die Abstände im ganzen Dokument zu definieren, gibt es drei Befehle: `\singlespacing`, `\onehalfspacing` und `\doublespacing`; sie werden in der Präambel festgelegt. Daneben kann mit dem Befehl `\setstretch` ein anderer Abstand in der Präambel angegeben werden. Im (obligatorischen) Argument wird der gewünschte Vergrößerungsfaktor angegeben. Wenn keiner dieser Befehle vorhanden ist, wird die Standardeinstellung „einfacher Zeilenvorschub“ verwendet.

Abstände innerhalb des Dokumentes können mit drei spezifischen Umgebungen geändert werden: `singlespace`, `onehalfspace` und `doublespace`. Sie definieren einen einfachen, anderthalbfachen bzw. doppelten Zeilenvorschub. Diese Umgebungen lassen sich nicht verschachteln.

```
\usepackage{setspace}
\begin{doublespace}
 Am Anfang schuf Gott Himmel und Erde.
 Und die Erde war wüst und leer, und
 es war finster auf der Tiefe, und der
 Geist Gottes schwebte auf dem Wasser.
\end{doublespace}
```

Bsp. 3-1-50

Am Anfang schuf Gott Himmel und Erde. Und die Erde war wüst und leer, und es war finster auf der Tiefe, und der Geist Gottes schwebte auf dem Wasser.

Mit der Umgebung `spacing` können beliebige Zeilenvorschübe definiert werden. Sie erwartet als obligatorisches Argument den Wert für `\baselinestretch`, der für den Inhalt der Umgebung verwendet werden soll.

```
\usepackage{setspace}
\begin{spacing}{2.0}
 Am Anfang schuf Gott Himmel und Erde.
 Und die Erde war wüst und leer, und
 es war finster auf der Tiefe, und der
 Geist Gottes schwebte auf dem Wasser.
\end{spacing}
```

Bsp. 3-1-51

Am Anfang schuf Gott Himmel und Erde. Und die Erde war wüst und leer, und es war finster auf der Tiefe, und der Geist Gottes schwebte auf dem Wasser.

[1] Die veraltete LATEX 2.09 Lösung `\renewcommand\baselinestretch{1.5}` benötigt dagegen einen nachfolgenden Schriftgrößenwechsel (wie `\small` oder `\Large`), damit der neue Wert in Kraft tritt.

Abstand	10pt	11pt	12pt
anderthalb	1.25	1.21	1.24
doppelt	1.67	1.62	1.66

Tabelle 3.3: Werte für `\baselinestretch` bei verschiedenen Schriftgrößen

Im obigen Beispiel erzeugt der Koeffizient „2.0“ einen Zeilenvorschub, der größer ist als der „doppelte Zeilenabstand“ (`doublespace`-Umgebung), der für einige Publikationen benötigt wird. Mit der Umgebung `spacing` wird der Zeilenvorschub zweimal vergrößert – einmal durch `\baselineskip` (durch den der Zeilenvorschub bereits um etwa 20% gegenüber der Schriftgröße erhöht wird) und zum Zweiten durch Setzen von `\baselinestretch`. „Doppelter Abstand“ bedeutet, dass der vertikale Abstand zwischen den Grundlinien ungefähr doppelt so groß ist wie die Zeichengröße. Da `\baselinestretch` das Verhältnis zwischen dem gewünschten Zeilenvorschub und `\baselineskip` angibt, lässt sich der korrekte Wert für `\baselinestretch` bei verschiedenen Grundschriftgraden leicht errechnen. In Tabelle 3.3 sind die Werte für zwei unterschiedliche Abstände und drei Grundschriftgrade dargestellt.

3.1.14 picinpar – Rechteckige Löcher in Absätzen

Mit dem Paket picinpar (das Friedhelm Sowa auf der Basis früherer Arbeiten von Alan Hoenig entwickelte) können „Fenster“ innerhalb von Absätzen gesetzt werden. Die Basisumgebung dafür ist `window`. Sie hat ein obligatorisches Argument in Form einer durch Kommas getrennten Liste aus vier Elementen, das im Gegensatz zu den LaTeX-Konventionen in eckigen Klammern gesetzt wird. Diese Elemente geben die Anzahl an Zeilen vor dem Beginn des Fensters sowie die Ausrichtung des Fensters im Absatz (`l` für links, `c` für zentriert und `r` für rechts) an, außerdem den im Fenster dargestellten Text, das Bild o. ä. und erläuternden Text über den Inhalt im Fenster (z.B. die Bildunterschrift).

Bsp. 3-1-52

Im vorliegenden Beispiel wird ein vertikal gedrucktes Wort horizontal und vertikal zentriert in einen Absatz eingefügt. Es ist leicht vorstellbar, dass sich mit der Umgebung `tabwindow` auch Tabellen einfach einbinden lassen.

H a l l o

Wenn ein Fenster länger ist als der Absatz, in dem es beginnt, dann reicht es über das Ende des Absatzes in den oder die nächsten Absätze hinein.

```
\usepackage{picinpar}
\begin{window}[1,c,%
    \fbox{\shortstack{H\\a\\l\\l\\o}},]
 Im vorliegenden Beispiel wird ein vertikal
gedrucktes Wort horizontal und vertikal
zentriert in einen Absatz eingefügt. Es ist
leicht vorstellbar, dass sich mit der Umgebung
\texttt{tabwindow} auch Tabellen einfach
einbinden lassen.\par Wenn ein Fenster länger
ist als der Absatz, in dem es beginnt, dann
reicht es über das Ende des Absatzes in den
oder die nächsten Absätze hinein.
\end{window}
```

Bei genauerer Betrachtung des obigen Beispiels kann man feststellen, dass der Einzug des zweiten Absatzes nicht stimmt. Dieser Fehler lässt sich einfach beheben, indem der Einzug bei Bedarf explizit mit `\par\indent` angefordert wird.

Ein Fenster wie im vorigen Beispiel zu zentrieren, funktioniert nur, wenn der umfließende Text auf beiden Seiten noch ausreichend breit ist (wobei „ausreichend“ mehr als ein Zoll ist). Ansonsten wird das Paket den Platz einfach mit Weißfläche auffüllen.

Das Paket stellt zwei Varianten der Umgebung namens `figwindow` und `tabwindow` zur Verfügung. Sie formatieren den erläuternden Text als Bild- oder Tabellenunterschriften und fügen eine entsprechende Nummerierung hinzu. Man sollte solch „nicht gleitende“ Gleitobjekte nur mit größter Sorgfalt mit den Standardumgebungen `figure` und `table` vermischen, da letztere hinter die nicht gleitenden Objekte rutschen können, womit eine falsche Reihenfolge der Abbildungs- bzw. Tabellennummern entstünde.

Das nächste Beispiel zeigt solch eine eingebettete Abbildung – eine Karte von Großbritannien in einem Absatz. Leider ist die Formatierung der Bildunterschrift mehr oder weniger fest im Paket integriert; um sie zu ändern, muss der interne Befehl `\@makewincaption` angepasst werden.

Is this a dagger which I see before me, The handle toward my hand? Come, let me clutch thee. I have thee not, and yet

Figure 1: United Kingdom

I see thee still. Art thou not, fatal vision, sensible To feeling as to sight? or art thou but A dagger of the mind, a false creation, Proceeding from the heat-oppressed brain? I see thee yet, in form as palpable As this which now I draw. Thou marshall’st me the way that I was going; And such an instrument I was to use. Mine eyes are made the fools o’ the other senses, Or else worth all the rest; I see thee still, And on thy blade and dudgeon gouts of blood, Which was not so before. (*Macbeth*, Act II, Scene 1).

```
\usepackage{picinpar,graphicx}
\begin{figwindow}[3,l,%
 \fbox{\includegraphics[width=20mm]{ukmap}},%
                    {United Kingdom}]
 Is this a dagger which I see before me, The
 handle toward my hand? Come, let me clutch
 thee. I have thee not, and yet I see thee
 still.  Art thou not, fatal vision,
 sensible To feeling as to sight?  or art
 thou but A dagger of the mind, a false
 creation, Proceeding from the
 heat-oppressed brain?  I see thee yet, in
 form as palpable As this which now I draw.
 Thou marshall’st me the way that I was
 going; And such an instrument I was to use.
 Mine eyes are made the fools o’ the other
 senses, Or else worth all the rest; I see
 thee still, And on thy blade and dudgeon
 gouts of blood, Which was not so before.
 (\emph{Macbeth}, Act II, Scene 1).
\end{figwindow}
```

Bsp. 3-1-53

3.2 Fußnoten, Endnoten und Marginalien

LaTeX verfügt über Möglichkeiten „eingefügten“ Text, wie Marginalien, Fußnoten, Abbildungen und Tabellen zu setzen. Der vorliegende Abschnitt befasst sich eingehend mit den verschiedenen Arten von Anmerkungen, während in Kapitel 6 Gleitobjekte ausführlich beschrieben werden.

Den Anfang macht eine Erörterung der Möglichkeiten, die LaTeXs Fußnotenbefehle bieten. Es wird gezeigt, wie und wie weit sie an individuelle Bedürfnisse angepasst werden können. Für zweispaltige Dokumente bietet das Paket ftnright ein spezielles Layout für Fußnoten, bei dem alle Fußnoten am unteren Rand der rechten Spalte gesetzt werden. Anschließend wird das Paket footmisc vorgestellt, das die meisten Einschränkungen der Standardbefehle beseitigt und eine Fülle zusätzlicher Funktionen bietet. Das Paket manyfoot

(das mit footmisc kombiniert werden kann) erweitert die Unterstützung von Fußnoten auf Bereiche wie die Linguistik, indem es mehrere, voneinander unabhängige Fußnotenbefehle bereitstellt.

Endnoten werden durch das Paket endnotes unterstützt, das eine Mischung von Fußnoten und Endnoten ermöglicht. Es kann auch zum Erstellen von Anmerkungsapparaten verwendet werden, wie es manche Verlage vorschreiben. Der Abschnitt schließt mit einer Beschreibung von Marginalien, die bereits Bestandteil von Standard-LaTeX sind.

3.2.1 Verwenden von Standardfußnoten

LaTeX unterscheidet streng zwischen Fußnoten im Haupttext und Fußnoten in einer minipage-Umgebung. Erstere werden mithilfe des Zählers footnote nummeriert, während der Befehl \footnote innerhalb der minipage-Umgebung so definiert wird, dass er den Zähler mpfootnote verwendet. Dadurch wird die Darstellung des Fußnotenzeichens je nach Kontext durch den Befehl \thefootnote oder \thempfootnote festgelegt. Als Standardzeichen wird im Text eine arabische Zahl und in minipage-Umgebungen ein Kleinbuchstabe verwendet. Andere Darstellungen für Fußnotenzeichen erhält man durch Neudefinition dieser Befehle:

Bsp. 3-2-1

Text Text Text* Text Text† Text.

*Die erste
†Die zweite

```
\renewcommand\thefootnote
             {\fnsymbol{footnote}}
Text Text Text\footnote{Die erste}
Text Text\footnote{Die zweite} Text.
```

Besonderheiten innerhalb von minipage

Fußnoten, die innerhalb einer minipage-Umgebung mit dem Befehl \footnote erstellt werden, verwenden den Zähler mpfootnote und werden am Fuß der Absatzbox gesetzt, die von minipage erzeugt wird. Wenn man allerdings in der minipage-Umgebung den Befehl \footnotemark verwendet, erzeugt dieser Fußnotenzeichen im gleichen Stil und in derselben Reihenfolge wie die Haupttextfußnoten – d.h. der Zähler footnote wird weiter hochgesetzt, und die Darstellung erfolgt mithilfe des Befehls \thefootnote. Dadurch kann man innerhalb der minipage-Umgebung auch eine Fußnote erzeugen, die am Fuß der Seite in einer Sequenz mit den Fußnoten des Haupttextes gesetzt wird: Man platziert den Befehl \footnotemark innerhalb der minipage-Umgebung und hinter die Umgebung setzt man den entsprechenden Befehl \footnotetext.

Bsp. 3-2-2

... Haupttext ...

Fußnoten werden hier mit Kleinbuchstaben nummeriert.[a]
Dieser Text bezieht sich auf eine Fußnote am Fuß der Seite.[1] Und eine weitere[b] Fußnote.

[a] Auf der Miniseite.
[b] Wieder im Text

... Haupttext ...

[1] Am Fuß der Seite

```
\noindent\ldots{} Haupttext \ldots
\begin{center}
 \begin{minipage}{.9\linewidth}
  Fußnoten werden hier mit Kleinbuchstaben
  nummeriert.\footnote{Auf der Miniseite.}
  \par Dieser Text bezieht sich auf eine
  Fußnote am Fuß der Seite.\footnotemark{}
  Und eine weitere\footnote{Wieder im Text}
  Fußnote.
 \end{minipage}\footnotetext{Am Fuß der Seite}
\end{center}
\ldots{} Haupttext \ldots
```

Wie das vorangegangene Beispiel zeigt, kann beim Verweis auf eine Fußnote innerhalb einer minipage-Umgebung \footnotemark nicht verwendet werden, weil dieser Befehl auf Fußnoten am Fuß der Seite verweist. Für diesen Fall kann man das Paket footmisc laden und anstelle von \footnotemark den Befehl \mpfootnotemark verwenden. Wie bei \footnotemark wird mit \mpfootnotemark zunächst der Zähler hochgesetzt und dann der Wert dargestellt. Um auf den vorherigen Wert zu verweisen, muss der Wert in der Regel zuerst heruntergesetzt werden, wie das nächste Beispiel zeigt.

Haupttext ...

Fußnoten werden hier mit Kleinbuchstaben nummeriert.[a]
Dieser Text bezieht sich auf die vorangegangene Fußnote.[a] Und eine weitere[b] Fußnote.

[a] Auf der Miniseite.
[b] Auch innerhalb.

... Haupttext ...

```
\usepackage{footmisc}
\noindent Haupttext \ldots \begin{center}
 \begin{minipage}{.9\linewidth}
  Fußnoten werden hier mit Kleinbuchstaben
  nummeriert.\footnote{Auf der Miniseite.}
  \par Dieser Text bezieht sich auf die
  vorangegangene Fußnote.%
\addtocounter{mpfootnote}{-1}%
\mpfootnotemark{}
  Und eine weitere\footnote{Auch innerhalb.}
  Fußnote.
\end{minipage}
\end{center} \ldots{} Haupttext \ldots
```

Bsp. 3-2-3

Obwohl es in einigen Disziplinen durchaus üblich ist, können mit LaTeX innerhalb von Fußnoten keine weiteren \footnote-Befehle gesetzt werden. Man kann jedoch den Befehl \footnotemark in der ersten Fußnote verwenden und den dazugehörigen Text als Argument des Befehls \footnotetext eingeben. Weitere spezielle Fußnotenanforderungen werden vom Paket manyfoot bedient (siehe unten).

Text[1] und noch mehr Text.

[1] Eine Beispiel[2]-Fußnote.
[2] Eine Fußnoten-Fußnote.

```
Text\footnote{Eine Beispiel\footnotemark{}-%
Fußnote.}\footnotetext{Eine Fußnoten-Fußnote.}
und noch mehr Text.
```

Bsp. 3-2-4

Wie erstellt man einen Querverweis zu einer Fußnote? Zu diesem Zweck kann man den normalen LaTeX-Mechanismus von \label und \ref verwenden. Alternativ besteht die Möglichkeit, sich einen eigenen Befehl zu definieren, um diese Art von Verweisen besonders darzustellen, wie z.B.:

Dies ist irgendein Text.[1]

... wie an Fußnote (1) auf Seite 6 ersichtlich, kann man auf Fußnoten verwei-

[1] Text innerhalb der Bezugsfußnote.

```
\newcommand\fnref[1]{\unskip~(\ref{#1})}
Dies ist irgendein Text.\footnote{Text innerhalb
der Bezugsfußnote\label{fn:myfoot}.}

\ldots wie an Fußnote\fnref{fn:myfoot}
auf Seite~\pageref{fn:myfoot} ersichtlich,
kann man auf Fußnoten verweisen \ldots
```

Bsp. 3-2-5

Standard-LaTeX erlaubt keine Fußnoten innerhalb von Tabellen. In Abschnitt 5.8 werden einige Wege zur Lösung des Problems aufgezeigt.

3.2.2 Anpassen von Fußnoten

Fußnoten sind in LaTeX in der Regel einfach zu benutzen, und LaTeX besitzt einen leistungsfähigen Mechanismus, um Text am Fuß einer Seite zu setzen.[1] Eine Fußnote kann dabei aus mehreren Absätzen bestehen und sowohl Listen als auch im Text eingebettete oder abgesetzte Formeln, Tabellen und Ähnliches enthalten.

LaTeX bietet verschiedene Parameter, mit denen Fußnoten angepasst werden können. Sie sind in Abbildung 3.1 auf der nächsten Seite schematisch dargestellt und bedeuten:

`\footnotesize` Die Schriftgröße, die für Fußnoten verwendet wird (siehe auch Tabelle 7.1 auf Seite 353).

`\footnotesep` Die Höhe einer Linie ohne horizontale Ausdehnung (englisch: strut), die am Anfang jeder Fußnote platziert wird. Wenn sie größer ist als der von `\footnotesize` verwendete Zeilenvorschub `\baselineskip`, wird über jeder Fußnote ein zusätzlicher vertikaler Leerraum eingefügt. In Anhang A.2.3 werden „struts“ näher erläutert.

`\skip\footins` Ein Low-Level-Längenparameter von TeX, der den Abstand zwischen dem Haupttext und dem Beginn der Fußnoten definiert. Dieser Wert kann mit dem Befehl `\setlength` oder `\addtolength` verändert werden, indem man im ersten Argument `\skip\footins` verwendet:

```
\addtolength{\skip\footins}{10mm plus 2mm}
```

`\footnoterule` Ein Makro, das die Linie zeichnet, welche die Fußnote vom Haupttext trennt. Das Makro wird sofort nach dem vertikalen Abstand `\skip\footins` ausgeführt. Die Linie sollte rechnerisch keinen vertikalen Raum belegen, d.h. der Raum, den sie belegt, sollte durch einen negativen Abstand kompensiert werden. Die Standarddefinition sieht in etwa so aus:

```
\renewcommand\footnoterule{\vspace*{-3pt}%
   \hrule width 2in height 0.4pt \vspace*{2.6pt}}
```

Zu beachten ist, dass hier der TeX-Befehl `\hrule` und nicht der LaTeX-Befehl `\rule` verwendet wurde. Letzterer leitet einen neuen Absatz ein. Dies erschwert die Berechnung des Leerraums, der benötigt wird, damit der Effekt „null Höhe“ erzielt wird. Aus diesem Grund lässt sich eine originellere „Linie“ am besten erstellen, indem man eine `picture`-Umgebung ohne Ausdehnung verwendet, in welche die Linie platziert wird, ohne vertikalen Abstand hinzuzufügen.

In den Klassen report und book werden Fußnoten innerhalb von Kapiteln nummeriert; in article erhalten Fußnoten eine im ganzen Dokument durchgängige sequentielle Nummerierung. Letzteres kann mit dem Befehl `\@addtoreset` geändert werden (siehe Anhang A.1.4). Dieser Mechanismus

[1] Eine interessante und vollständige Abhandlung dieses Themas erschien in der Zeitschrift *Cahiers GUTenberg* [10, 135] der französischen TeX-Usergroup.

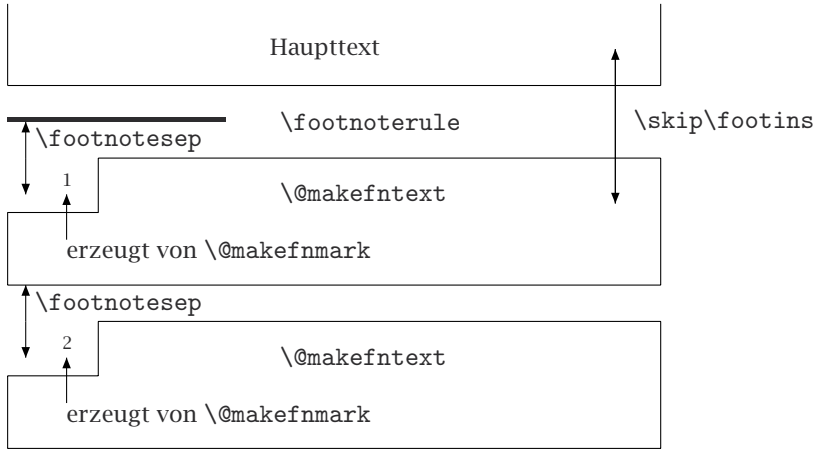


Abbildung 3.1: Schematische Darstellung einer Fußnote

sollte jedoch nicht verwendet werden, um Fußnoten seitenweise zu nummerieren. LaTeX arbeitet vorausschauend, während es die endgültigen Seiten erstellt, und würde deshalb viele Fußnoten vollkommen falsch nummerieren. Zur seitenbezogenen Nummerierung von Fußnoten sollte das Paket footmisc oder perpage verwendet werden (Beschreibung siehe unten).

Der Befehl `\@makefnmark` wird normalerweise verwendet, um ein Fußnotenzeichen zu erstellen. Man würde erwarten, dass dieser Befehl ein Argument hat (die aktuelle Fußnotennummer), aber dies ist nicht der Fall. Stattdessen nutzt er den Befehl `\@thefnmark`, um indirekt auf diese Nummer zu verweisen. Die Erklärung ist einfach: Je nachdem, ob die Nummer innerhalb oder außerhalb einer `minipage`-Umgebung liegt, muss auf einen anderen Zähler zugegriffen werden. Die Definition, die standardmäßig ein hochgestelltes Zeichen erzeugt, sieht in etwa folgendermaßen aus:

```
\renewcommand\@makefnmark
             {\mbox{\textsuperscript{\normalfont\@thefnmark}}}
```

Der `\footnote`-Befehl führt `\@makefntext` in einer `\parbox` der Breite `\columnwidth` aus. Die Standardversion sieht in etwa folgendermaßen aus:

```
\newcommand\@makefntext[1]
           {\noindent\makebox[1.8em][r]{\@makefnmark}#1}
```

Hier wird das Fußnotenzeichen rechtsbündig in einer Box der Breite `1.8em` gesetzt, unmittelbar gefolgt vom Fußnotentext. Zu beachten ist die Mehrfachverwendung des Makros `\@makefnmark`; dadurch bewirken alle Änderungen am Makro, dass die Darstellung des Fußnotenzeichens an beiden Stellen geändert wird. Wenn der Text linksbündig gesetzt werden und die Fußnotenmarkierung im Rand stehen soll, sollte die im nächsten Beispiel definierte Variante verwendet werden. In diesem Fall wird das Fußnotenzeichen nicht mithilfe von `\@makefnmark` formatiert, sondern stattdessen per `\@thefnmark`

auf die Nummer zugegriffen. Das Ergebnis ist, dass das Zeichen nicht hochgestellt, sondern auf der Grundlinie platziert wird. Auf diese Weise werden die Markierungen im Text und am Fuß der Seite unterschiedlich formatiert.

Text Text Text[1] Text Text[2] Text.

1. Die erste
2. Die zweite

Bsp. 3-2-6

```
\makeatletter
\renewcommand\@makefntext[1]%
   {\noindent\makebox[0pt][r]{\@thefnmark.\,}#1}
\makeatother
Text Text Text\footnote{Die erste}
Text Text\footnote{Die zweite} Text.
```

3.2.3 ftnright – Rechte Fußnoten im Zweispaltensatz

Bei zweispaltigem Satz ist es manchmal wünschenswert, alle Fußnoten am Fuß der rechten Spalte zu positionieren. Dies lässt sich mithilfe des Paketes ftnright (von Frank Mittelbach) bewerkstelligen. Die Wirkung dieses Paketes ist in Abbildung 3.2 auf der nächsten Seite dargestellt – die erste Seite der Originaldokumentation des Paketes (einschließlich Rechtschreibfehler). In der Abbildung ist deutlich zu sehen, wie die Fußnoten im unteren Teil der rechten Spalte versammelt sind.

Die zentrale Idee bei dem Paket ftnright ist, dass alle Fußnoten einer Seite am Fuß der rechten Spalte gesetzt werden. Es wird ausreichend Abstand zwischen Fußnoten und Text gelassen und die Fußnoten werden in einem kleineren Schriftgrad gesetzt.[1] Außerdem werden die Fußnotenzeichen auf der Grundlinie gesetzt und nicht hochgestellt.[2]

Das Paket kann zusammen mit den meisten Klassendateien von LaTeX verwendet werden. Allerdings wird das Paket ftnright nur bei Dokumenten wirksam, die mithilfe der Option `twocolumn` für den Befehl `\documentclass` zweispaltig formatiert werden. In den meisten Fällen ist es am sichersten, ftnright als letztes Paket zu verwenden, um sicherzugehen, dass seine Einstellungen nicht durch andere Pakete überschrieben werden.

3.2.4 footmisc – Verschiedene Fußnotenstile

Da Standard-LaTeX nur eine Art von Fußnoten und nur begrenzte (und irgendwie Low-Level-) Unterstützung zur Anpassung bietet, entwickelten einige Leute kleine Pakete, die Funktionen anbieten, die ansonsten nicht verfügbar sind. Viele dieser früheren Bemühungen wurden von Robin Fairbairns in seinem Paket footmisc zusammengefasst, das unter anderem die seitenweise Nummerierung von Fußnoten und das Setzen von mehreren Fußnoten in einem einzigen Absatz am Fuß der Seite unterstützt. In diesem Abschnitt werden die Funktionen dieses Paketes beschrieben und es wird (sofern zutreffend) aufgezeigt, welche älteren Pakete es ersetzt.

[1] Manche Zeitschriften benutzen den gleichen Schriftgrad für Fußnoten und Text, so dass man beide nur schwer auseinander halten kann.

[2] Dies gilt natürlich nur für das Zeichen vor dem Fußnotentext, nicht für die Markierung im Haupttext, wo eine hochgestellte, in kleinerer Schriftgröße gesetzte Zahl oder ein Symbol den Lesefluss nicht unterbrechen.

Footnotes in a multi-column layout*

Frank Mittelbach

August 10, 1991

1 Introduction

The placement of footnotes in a multi-column layout always bothered me. The approach taken by LaTeX (i.e., placing the footnotes separately under each column) might be all right if nearly no footnotes are present. But it looks clumsy when both columns contain footnotes, especially when they occupy different amounts of space.

In the multi-column style option [5], I used page-wide footnotes at the bottom of the page, but again the result doesn't look very pleasant since short footnotes produce undesired gaps of white space. Of course, the main goal of this style option was a balancing algorithm for columns which would allow switching between different numbers of columns on the same page. With this feature, the natural place for footnotes seems to be the bottom of the page[1] but looking at some of the results it seems best to avoid footnotes in such a layout entirely.

Another possibility is to turn footnotes into endnotes, i.e., printing them at the end of every chapter or the end of the entire document. But I assume everyone who has ever read a book using such a layout will agree with me, that it is a pain to search back and forth, so that the reader is tempted to ignore the endnotes entirely.

When I wrote the article about "Future extensions of TeX" [6] I was again dissatisfied with the outcome of the footnotes, and since this article should show certain aspects of high quality typesetting, I decided to give the footnote problem a try and modified the LaTeX output routine for this purpose. The layout I used was inspired by the yearbook of the Gutenberg Gesellschaft Mainz [1]. Later on, I found that it is also recommended by Jan White [9]. On the layout of footnotes I also consulted books by Jan Tschichold [8] and Manfred Simoneit [7], books, I would recommend to everyone being able to read German texts.

1.1 Description of the new layout

The result of this effort is presented in this paper and the reader can judge for himself whether it was successful or not.[2] The main idea for this layout is to assemble the footnotes of all columns on a page and place them all together at the bottom of the right column. Allowing for enough space between footnotes and text, and in addition, setting the footnotes in smaller type[3] I decided that one could omit the footnote separator rule which is used in most publications prepared with TeX.[4] Furthermore, I decided to place the footnote markers[5] at the baseline instead of raising them as superscripts.[6]

All in all, I think this generates a neat layout, and surprisingly enough, the necessary changes to the LaTeX output routine are nevertheless astonishingly simple.

1.2 The use of the style option

This style option might be used together with any other style option for LaTeX which does not change the three internals changed by `ftnright.sty`.[7] In most cases, it is best to use this style option as the very last option in the `\documentstyle` command to make sure that its settings are not overwritten by other options.[8]

*. The LaTeX style option `ftnright` which is described in this article has the version number v1.0d dated 92/06/19. The documentation was last revised on 92/06/19.

1. You can not use column footnotes at the bottom, since the number of columns can differ on one page.

2. Please note, that this option only changed the placement of footnotes. Since this article also makes use of the `doc` option [4], that assigns tiny numbers to code lines sprincled throughout the text, the resulting design is not perfect.

3. The standard layout in *TUGboat* uses the same size for footnotes and text, giving the footnotes, in my opinion, much too much prominence.

4. People who prefer the rule can add it by redefining the command `\footnoterule` [2, p. 156]. Please, note, that this command should occupy no space, so that a negative space should be used to compensate for the width of the rule used.

5. The tiny numbers or symbols, e.g., the '5' in front of this footnote.

6. Of course, this is only done for the mark preceeding the footnote text and not the one used within the main text where a raised number or symbol set in smaller type will help to keep the flow of thoughts, uninterrupted.

7. These are the macros `\@startcolumn`, `\@makecol` and `\@outputdblcol` as we will see below. Of course, the option will take only effect with a document style using a twocolumn layout (like `ltugboat`) or when the user additionally specifies `twocolumn` as a document style option in the `\documentstyle` command.

8. The `ltugboat` option (which is currently set up as a style option instead of a document style option which it actually is) will overwrite

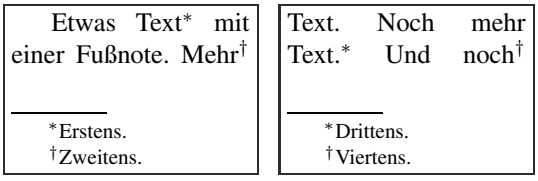
Abbildung 3.2: Platzieren von Text und Fußnoten mit dem Paket ftnright

Die Schnittstelle für footmisc ist recht einfach: Nahezu alles kann durch Angabe von Optionen angepasst werden, wenn das Paket geladen wird. In einigen Fällen ist allerdings eine weitere Steuerung durch Parameter möglich.

In der Klasse article werden Fußnoten im gesamten Dokument fortlaufend nummeriert, in den Klassen report und book auf Kapitel bezogen. Manchmal ist es jedoch sinnvoller, Fußnoten seitenweise zu nummerieren. Hierzu muss footmisc mit der Option perpage geladen werden. Das Paket footnpag (von Joachim Schrod) bietet dieselbe Funktionalität in einer etwas anderen Implementierung als separates Paket. Eine universelle Variante zur Rücksetzung von Zählern auf Seitenbasis liefert das Paket perpage (siehe Abschnitt 3.2.5 auf Seite 127). Da TeXs Seitenaufbau-Mechanismus asynchron ist, müssen Dokumente immer zweimal bearbeitet werden, um eine korrekte Nummerierung

zu erhalten. Glücklicherweise warnt das Paket mit der Meldung „Rerun to get cross-references right“ vor, wenn die Fußnotennummerierung falsch ist. Das Paket speichert die Information zwischen den einzelnen Läufen in der Datei `.aux`. Nach vielen Bearbeitungsschritten entspricht diese Information möglicherweise nicht mehr ganz den Tatsachen. In solch einem Fall hilft es, die Datei `.aux` zu löschen, damit das Paket die richtige Nummerierung schneller findet.[1]

Bsp. 3-2-7




```
\usepackage[perpage,symbol]{footmisc}
Etwas Text\footnote{Erstens.} mit einer
Fußnote. Mehr\footnote{Zweitens.} Text.
Noch mehr Text.\footnote{Drittens.} Und
noch\footnote{Viertens.} mehr Text.
```

Für diesen speziellen Fall zeigt das obige Beispiel zwei Seiten nebeneinander, auf denen die Auswirkungen von `perpage` deutlich zu sehen sind. Das Beispiel zeigt auch den Effekt einer weiteren Option, und zwar `symbol`. Diese Option verwendet Fußnotenzeichen anstelle von Nummern. Da es nur eine begrenzte Anzahl solcher Textsymbole gibt, kann diese Option nur verwendet werden, wenn es insgesamt nur wenige Fußnoten gibt oder die Fußnotennummerierung auf jeder Seite neu beginnt. Es stehen sechs verschiedene Textsymbole für Fußnoten zur Verfügung, wobei Standard-LaTeX durch Duplikation einiger Symbole bis zu neun Fußnoten bereitstellen kann. Durch Verdreifachung einiger Symbole unterstützt footmisc bis zu 16 Fußnoten (insgesamt bzw. pro Seite). Wenn diese Anzahl überschritten wird, wird eine LaTeX-Fehlermeldung ausgegeben.

Fehler: „Counter too large“

Insbesondere bei der Option `perpage` kann diese Eigenschaft sehr störend sein, weil der Fehler möglicherweise nur dann auftritt, während das Paket noch zu ermitteln versucht, welche Fußnote zu welcher Seite gehört, d.h. er keinen wirklichen Fehler darstellt. Als Ausweichlösung für dieses Problem kann man auch die alternative Option `symbol*` verwenden. Sie erzeugt ebenfalls Fußnotensymbole, aber wenn diese zur Neige gehen, setzt sie die Nummerierung mit arabischen Ziffern fort. In diesem Fall wird am Ende eines Laufs eine Warnung ausgegeben, dass einige Fußnoten nicht mehr im Gültigkeitsbereich liegen, und die Details werden in der Transkriptdatei gespeichert.

`\setfnsymbol{`*name*`}` `\DefineFNsymbols*{`*name*`}[`*typ*`]{`*symbol-liste*`}`

Bei Auswahl von `symbol` oder `symbol*` wird die von Leslie Lamport definierte Standardreihenfolge der Fußnotensymbole verwendet. Andere Autoritäten schlagen andere Reihenfolgen vor; aus diesem Grund bietet footmisc drei weitere Sequenzen zur Auswahl, die mit der Deklaration `\setfnsymbol` verwendet werden können (siehe Tabelle 3.4 auf der nächsten Seite).

[1] In der Tat hat es der Autor dieses Buches geschafft, footmisc während der Bearbeitung dieses Kapitels so sehr durcheinander zu bringen (indem er in einem Beispiel `\textheight` änderte), dass das Paket die richtige Nummerierung hinterher nicht mehr fand und ständig einen neuen Bearbeitungslauf verlangte. Durch Löschen der Datei `.aux` konnte das Problem behoben werden.

lamport	*	†	‡	§	¶	‖	**	††	‡‡	§§	¶¶	***	†††	‡‡‡	§§§	¶¶¶
bringhurst	*	†	‡	§	‖	¶										
chicago	*	†	‡	§	‖	#										
wiley	*	**	†	‡	§	¶	‖									

Tabelle 3.4: Von footmisc vordefinierte Listen von Fußnotensymbolen

Zusätzlich können in der Präambel mit der `\DefineFNsymbols`-Deklaration eigene Sequenzen definiert werden. Die Deklaration hat zwei obligatorische Argumente: den *name*n, mit dem später per `\setfnsymbol` auf die Liste zugegriffen werden kann, und die *symbol-liste*. Aus dieser Liste werden nacheinander Symbole entnommen (wobei Leerzeichen ignoriert werden). Wenn ein Symbol aus mehr als einem Zeichen besteht, muss es in Klammern eingefasst werden. Wenn die Sternform der Deklaration benutzt wird, gibt LaTeX eine Fehlermeldung aus, sobald die Symbole zur Neige gehen. Ohne Stern werden arabische Ziffern benutzt und am Ende des LaTeX-Laufs wird eine Warnung ausgegeben.

Aufgrund einer unglücklichen Designentscheidung wurden Fußnotensymbole (genau wie einige andere Textsymbole) ursprünglich in den mathematischen Zeichensätzen von TeX untergebracht, anstatt in den Textschriften. Das Ergebnis war, dass sie sich nicht änderten, wenn man die Textschrift änderte. In LaTeX wurde dieser Mangel teilweise behoben, indem diese Symbole zur Textsymbolkodierung (TS1) hinzugefügt wurden; siehe Abschnitt 7.5.4. Aus Kompatibilitätsgründen werden Fußnotensymbole jedoch standardmäßig immer noch aus den mathematischen Fonts genommen, auch wenn diese Wahl nicht sehr glücklich ist, wenn die Textschrift von Computer Modern in eine andere Schrift geändert wird. Mithilfe des Wertes `text` im optionalen Argument *typ* kann man footmisc mitteilen, dass die Liste aus Textsymbolen besteht. Zu beachten ist, dass alle vordefinierten Symbollisten aus mathematischen Symbolen bestehen und gegebenenfalls neu deklariert werden müssen, wenn sie mit anderen Schriften als Computer Modern verwendet werden sollen.

Etwas Text* mit einer Fußnote. Mehr** Text. Noch mehr Text.*** Und noch**** mehr Text. Ein letzter Text.

*Erstens.
**Zweitens.
***Drittens.
****Viertens.

```
\usepackage[symbol]{footmisc}
\DefineFNsymbols{stars}[text]{*{**}{***}{****}}
\setfnsymbol{stars}
Etwas Text\footnote{Erstens.} mit einer Fußnote.
Mehr\footnote{Zweitens.} Text. Noch mehr
Text.\footnote{Drittens.} Und noch\footnote
 {Viertens.} mehr Text. Ein letzter Text.
```

Bsp. 3-2-8

Bei vielen kurzen Fußnoten ist die Standardanordnung untereinander am Fuß der Seite möglicherweise keine wirklich befriedigende Lösung. Ein typisches Beispiel sind kritische Editionen, die viele kurze Fußnoten enthalten.[1]

[1] Die Beschreibung des Paketes ledmac [172] zeigt beispielsweise, welche Art von Fuß- und Endnoten üblicherweise in kritischen Editionen verwendet werden. Dieses Paket ist eine einfachere Version des EDMAC-Systems [114] für LaTeX und wurde kürzlich von Peter Wilson bereitgestellt. Siehe auch das Paket bigfoot von David Kastrup.

Die Anordnung der Fußnoten kann über die Option `para` geändert werden, die dann alle Fußnoten in einen einzigen Absatz platziert. Wenn diese Option gesetzt ist, werden Fußnoten nie auf die nächste Seite umbrochen. Der Code für diese Option basiert auf der Arbeit von Chris Rowley und Dominik Wujastyk (verfügbar als Paket fnpara), die wiederum von einem Beispiel im *TEXbook* von Donald Knuth inspiriert wurde.

Bsp. 3-2-9

Etwas Text mit einer Fußnote.[1] Mehr[2] Text. Noch mehr Text.[3] Ein letzter Text.

[1] Erstens. [2] Zweitens. [3] Drittens.

```
\usepackage[para]{footmisc}
Etwas Text mit einer Fußnote.\footnote{Erstens.}
Mehr\footnote{Zweitens.} Text. Noch mehr
Text.\footnote{Drittens.} Ein letzter Text.
```

Eine weitere Möglichkeit, Fußnoten zu setzen, bietet die Option `side`. Hier werden Fußnoten in den Rand gesetzt, wenn möglich auf dieselbe Zeile, in der auf sie verwiesen wird. Intern werden dabei spezielle `\marginpar`-Befehle verwendet, um den Fußnotentext zu platzieren. Alles, was in Abschnitt 3.2.8 über `\marginpar`-Befehle gesagt wird, gilt also auch hier. Diese Option kann nicht zusammen mit der bereits beschriebenen Option `para` verwendet werden, ist aber mit den meisten anderen kombinierbar.

Bsp. 3-2-10

[1]Erstens.
[2]Zweitens.
[3]Drittens.
[4]Auch noch viertens.

Etwas Text mit einer Fußnote.[1] Ganz viel zusätzlicher Text mit einer Fußnote.[2] Mehr Text und eine weitere Fußnote.[3] Noch mehr Text.[4] Viele zusätzliche Textzeilen, die den Raum auf der linken Seite füllen sol-

```
\usepackage[side,flushmargin]{footmisc}
Etwas Text mit einer Fußnote.\footnote{Erstens.}
Ganz viel zusätzlicher Text mit einer
Fußnote.\footnote{Zweitens.} Mehr Text
und eine weitere Fußnote.\footnote{Drittens.}
Noch mehr Text.\footnote{Auch noch viertens.}
Viele zusätzliche Textzeilen, die den
Raum auf der linken Seite füllen sollen.
```

Bei der Option `flushmargin`, die im vorangegangenen Beispiel verwendet wurde, beginnt der Fußnotentext ohne Einzug, wobei das Fußnotenzeichen in den Rand hineinragt. Der Fußnotentext wird standardmäßig eingezogen. Es ist offensichtlich, dass sich dieses Verhalten nicht mit der Option `para` verträgt. Eine Alternative ist `marginal`. Bei der Verwendung dieser Option reicht das Zeichen noch weiter in den Rand hinein, wie das folgende Beispiel zeigt.

Bsp. 3-2-11

Etwas Text[1] mit einer Fußnote. Mehr Text.[2] Noch mehr Text.[3] Ein letzter Text.

[1] Erstens.
[2] Zweitens.
[3] Drittens.

```
\usepackage[marginal]{footmisc}
Etwas Text\footnote{Erstens.} mit einer
Fußnote. Mehr Text.\footnote{Zweitens.}
Noch mehr Text.\footnote{Drittens.}
Ein letzter Text.
```

Anstatt eine der oben beschriebenen Optionen zu verwenden, kann die Position des Fußnotenzeichens alternativ über den `\footnotemargin`-Parameter gesteuert werden. Bei einem negativen Wert wird es in den Rand gesetzt. Der Wert `0pt` entspricht der Verwendung der Option `flushmargin`. Ein positiver Wert bedeutet, dass der Fußnotentext um diesen Betrag eingezo-

gen wird; das Zeichen wird rechtsbündig in den vom Einzug erzeugten Leerraum gesetzt.

Etwas Text[1] mit einer Fußnote. Mehr[2] Text. Noch mehr Text.[3] Ein letzter Text.

[1] Erstens.
[2] Zweitens.
[3] Drittens.

```
\usepackage{footmisc}
\setlength\footnotemargin{10pt}
Etwas Text\footnote{Erstens.} mit einer
Fußnote. Mehr\footnote{Zweitens.} Text.
Noch mehr Text.\footnote{Drittens.}
Ein letzter Text.
```

Bsp. 3-2-12

Standardmäßig wird der Fußnotentext mit Randausgleich gesetzt, aber dies führt nicht immer zu zufriedenstellenden Ergebnissen, insbesondere nicht mit den Optionen `para` und `side`. Bei `para` lässt sich das nicht ändern, für andere Layouts kann man mit der Option `ragged` auf Flattersatz umschalten. Das nächste Beispiel verwendet nicht `flushmargin`, deshalb wird hier ein Einzug der Breite `\footnotemargin` erzeugt – vgl. hierzu Beispiel 3-2-10 auf der vorherigen Seite.

Etwas Text mit einer Fußnote.[1] Ganz viel zusätzlicher Text, um den Leerraum im Beispiel hier zu füllen. Ganz viel zusätzlicher Text, um den Leer-

[1] Im Rand sieht Flattersatz oft besser aus.

```
\usepackage[side,ragged]{footmisc}
Etwas Text mit einer Fußnote.\footnote{Im Rand
sieht Flattersatz oft besser aus.}
Ganz viel zusätzlicher Text, um den Leerraum
im Beispiel hier zu füllen. Ganz viel
zusätzlicher Text, um den Leerraum im Beispiel
hier zu füllen.
```

Bsp. 3-2-13

Die beiden Optionen `norule` und `splitrule` (Dank an Donald Arseneau) ermöglichen eine Modifizierung der Linie, die zwischen Text und Fußnoten steht. Bei Verwendung von `norule` wird die Trennlinie unterdrückt. Als Ausgleich wird der Wert von `\skip\footins` leicht erhöht. Wenn eine Fußnote nicht auf die aktuelle Seite passt, wird sie geteilt und auf der nächsten Seite fortgesetzt, außer wenn die Option `para` verwendet wird (sie unterstützt keine Teilung von Fußnoten). Standardmäßig ist die Linie, die übernommene Fußnoten der vorherigen Seite und Fußnoten der aktuellen Seite vom Text trennt, ein und dieselbe. Mit der Option `splitrule` lässt sich dies jedoch anpassen: Die Linie über geteilten Fußnoten geht über die gesamte Spalte, während die Linie über den normalen Fußnoten die Standarddefinition von `\footnoterule` behält. Genauer gesagt führt diese Option die Befehle `\mpfootnoterule` (zur Verwendung auf Miniseiten), `\pagefootnoterule` (zur Verwendung auf normalen Seiten) und `\splitfootnoterule` (zur Verwendung auf Seiten, die mit einer geteilten Fußnote beginnen) ein. Durch Modifikation ihrer Definitionen kann man das Layout an die eigenen Bedürfnisse anpassen – ähnlich wie es bereits am Beispiel des Befehls `\footnoterule` gezeigt wurde.

Etwas Text mit einer Fußnote.[1] Mehr[2] Text. Noch mehr Text.[3] Ein letzter Text.

[1] Erste. [2] Eine zweite. [3] Eine dritte.

```
\usepackage[norule,para]{footmisc}
Etwas Text mit einer Fußnote.\footnote{Erste.}
Mehr\footnote{Eine zweite.} Text.  Noch mehr
Text.\footnote{Eine dritte.}  Ein letzter Text.
```

Bsp. 3-2-14

In Klassen wie article oder report, in denen `\raggedbottom` wirksam ist (was zur Folge hat, dass Spalten unterschiedlich hoch sein können), wer-

den Fußnoten mit einem Abstand von `\skip\footins` an den Spaltentext angehängt. Alternativ können sie auch am Fuß der Spalte platziert werden, wobei hier zwischen Text und Fußnote zusätzlicher Leerraum eingefügt werden muss; dies geht mit der Option `bottom`. In Klassen wie book, in denen `\flushbottom` aktiv ist, bleibt diese Option ohne Wirkung.

In einigen Dokumenten, z.B. literarischen Analysen, kann es vorkommen, dass an einer Stelle mehrere Fußnoten gesetzt werden. Leider sind die Standardfußnotenbefehle von LaTeX nicht in der Lage, mit dieser Situation umzugehen; die Fußnotenzeichen werden einfach hintereinander aufgereiht, so dass es nicht mehr unterscheidbar ist, ob es sich um die Fußnoten 1 und 2 oder die Fußnote 12 handelt.

Bsp. 3-2-15

Etwas Text[12] mit zwei Fußnoten. Noch mehr Text.[3]

[1] Erste. [2] Zweite. [3] Dritte.

```
\usepackage[para]{footmisc}
Etwas Text\footnote{Erste.}\footnote{Zweite.} mit
zwei Fußnoten.  Noch mehr Text.\footnote{Dritte.}
```

Dieses Problem lässt sich mithilfe der Option `multiple` lösen. Sie stellt sicher, dass mehrere aufeinander folgende Fußnotenzeichen durch Kommas getrennt werden. Das Trennzeichen kann auch etwas Anderes sein, wie ein kleiner Leerraum. Die Auswahl des Trennzeichens geschieht über den Befehl `\multfootsep`.

Bsp. 3-2-16

Etwas Text[1,2] mit zwei Fußnoten. Noch mehr Text.[3]

[1] Erste. [2] Zweite. [3] Dritte.

```
\usepackage[multiple,para]{footmisc}
Etwas Text\footnote{Erste.}\footnote{Zweite.} mit
zwei Fußnoten.  Noch mehr Text.\footnote{Dritte.}
```

Das Paket footmisc beschäftigt sich mit einem anderen potentiellen Problem: Wenn Fußnoten in eine Überschrift gesetzt werden, erscheinen sie möglicherweise im Inhaltsverzeichnis oder dem Kolumnentitel, was ungeahnte Folgen haben kann. Natürlich könnte man dieses Problem (manuell) umgehen, indem man das optionale Argument des Gliederungsbefehls verwendet. Oder man gibt die Option `stable` an: Sie verhindert, dass Fußnoten an solchen Stellen erscheinen.

3.2.5 perpage – Zurücksetzen des Zählers auf Seitenbasis

Wie bereits erwähnt, wird die Möglichkeit, Zähler auf Seitenbasis zurückzusetzen, mit dem kleinen Paket perpage von David Kastrup eröffnet.

`\MakePerPage[`*start*`]{`*zähler*`}`

Mit der Deklaration `\MakePerPage` wird der *zähler* auf jeder Seite zurückgesetzt, wobei optional der Startwert *start* (voreingestellt auf 1) definiert werden kann. Zur Verdeutlichung wird hier noch einmal das Beispiel 3-2-7 von Seite 123 aufgenommen, wobei der Beginn einer Fußnotensequenz diesmal

mit dem zweiten Symbol (d.h. „†“ anstatt „*“) gekennzeichnet ist.

Etwas Text[†] mit einer Fußnote. Mehr[‡]

[†]Erstens.
[‡]Zweitens.

Text. Noch mehr Text.[†] Und noch[‡]

[†]Drittens.
[‡]Viertens.

```
\usepackage[symbol]{footmisc}
\usepackage{perpage}
\MakePerPage[2]{footnote}
Etwas Text\footnote{Erstens.} mit einer
Fußnote. Mehr\footnote{Zweitens.} Text.
Noch mehr Text.\footnote{Drittens.} Und
noch\footnote{Viertens.} mehr Text.
```

Bsp. 3-2-17

Das Paket synchronisiert die Nummerierung über die `.aux`-Datei des Dokumentes. Dadurch müssen mindestens zwei Kompilierungsläufe durchgeführt werden, bevor die Nummerierung richtig ist. Außerdem kann es vorkommen, dass im ersten Lauf überflüssige Fehlermeldungen „Counter too large“ ausgegeben werden, wenn zur Nummerierung `\fnsymbol` oder `\alph` verwendet wird (siehe die Diskussion der Option `symbol*` für das footmisc-Paket auf Seite 123).

Von den LaTeX-Standardzählern lässt sich wahrscheinlich nur der Zähler `footnote` sinnvoll auf diese Art manipulieren. Man kann sich aber durchaus Anwendungen vorstellen, welche z.B. nummerierte Marginalien verwenden, die etwa wie folgt definiert sind:

```
\newcounter{mnote}
\newcommand\mnote[1]{{\refstepcounter{mnote}%
  \marginpar[\itshape\small\raggedleft\themnote.\ #1]%
            {\itshape\small\raggedright\themnote.\ #1}}}
\usepackage{perpage} \MakePerPage{mnote}
```

Der neue Zähler `mnote` wird außerhalb von `\marginpar` hochgesetzt, so dass er nur einmal ausgeführt wird.[1] Außerdem muss der Gültigkeitsbereich für die aktuelle Neudefinition von `\label` (durch `\refstepcounter`) begrenzt werden, d.h. die komplette Definition wird eingeklammert. Anmerkungen auf linken Seiten sollten rechtsbündig sein, was mithilfe des optionalen Argumentes von `\marginpar` realisiert wird.

1. Eins Etwas Text mit einer Fußnote.[1] Mehr Text. Noch

[1]Zwei.

mehr Text. Und *1. Drei!* noch mehr Text. *2. Vier!* Ein letzter Text.[2]

[2]Fünf!

```
% Code wie oben
Etwas Text\mnote{Eins} mit
einer Fußnote.\footnote{Zwei.}
Mehr Text. Noch mehr
Text.\mnote{Drei!} Und noch
mehr\mnote {Vier!} Text. Ein
letzter Text.\footnote{Fünf!}
```

Bsp. 3-2-18

Eine weitere Anwendung für dieses Paket zeigt das Beispiel 3-2-24 auf Seite 132, in dem mehrere, voneinander unabhängige Fußnotensequenzen jeweils auf Seitenbasis nummeriert werden.

[1]Wenn er in beiden Argumenten von `\marginpar` verwendet wird, wird er zweimal ausgeführt. Es funktioniert auch, wenn man den Zähler nur im optionalen Argument setzt, dies jedoch liegt an der speziellen Implementierung (bei welcher das optionale Argument zuerst ausgewertet wird), die sich gegebenenfalls ändern kann.

3.2.6 manyfoot – Unabhängige Fußnoten

Die meisten Dokumente haben, wenn überhaupt, nur wenige Fußnoten. Für diese reichen LaTeXs Standardbefehle und die Erweiterungen aus, die das Paket footmisc bietet. Einige Anwendungen, wie kritische Editionen, benötigen allerdings mehrere, unabhängig nummerierte Fußnotensequenzen. Für diese Fälle ist das Paket manyfoot von Alexander Rozhenko sehr hilfreich.[1]

`\DeclareNewFootnote[`*fußnotenstil*`]{`*suffix*`}[`*nummerierungsstil*`]`

Diese Deklaration kann zur Einführung einer neuen Fußnotenebene verwendet werden. In ihrer einfachsten Form benötigt sie lediglich ein *suffix*, wie „B“. Dadurch wird ein neuer Zähler `footnote`⟨*suffix*⟩ reserviert, der zur automatischen Nummerierung der Fußnoten auf dieser Ebene dient. Standardmäßig werden hierfür arabische Ziffern verwendet. Mithilfe des optionalen Argumentes *nummerierungsstil* können andere Zählerstile (z.B. `roman` oder `alph`) ausgewählt werden.

Das optionale Argument *fußnotenstil* definiert den Stil, der im Allgemeinen für Fußnoten der neuen Ebene benutzt wird; der voreingestellte Wert ist `plain`. Wenn das Paket mit der Option `para` bzw. `para*` geladen wurde, steht `para` auch als Fußnotenstil zur Verfügung.

Die Deklaration definiert dann automatisch sechs Befehle. Die ersten drei haben folgende Bedeutung:

`\footnote`⟨*suffix*⟩`[`*nummer*`]{`*text*`}` Wie `\footnote`, aber für die neue Ebene. Erhöht den Zähler `footnote`⟨*suffix*⟩, falls das optionale Argument *nummer* nicht angegeben ist. Erzeugt Fußnotenzeichen und setzt den *text* an den Fuß der Seite.

`\footnotemark`⟨*suffix*⟩`[`*nummer*`]` Wie `\footnotemark`, aber für die neue Ebene. Erhöht den entsprechenden Zähler (bei fehlendem optionalen Argument) und druckt das seinem Wert entsprechende Fußnotenzeichen.

`\footnotetext`⟨*suffix*⟩`[`*nummer*`]{`*text*`}` Wie `\footnotetext`, aber für die neue Ebene. Setzt den *text* an den Fuß der Seite, wobei er zum Erzeugen des Fußnotenzeichens den aktuellen Wert von `footnote`⟨*suffix*⟩ oder des optionalen Argumentes verwendet.

In allen drei Fällen hängt der Stil des Zeichens von dem gewählten *nummerierungsstil* ab.

Die übrigen drei Befehle, die durch eine `\DeclareNewFootnote`-Deklaration zur Verwendung im Dokument definiert werden, heißen `\Footnote`⟨*suffix*⟩, `\Footnotemark`⟨*suffix*⟩ und `\Footnotetext`⟨*suffix*⟩ (d.h. dieselben Namen wie oben, aber mit großem Anfangsbuchstaben `F`). Der wichtige Unterschied zu den vorherigen Befehlen ist folgender: Anstelle des optionalen Argumentes *nummer* erwarten sie ein obligatorisches *marker*-Argument, das die Verwendung beliebiger Fußnotenzeichen ermöglicht. Einige Beispiele sind unten aufgeführt.

Das Layout der Fußnoten kann verändert werden, indem man zusätzlich zu manyfoot das Paket footmisc lädt. Die Option `para` von footmisc kann jedoch nicht verwendet werden. Im nächsten Beispiel wird das Standardfußno-

[1] Ein umfassenderes Paket, bigfoot, wird gerade von David Kastrup entwickelt.

tenlayout für Fußnoten der höchsten Ebene verwendet und das eingezogene Layout (Option para) für die zweite Ebene. Wenn alle Fußnotenebenen eingezogen werden sollen, müssen Fußnoten höchster Ebene (z.B. \footnote) vermieden werden. Alle benötigten Ebenen müssen dann über manyfoot definiert werden. Man beachte, dass footmiscs Option multiple alle Fußnoten richtig behandelt.

```
\usepackage[multiple]{footmisc}
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[alph]
Etwas Text\footnote{Eine erste.}%
          \footnoteB{Ebene 2.}
mit Fußnoten. Noch mehr
Text.\footnoteB{Eine zweite.} Etwas
Text\footnote{Noch eine Hauptfußnote.}%
    \FootnoteB{*}{Ein manuelles
                  Fußnotenzeichen.}
mit Fußnoten. Noch mehr
Text.\footnoteB{Noch eine Fußnote
                zweiter Ebene.}
```

Etwas Text[1,a] mit Fußnoten. Noch mehr Text.[b] Etwas Text[2,*] mit Fußnoten. Noch mehr Text.[c]

[1] Eine erste.
[2] Noch eine Hauptfußnote.

[a] Ebene 2. [b] Eine zweite. [*] Ein manuelles Fußnotenzeichen. [c] Noch eine Fußnote zweiter Ebene.

Bsp. 3-2-19

Im folgenden Beispiel werden die Fußnoten erster Ebene in den Rand verschoben, indem footmisc mit anderen Optionen geladen wird. Dieses Mal wird manyfoot mit der Option para* geladen. Sie unterscheidet sich von der zuvor verwendeten Option para dadurch, dass sie Einzüge für den eingebetteten Fußnotenblock unterdrückt. Außerdem werden Fußnoten der zweiten Ebene jetzt mit römischen Ziffern nummeriert. Zum Vergleich wird im nächsten Beispiel derselbe Text wie im Beispiel 3-2-19 ausgegeben, es wird jedoch ein anderer Satzspiegel verwendet, damit die Marginalien angezeigt werden können.

```
\usepackage[side,flushmargin,ragged,multiple]
           {footmisc}
\usepackage[para*]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Etwas Text\footnote{Eine erste.}%
          \footnoteB{Ebene 2.}
mit Fußnoten. Noch mehr
Text.\footnoteB{Eine zweite.} Etwas
Text\footnote{Noch eine Hauptfußnote.}%
    \FootnoteB{*}{Ein manuelles
                  Fußnotenzeichen.}
mit Fußnoten. Noch mehr
Text.\footnoteB{Noch eine Fußnote
                zweiter Ebene.}
```

[1] Eine erste.

[2] Noch eine Hauptfußnote.

Etwas Text[1,i] mit Fußnoten. Noch mehr Text.[ii] Etwas Text[2,*] mit Fußnoten. Noch mehr Text.[iii]

[i] Ebene 2. [ii] Eine zweite. [*] Ein manuelles Fußnotenzeichen. [iii] Noch eine Fußnote zweiter Ebene.

Bsp. 3-2-20

Bei eingebetteten Fußnoten, die mit der Option para bzw. para* erzeugt werden, tritt wahrscheinlich folgender Fehler auf: Sehr lange Fußnoten am Ende der Seite, die aufgrund ihrer Länge umbrochen werden müssten, werden nicht geteilt. Zur Lösung dieses Problems bietet das Paket manyfoot die (halb) manuelle Möglichkeit, an der Stelle, an der die Fußnote umbrochen werden soll, den Befehl \SplitNote einzufügen und die Fußnote zu beenden.

Der restliche Text der Fußnote wird dann in einem späteren Absatz des Dokumentes über ein leeres *marker*-Argument in einen `\Footnotetext⟨suffix⟩` platziert.

Bsp. 3-2-21

Etwas[1] Text mit zwei Fußnoten.[i] Mehr Text.[ii] Noch mehr Text.
Etwas Text hier und[2] noch mehr dort.

[1] Eine erste.
[2] Noch eine erste.
[i] Eine zweite. [ii] Dies ist eine sehr lange Fußnote, die hier fortgesetzt wird.

```
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Etwas\footnote{Eine erste.} Text mit zwei
Fußnoten.\footnoteB{Eine zweite.} Mehr
Text.\footnoteB{Dies ist eine sehr
lange Fußnote,\SplitNote} Noch mehr Text.

Etwas\FootnotetextB{}{die hier
                     fortgesetzt wird.}
Text hier und\footnote{Noch eine erste.}
noch mehr dort.
```

Wenn beide Teile der Fußnote nach einer erneuten Formatierung des Dokumentes auf derselben Seite stehen, werden sie richtig zusammengesetzt, wie das nächste Beispiel zeigt. Dieses verwendet denselben Beispieltext, jedoch einen anderen Satzspiegel. Wenn es jedoch nach einer erneuten Formatierung erforderlich ist, die Fußnote an einer anderen Stelle zu umbrechen, dann ist weiteres manuelles Eingreifen unvermeidlich. Diese Arbeit sollte daher erst zum Schluss durchgeführt werden.

Bsp. 3-2-22

Etwas[1] Text mit zwei Fußnoten.[i] Mehr Text.[ii] Noch mehr Text.
Etwas Text hier und[2] noch mehr dort.

[1] Eine erste.
[2] Noch eine erste.
[i] Eine zweite. [ii] Dies ist eine sehr lange Fußnote, die hier fortgesetzt wird.

```
\usepackage[para]{manyfoot}
\DeclareNewFootnote[para]{B}[roman]
Etwas\footnote{Eine erste.} Text mit zwei
Fußnoten.\footnoteB{Eine zweite.} Mehr
Text.\footnoteB{Dies ist eine sehr
lange Fußnote,\SplitNote} Noch mehr Text.

Etwas\FootnotetextB{}{die hier
                     fortgesetzt wird.}
Text hier und\footnote{Noch eine erste.}
noch mehr dort.
```

Der vertikale Abstand zwischen zwei untereinander stehenden Fußnotenblöcken wird durch `\skip\footins⟨suffix⟩` festgelegt. Standardmäßig entspricht er `\skip\footins` (d.h. dem Abstand zwischen dem Haupttext und den Fußnoten). Ohne nähere Spezifikation werden die Fußnotenblöcke nur durch solche Leerräume voneinander getrennt, aber mit der Option `ruled` wird auch hier `\footnoterule` angewendet. Genau genommen kann beliebiges Material an dieser Stelle platziert werden, indem man den Befehl `\extrafootnoterule` neu definiert – die einzige Beschränkung ist, dass das formatierte Ergebnis dieses Befehls keinen zusätzlichen vertikalen Leerraum beanspruchen darf (Einzelheiten hierzu in der Erörterung von `\footnoterule` auf Seite 119). Es ist sogar möglich, für verschiedene Fußnotenblöcke unterschiedliche Linien zu definieren (die Einzelheiten hierzu findet man in der Paketdokumentation).

Etwas[1,*] Text mit einer Fußnote. Noch[A] mehr Text. Etwas Text[†] mit einer Fußnote.[B] Noch mehr Text für das Beispiel.

[1] Eine erste.

[*] Eine zweite.
[†] Ein Beispiel.

[A] Eine dritte.
[B] Ein weiteres Beispiel.

```
\usepackage[marginal,multiple]{footmisc}
\usepackage[ruled]{manyfoot}
\DeclareNewFootnote{B}[fnsymbol]
\DeclareNewFootnote{C}[Alph]
\setlength{\skip\footinsB}{5pt minus 1pt}
\setlength{\skip\footinsC}{5pt minus 1pt}
Etwas\footnote{Eine erste.}\footnoteB{Eine zweite.}
Text mit einer Fußnote. Noch\footnoteC{Eine dritte.}
mehr Text. Etwas Text\footnoteB{Ein Beispiel.} mit
einer Fußnote.\footnoteC{Ein weiteres Beispiel.}
Noch mehr Text für das Beispiel.
```

Bsp. 3-2-23

Seitenweise Nummerierung der Fußnoten

Das vorige Beispiel verwendete zwei zusätzliche *nummerierungsstil*e, `Alph` und `fnsymbol`. Da in beiden Stilen nur wenige Fußnotensymbole verfügbar sind, ist diese Wahl meist nicht empfehlenswert, es sei denn, dass diese Fußnotensequenzen seitenweise nummeriert werden. Die Option `perpage` von `footmisc` hilft in diesem Fall nicht, da sie nur auf Fußnoten der obersten Ebene anwendbar ist. Die gewünschte Wirkung lässt sich erzielen, indem man entweder für die Zähler `footnoteB` und `footnoteC` `\MakePerPage` aus dem Paket `perpage` verwendet (wie unten) oder die Option `perpage` von `manyfoot` einsetzt (welches dazu das Paket `perpage` aufruft. Dadurch werden alle neu definierten Fußnotenebenen seitenweise nummeriert.). Zu beachten ist, dass die Fußnoten der höchsten Ebene, wie im Beispiel, immer noch sequentiell nummeriert werden.

Etwas Text[1] mit einer Fußnote. Noch mehr[*,A] Text. Etwas Text mit[B] einer Fußnote hier.[A] Mehr Text. Und[2,*] eine letzte Fußnote.

[1] Eine erste.

[*] Zweitens.

[A] Drittens.
[B] Ein Beispiel.

[2] Wieder.

[*] Eine letzte.

[A] Ein weiteres Beispiel.

```
\usepackage[multiple]{footmisc}
\usepackage{manyfoot,perpage}
\DeclareNewFootnote{B}[fnsymbol]
\DeclareNewFootnote{C}[Alph]
\MakePerPage{footnoteB}\MakePerPage{footnoteC}
Etwas Text\footnote{Eine erste.} mit einer
Fußnote. Noch mehr\footnoteB{Zweitens.}%
  \footnoteC{Drittens.} Text. Etwas Text
mit\footnoteC{Ein Beispiel.} einer Fußnote
hier.\footnoteC{Ein weiteres Beispiel.} Mehr
Text. Und\footnote{Wieder.}\footnoteB{Eine
  letzte.} eine letzte Fußnote.
```

Bsp. 3-2-24

3.2.7 endnotes – Eine Alternative zu Fußnoten

In geisteswissenschaftlichen Arbeiten werden Anmerkungen üblicherweise am Ende jedes Kapitels oder am Ende des Dokumentes zu sogenannten Endnoten zusammengefasst. Standard-LaTeX unterstützt zunächst keine Endnoten. Sie können aber auf verschiedene Weise erzeugt werden.

Das Paket `endnotes` (von John Lavagnino) enthält einen eigenen `\endnote`-Befehl, der den gleichzeitigen Gebrauch von Fußnoten und Endnoten ermöglicht.

Die Syntax auf Dokumentenebene ist den Fußnotenbefehlen nachgebildet, wenn `foot` durch `end` ersetzt wird – so erzeugt `\endnote` beispielsweise

eine Endnote, \endnotemark nur das Endnotenzeichen und \endnotetext nur den Text. Der Zähler, der die aktuelle Endnotennummer wiedergibt, heißt endnote. Er wird hochgezählt, wenn \endnote oder \endnotemark ohne optionales Argument verwendet wird.

Alle Endnoten werden in einer externen Datei mit der Erweiterung .ent gespeichert. Sie werden bereitgestellt, sobald der Befehl \theendnotes ausgegeben wird.

Bsp. 3-2-25

Dies ist[1] ein einfacher Text. Dies ist ein einfacher Text.[2] Noch mehr Text mit einem Endnotenzeichen.[1]

Notes

[1]Die erste Endnote.
[2]Die zweite Endnote.

```
\usepackage{endnotes}
Dies ist\endnote{Die erste Endnote.} ein
einfacher Text. Dies ist ein einfacher
Text.\endnote{Die zweite Endnote.} Noch mehr
Text mit einem Endnotenzeichen.\endnotemark[1]

\theendnotes   % Endnoten hier ausgeben
```

Dieser Prozess unterscheidet sich von der Art und Weise, wie Inhaltsverzeichnisse generiert werden (siehe Abschnitt 2.3). Die Endnoten werden direkt in die Datei geschrieben, so dass nur solche Endnoten sichtbar sind, die früher im Dokument definiert wurden. Der Vorteil dieses Ansatzes ist, dass \theendnotes mehrfach aufgerufen werden kann, z.B. am Ende jeden Kapitels. Um zusätzlich die Nummerierung neu zu beginnen, muss der Zähler endnote nach dem Aufruf von \theendnotes auf null gesetzt werden.

Es gibt verschiedene Möglichkeiten, die von \theendnotes erzeugte Überschrift zu steuern. Der Text kann durch Anpassung von \notesname geändert werden (voreingestellt ist die Zeichenkette Notes). Wenn dies nicht ausreicht, kann auch noch \enoteheading neu definiert werden, der den Gliederungsbefehl vor der Endnote angibt.

Das Layout für die Endnotennummern wird über \theendnote gesteuert, der üblichen Art, die Formatierung von Zählern festzulegen. Das Format des Endnotenzeichens wird von \makeenmark mithilfe von \theenmark erzeugt, letzteres Makro enthält die formatierte Nummer für das aktuelle Endnotenzeichen.

Bsp. 3-2-26

Dies ist ein einfacher Text.[a)] Dies ist ein einfacher Text.[b)] Noch mehr Text mit einem Endnotenzeichen.[a)]

Kapitelanmerkungen

[a)]Die erste Endnote.
[b)]Die zweite Endnote.

```
\usepackage{endnotes}
\renewcommand\theendnote{\alph{endnote}}
\renewcommand\makeenmark{\textsuperscript{\theenmark)}}
\renewcommand\notesname {Kapitelanmerkungen}
Dies ist ein einfacher Text.\endnote{Die erste Endnote.}
Dies ist ein einfacher Text.\endnote{Die zweite Endnote.}
Noch mehr Text mit einem Endnotenzeichen.\endnotemark[1]
\theendnotes
```

Die Schriftgröße für die Liste der Endnoten wird über den Befehl \enotesize gesteuert, der auf \footnotesize voreingestellt ist. Durch Veränderung von \enoteformat kann man die Darstellung der einzelnen Endnoten in ihrer Liste verändern. Dieser Befehl dient dazu, die Absatzparameter

für die Endnoten einzustellen und die in \theenmark gespeicherte Endnotennummer zu setzen. Im nächsten Beispiel erhält der erste Absatz keinen Einzug, und die Nummer wird in den Seitenrand geschrieben.

Dies ist ein einfacher Text.[1] Dies ist ein einfacher Text.[2] Noch mehr Text mit einem Endnotenzeichen.[1]

Notes

1. Die erste Endnote hat sehr viel Text, damit zwei Zeilen erzeugt werden.
   Und sogar noch ein zweiter Absatz.
2. Die zweite Endnote.

```
\usepackage{endnotes}
\renewcommand\enoteformat{\noindent\raggedright
 \setlength\parindent{12pt}\makebox[0pt][r]{\theenmark.\,}}
\renewcommand\enotesize{\scriptsize}
Dies ist ein einfacher Text.\endnote{Die erste Endnote hat
 sehr viel Text, damit zwei Zeilen erzeugt werden.\par Und
 sogar noch ein zweiter  Absatz.}
Dies ist ein einfacher Text.\endnote{Die zweite Endnote.}
Noch mehr Text mit einem Endnotenzeichen.\endnotemark[1]
\theendnotes
```

Bsp. 3-2-27

3.2.8 Marginalien

Der LaTeX-Befehl \marginpar erzeugt eine Marginalie. Er setzt den Text, den er als Argument enthält, in den Seitenrand, wobei die erste Zeile in der gleichen Höhe erscheint wie der \marginpar-Befehl im Haupttext. Wenn nur das obligatorische Argument angegeben ist, erscheint der Text bei einseitigem Druck im rechten Rand; bei zweiseitigem Druck erscheint er im äußeren Rand und bei zweispaltigem Druck erscheint er im am nächsten gelegenen Rand. Ist ein optionales Argument angegeben, wird dieses für den linken Rand verwendet, während das obligatorische Argument dann für den rechten Rand gilt.

Diese Voreinstellung lässt sich mit \reversemarginpar umkehren (außer bei zweispaltiger Formatierung). Dieser Befehl wirkt sich auf alle nachfolgenden Marginalien aus. Zurück zur Voreinstellung geht es wieder mit \normalmarginpar.

Beim Arbeiten mit Marginalien sind einige wichtige Punkte zu berücksichtigen. Zunächst beginnt der Befehl \marginpar keinen neuen Absatz. Wenn er also vor dem ersten Wort eines Absatzes erscheint, stimmt die vertikale Ausrichtung nicht mit dem Beginn des Absatzes überein. Außerdem wird das erste Wort im Argument nicht automatisch getrennt. Wenn die Ränder schmal und die Wörter lang sind (wie in der deutschen Sprache), kann es daher erforderlich sein, dem ersten Wort den Befehl \hspace{0pt} voranzustellen, damit es getrennt werden kann. Diese beiden möglichen Schwierigkeiten kann man umgehen, indem man einen Befehl wie \marginlabel definiert, der mit einer leeren Box \mbox{} beginnt, Marginalien linksbündig formatiert und dem Argument den Befehl \hspace{0pt} voranstellt.

Etwas Text mit einer Marginalie. Ein weiterer Text mit einer Marginalie. Noch mehr Text. Eine Menge zusätzlicher Text hier, der im Beispiel die Kolumne füllen soll.

EinSuperlangesErstesWort mit Problemen

EinSuperlangesErstesWort ohne Probleme

```
\newcommand\marginlabel[1]{\mbox{}\marginpar
   {\raggedright\hspace{0pt}#1}}
Etwas\marginpar{EinSuperlangesErstesWort mit
Problemen} Text mit einer Marginalie. Ein
weiterer\marginlabel{EinSuperlangesErstesWort
 ohne Probleme} Text mit einer Marginalie. Noch
mehr Text. Eine Menge zusätzlicher Text hier,
der im Beispiel die Kolumne füllen soll.
```

Bsp. 3-2-28

Natürlich kann die obige Definition keine unterschiedlichen Texte mehr erzeugen, die je nach Randeinstellung variieren. Mit einem kleinen Trick kann dieses Problem gelöst werden, indem man beispielsweise `\ifthenelse`-Konstrukte aus dem Paket ifthen verwendet.

Falsch platzierte `\marginpar`s

Der LaTeX-Kernel bemüht sich (ohne bei der Verarbeitung einen zu großen Overhead zu erzeugen) sicherzustellen, dass der Inhalt des `\marginpar`-Befehls stets im richtigen Rand erscheint, und in den meisten Fällen gelingt ihm das auch. Manchmal geht es allerdings schief. Wenn solch ein Missgeschick auftaucht, ist eine Sofortlösung, einen expliziten `\pagebreak` hinzuzufügen, um eine zu weit vorausschauende Seitenerstellung zu verhindern. Natürlich hat dies den Nachteil, dass diese rein optische Korrektur bei späteren Änderungen im Dokument rückgängig gemacht werden muss. Besser ist, das Paket mparhack von Tom Sgouros und Stefan Ulrich zu verwenden. Sobald das Paket geladen ist, werden alle `\marginpar`-Positionen beobachtet (wobei intern ein Markierungsverfahren verwendet wird und die Daten in die `.aux`-Datei geschrieben werden). In diesem Fall wird die Warnung „Marginpars may have changed. Rerun to get them right“ ausgegeben. Sie weist darauf hin, dass die Positionen im Vergleich zum vorherigen LaTeX-Kompilierungslauf geändert wurden und ein weiterer Lauf erforderlich ist, um das Dokument zu stabilisieren.

Wie in Tabelle 4.2 auf Seite 205 erläutert, wird der Stil von Marginalien durch drei Parameter festgelegt: `\marginparwidth`, `\marginparsep` und `\marginparpush`.

3.3 Listen

Listen sind ein sehr wichtiges LaTeX-Konstrukt. Sie werden zum Bilden vieler der absatzerzeugenden Umgebungen von LaTeX verwendet. Die drei Standardlistenumgebungen von LaTeX werden im Abschnitt 3.3.1 beschrieben, in dem auch gezeigt wird, wie sie angepasst werden können. Abschnitt 3.3.2, der auf Seite 139 beginnt, stellt das Paket paralist im Detail vor, das eine Reihe neuer Listenstrukturen enthält und umfangreiche Methoden zur Anpassung dieser neuen Listen sowie der Standardlisten bietet. Anschließend folgt eine Erörterung von Listen mit Überschriften, wie Theoreme und Übungen. Zum Schluss wird LaTeXs generische Listenumgebung in Abschnitt 3.3.4 auf Seite 152 beschrieben.

3.3.1 Ändern der Standardlisten

Es ist relativ einfach, die drei Standardlistenumgebungen von LaTeX `itemize`, `enumerate` und `description` eigenen Erfordernissen anzupassen. Die drei folgenden Abschnitte befassen sich nacheinander mit diesen Umgebungen. Änderungen an den Standardeinstellungen lassen sich global in der Präambel vornehmen, indem man dort bestimmte Listenparameter neu definiert. Sie können aber auch lokal durchgeführt werden.

Anpassen der `itemize`-Listenumgebung

Die Label für eine einfache `itemize`-Liste ohne Nummerierung werden durch die Befehle in Tabelle 3.5 auf der nächsten Seite definiert. Eine Liste mit anders

	Befehl	*Voreinstellung*	*Darstellung*
Erste Ebene	`\labelitemi`	`\textbullet`	•
Zweite Ebene	`\labelitemii`	`\normalfont\bfseries \textendash`	–
Dritte Ebene	`\labelitemiii`	`\textasteriskcentered`	*
Vierte Ebene	`\labelitemiv`	`\textperiodcentered`	·

Tabelle 3.5: Befehle zum Steuern der Listenumgebung `itemize`

aussehenden Labels erhält man, indem man die labelerzeugenden Befehle neu definiert. Die Änderungen können, wie im nächsten Beispiel, lokal für eine Liste vorgenommen oder global bereitgestellt werden. Dafür wird die neue Definition in die Präambel eingefügt. Bei der folgenden einfachen Liste handelt es sich um eine `itemize`-Standardliste mit einem Symbol aus dem PostScript-Zeichensatz Zapf Dingbats (siehe Abschnitt 7.6.4 auf Seite 390) für das Label der obersten Ebene:

```
\usepackage{pifont}
\newenvironment{MYitemize}{\renewcommand\labelitemi
          {\ding{43}}\begin{itemize}}{\end{itemize}}
\begin{MYitemize}
\item Text des ersten Punktes in der Liste.
\item Text des ersten Satzes für den zweiten
      Punkt der Liste. Und der zweite Satz.
\end{MYitemize}
```

Bsp. 3-3-1

- Text des ersten Punktes in der Liste.
- Text des ersten Satzes für den zweiten Punkt der Liste. Und der zweite Satz.

Anpassen der `enumerate`-Listenumgebung

Die LaTeX-Umgebung `enumerate` für nummerierte Listen wird durch die Befehle und Darstellungsformen in Tabelle 3.6 auf der nächsten Seite charakterisiert. Die erste Zeile enthält die Namen der Zähler, die zum Nummerieren der vier möglichen Listenebenen verwendet werden. Die zweite und dritte Zeile enthalten die Befehle, welche die Darstellung der Zähler festlegen, und ihre voreingestellten Definitionen in den LaTeX-Standardklassen. Die Zeilen vier, fünf und sechs enthalten den Befehl, die Standarddefinition und ein Beispiel für die tatsächliche Nummerierung, wie sie von der Liste ausgegeben wird.

Ein Verweis auf ein nummeriertes Listenelement wird mithilfe von `\theenumi`, `\theenumii` und weiteren, ähnlichen Befehlen gebildet. Ihnen werden entsprechend die Befehle `\p@enumi`, `\p@enumii` usw. als Präfix vorangestellt. Die letzten drei Zeilen der Tabelle 3.6 auf der gegenüberliegenden Seite zeigen den Befehl und die Standarddefinition für Verweise sowie ein Beispiel ihrer Darstellung. Für korrekte Verweise müssen sowohl die Befehlsdefinitionen für die Darstellung als auch diejenigen zum Bilden der Verweise berücksichtigt werden.

Mithilfe dieser Informationen lassen sich nun leicht eigene nummerierte Listen erstellen.

Das erste Beispiel definiert die Zähler der ersten und zweiten Ebene neu, so dass diese große römische Zahlen und lateinische Buchstaben verwenden.

	Erste Ebene	*Zweite Ebene*	*Dritte Ebene*	*Vierte Ebene*
Zähler	`enumi`	`enumii`	`enumiii`	`enumiv`
Darstellung	`\theenumi`	`\theenumii`	`\theenumiii`	`\theenumiv`
Voreinstellung	`\arabic{enumi}`	`\alph{enumii}`	`\roman{enumiii}`	`\Alph{enumiv}`
Labelfeld	`\labelenumi`	`\labelenumii`	`\labelenumiii`	`\labelenumiv`
Voreinstellung	`\theenumi.`	`(\theenumii)`	`\theenumiii.`	`\theenumiv.`
Beispiel	1., 2.	(a), (b)	i., ii.	A., B.
		Darstellung von Verweisen		
Präfix	`\p@enumi`	`\p@enumii`	`\p@enumiii`	`\p@enumiv`
Voreinstellung	`{}`	`\theenumi`	`\theenumi(\theenumii)`	`\p@enumiii\theenumiii`
Verweisbeispiel	1, 2	1a, 2b	1(a)i, 2(b)ii	1(a)iA, 2(b)iiB

Tabelle 3.6: Befehle zum Steuern der Listenumgebung `enumerate`

Die visuelle Darstellung soll aus dem Wert des Zählers gebildet werden, gefolgt von einem Punkt. Damit kann für `\labelenumi` der voreingestellte Wert aus Tabelle 3.6 verwendet werden.

I. **Einleitung**

A. **Anwendungen**
Motivation zu Forschung und Anwendung im Zusammenhang mit dem Thema.

B. **Aufbau**
Erklärung zum Aufbau des Berichtes, was dazugehört und was nicht.

II. **Literaturübersicht**

Bsp. 3-3-2

q1=I q2=IA q3=IB q4=II

```
\renewcommand\theenumi   {\Roman{enumi}}
\renewcommand\theenumii  {\Alph{enumii}}
\renewcommand\labelenumii{\theenumii.}
\begin{enumerate}
  \item \textbf{Einleitung}          \label{q1}
  \begin{enumerate}
    \item \textbf{Anwendungen}  \\
      Motivation zu Forschung und Anwendung
      im Zusammenhang mit dem Thema. \label{q2}
    \item \textbf{Aufbau}  \\
      Erklärung zum Aufbau des Berichtes, was
      dazugehört und was nicht.  \label{q3}
  \end{enumerate}
  \item \textbf{Literaturübersicht}  \label{q4}
\end{enumerate}
q1=\ref{q1} q2=\ref{q2} q3=\ref{q3} q4=\ref{q4}
```

Nach diesen Neudefinitionen sehen die Verweise etwas seltsam aus. Um dies zu korrigieren, muss die Definition des Präfixbefehls `\p@enumii` geändert werden. Um beispielsweise den Verweis „I–A“ anstelle von „IA“ (wie im vorherigen Beispiel) zu erhalten, benötigt man folgende Definition:

```
\makeatletter \renewcommand\p@enumii{\theenumi--} \makeatother
```

denn der Verweis wird formatiert, indem `\p@enumii`, gefolgt von `\theenumii` ausgeführt wird. Zu beachten ist, dass hier `\makeatletter` und `\makeatother` angegeben werden müssen, weil der Name des neu zu definierenden Befehls ein `@`-Zeichen enthält. Anstelle dieses Low-Level-Verfahrens kann auch der Befehl `\labelformat` aus dem Paket varioref verwendet werden, das in Abschnitt 2.4.2 beschrieben wird.

Das enumerate-Feld lässt sich ausschmücken, indem man das Labelfeld erweitert. Im nächsten Beispiel wird als Präfix für die Label aller Listenelemente der ersten Ebene das Paragraphenzeichen (§) und als Suffix ein Punkt verwendet (der in den Verweisen weggelassen wird).

```
\renewcommand\labelenumi{\S\theenumi.}
\usepackage{varioref} \labelformat{enumi}{\S#1}
\begin{enumerate}
\item \label{w1} Listentext, weiterer Listentext
\item \label{w2} Listentext, weiterer Listentext
\end{enumerate}
w1=\ref{w1}  w2=\ref{w2}
```

Bsp. 3-3-3

§1. Listentext, weiterer Listentext

§2. Listentext, weiterer Listentext

w1=§1 w2=§2

Möglicherweise sollen auch verschiedene Zeichen für aufeinander folgende Label verwendet werden. So nutzt das folgende Beispiel Zeichen des PostScript-Zeichensatzes Zapf Dingbats. In diesem Fall gibt es keinen einfachen Weg, die korrekten Querverweise automatisch über \ref-Befehle zu erzeugen. Anstatt \theenumi für die Darstellung des enumi-Zählers zu verwenden, kann man den Befehl so definieren, dass er aus dem Zählerwert das zu verwendende Symbol berechnet. Die Schwierigkeit besteht hierbei darin, diese Definition so zu gestalten, dass sie den Labelgenerierungsprozess überlebt. Hier hilft ein Trick: Man fügt \protect-Befehle hinzu, so dass \setcounter und \ding nicht ausgeführt werden, wenn das Label in die .aux-Datei geschrieben wird. Trotzdem muss sichergestellt sein, dass der aktuelle Wert dort abgelegt wurde. Dies erreicht man, indem man innerhalb von \setcounter dem Befehl \value den (internen) TEX-Befehl \the voranstellt (jedoch nicht innerhalb von \ding!); ohne dies würden die Verweise alle denselben Wert haben.[1]

```
\usepackage{calc,pifont}  \newcounter{local}
\renewcommand\theenumi{\protect\setcounter{local}%
  {171+\the\value{enumi}}\protect\ding{\value{local}}}
\renewcommand\labelenumi{\theenumi}
\begin{enumerate}
\item Listentext, Listentext, \label{l1}
      Listentext, Listentext, weiterer Listentext;
\item Listentext, Listentext, \label{l2}
\item Listentext, Listentext. \label{l3}
\end{enumerate}
l1=\ref{l1}  l2=\ref{l2}  l3=\ref{l3}
```

Bsp. 3-3-4

① Listentext, Listentext, Listentext, Listentext, weiterer Listentext;

② Listentext, Listentext,

③ Listentext, Listentext.

l1=① l2=② l3=③

Dieselbe Wirkung erzielt man mit der Umgebung dingautolist, die im Paket pifont definiert ist, einem Teil des PSNFSS-Systems (beschrieben in Abschnitt 7.6.4 auf Seite 390).

[1] Für TEXnisch Interessierte: LATEXs Befehl \value erzeugt, trotz seines Namens, nicht den „Wert“ eines LATEXs Zählers, sondern nur seinen internen TEX-Registernamen. In den meisten Situationen kann dieser als Wert verwendet werden, aber leider nicht innerhalb von \edef oder \write, bei denen nicht der „Wert“, sondern der interne Name überlebt. Durch Voranstellen des Befehls \the vor den internen Registernamen bekommt man auch in diesen Fällen den „Wert“.

Anpassen der Listenumgebung `description`

Zur Anpassung der `description`-Umgebung kann man den Parameter `\descriptionlabel` verändern, der die Label erzeugt. Im folgenden Beispiel wird der Zeichensatz für die Label von fett (Standardeinstellung) in serifenlos geändert.

Bsp. 3-3-5

A. Listentext, Listentext, Listentext, weiterer Listentext;

B. Listentext, Listentext, Listentext, weiterer Listentext;

```
\renewcommand\descriptionlabel[1]%
        {\hspace{\labelsep}\textsf{#1}}
\begin{description}
\item[A.] Listentext, Listentext,
     Listentext, weiterer Listentext;
\item[B.] Listentext, Listentext,
     Listentext, weiterer Listentext;
\end{description}
```

Die LaTeX-Standardklassen setzen den Anfangspunkt für Labelboxen in einer `description`-Umgebung vor dem linken Rand der umschließenden Umgebung, und zwar mit dem Abstand `\labelsep`. In dem oben aufgeführten Beispiel fügt `\descriptionlabel` daher zunächst den Wert `\labelsep` wieder hinzu, damit das Label linksbündig am Seitenrand beginnt (siehe Seite 156 für eine ausführliche Beschreibung).

3.3.2 paralist – Erweiterte Listenumgebungen

Das Paket paralist von Bernd Schandl enthält eine Reihe neuer Listenumgebungen und bietet Erweiterungen für LaTeXs Standardlistenumgebungen, was deren Anpassung stark vereinfacht. Standardumgebungen und neue Listenumgebungen können ineinander verschachtelt werden und die Umgebungen für nummerierte Listen unterstützen LaTeXs Querverweismechanismus (`\label`, `\ref`).

Nummerierte Listen

Alle LaTeX-Standardlisten sind abgesetzte Listen, d.h. sie fügen oberhalb und unterhalb der Liste sowie zwischen den einzelnen Listenpunkten einen Abstand ein. Manchmal sollen die Punkte in einem Absatz ohne diese optische Unterbrechung nummeriert werden. Zu diesem Zweck wurde die Umgebung `inparaenum` entwickelt. Sie hat ein optionales Argument, das zum Generieren der Labels verwendet werden kann; auf die genaue Syntax wird später noch eingegangen.

Bsp. 3-3-6

Vielleicht sollen die Punkte in einem Absatz nummeriert werden, um (a) Platz zu sparen (b) eine nicht so bedeutende Aussage zu treffen oder (c) aus einem anderen Grund.

```
\usepackage{paralist}
Vielleicht sollen die Punkte in einem
Absatz nummeriert werden, um
\begin{inparaenum}[(a)]
  \item Platz zu sparen
  \item eine nicht so bedeutende Aussage
        zu treffen oder
  \item aus einem anderen Grund.
\end{inparaenum}
```

Das ist möglicherweise aber nicht exakt das, was gewünscht war. Viele Leute bevorzugen abgesetzte Listen, wollen aber nicht so viel Weißraum drum herum. In diesem Fall kann vielleicht `compactenum` Abhilfe schaffen. Dieser Befehl formatiert die Liste wie `enumerate`, setzt aber den vertikalen Abstand auf `0pt`.

Manchmal sollen Aufzählungen aber auch so aussehen:

i) immer noch als abgesetzte Liste;
ii) Punkte werden wie gewöhnlich, aber mit weniger vertikalem Abstand formatiert, d.h.
iii) ähnlich einer `enumerate`-Liste.

```
\usepackage{paralist}
Manchmal sollen Aufzählungen aber auch so aussehen:
\begin{compactenum}[i)]
 \item immer noch als abgesetzte Liste;
 \item Punkte werden wie gewöhnlich, aber mit
       weniger vertikalem Abstand formatiert, d.h.
 \item ähnlich einer \texttt{enumerate}-Liste.
\end{compactenum}
```

Bsp. 3-3-7

Genaugenommen stimmt die Aussage von oben nicht – die vertikalen Abstände von `compactenum` können angepasst werden, und zwar mit folgenden Parametern: `\pltopsep` gibt den Abstand über und unter der Umgebung an, `\plpartopsep` gibt den zusätzlichen Abstand an, der hinzugefügt wird, wenn die Umgebung mit einem eigenen Absatz beginnt, `\plitemsep` gibt den Abstand zwischen den Listenpunkten an und `\plparsep` gibt den Abstand zwischen Absätzen eines Listenpunktes an.

Eine weitere Variante für nummerierte Listen bietet schließlich die Umgebung `asparaenum`; sie formatiert die Listenpunkte als separate Absätze, d.h. die erste Zeile wird um `\parindent` eingerückt und die folgenden Zeilen werden bündig zum linken Seitenrand gesetzt.

Manchmal sollen Aufzählungen aber auch so aussehen:

1) immer noch als abgesetzte Liste;
2) Punkte werden als Absätze formatiert, aber ohne eingerückte Folgezeilen, d.h.
3) ähnlich einer `enumerate`-Liste, aber doch sichtbar unterschiedlich.

```
\usepackage{paralist}
Manchmal sollen Aufzählungen aber auch so aussehen:
\begin{asparaenum}[1)]
 \item immer noch als abgesetzte Liste;
 \item Punkte werden als Absätze formatiert,
       aber ohne eingerückte Folgezeilen, d.h.
 \item ähnlich einer \texttt{enumerate}-Liste, aber
       doch sichtbar unterschiedlich.
\end{asparaenum}
```

Bsp. 3-3-8

Wie in den vorangegangenen Beispielen gezeigt, unterstützen alle ein optionales Argument, das beschreibt, wie die Labels für die Listenpunkte formatiert werden. Innerhalb des Argumentes haben die Zeichen `A`, `a`, `I`, `i` und `1` eine besondere Bedeutung: Sie werden durch den Nummerierungszähler ersetzt, der im Stil `\Alph`, `\alph`, `\Roman`, `\roman` bzw. `\arabic` angezeigt wird. Alle anderen Zeichen behalten ihre normale Bedeutung. Das Argument `[(a)]` erzeugt also beispielsweise Labels, wie (a), (b), (c) usw., während das Argument `[\S i:]` die Ausgabe §i:, §ii:, §iii: usw. produziert.

Wenn das Label Textfolgen enthält, wie Beispiel 1, Beispiel 2, … ist Vorsicht geboten. In diesem Fall muss das „`i`“ eingeklammert werden, was heißt,

dass ein Argument, wie `[{Beispiel} 1]`, verwendet wird. Ansonsten gibt es seltsame Ergebnisse, wie das nächste Beispiel zeigt.

Bsp. 3-3-9

Punkt ii zeigt, was schief gehen kann:

Beispiel i: Beim ersten Punkt ist nichts zu sehen, aber

Beiispiiel ii: der zweite zeigt, was passiert, wenn ein spezielles Zeichen fehlinterpretiert wird.

```
\usepackage{paralist}
Punkt~\ref{bad} zeigt, was schief gehen kann:
\begin{asparaenum}[Beispiel i:]
\item Beim ersten Punkt ist nichts zu sehen, aber
\item\label{bad} der zweite zeigt, was passiert,
 wenn ein spezielles Zeichen fehlinterpretiert wird.
\end{asparaenum}
```

Glücklicherweise entdeckt das Paket für gewöhnlich solch falsche Eingaben und gibt eine Warnung aus. Spezielle Zeichen zu verstecken, indem man sie in geschweifte Klammern einbettet, führt dazu, dass ein Argument wie `[\textbf{a)}]` auch nicht funktioniert, weil das „a“ nicht mehr als Sonderzeichen betrachtet wird. Eine provisorische Lösung ist, eine andere Konstruktion zu verwenden, die keine Klammern benötigt, wie `\bfseries`.

Wie oben gezeigt, wird bei der Referenzierung von `\label` nur der Zählerwert in der gewählten Darstellung erzeugt, jedoch kein schmückendes Beiwerk, das im optionalen Argument angegeben ist. Dies gilt für alle Umgebungen für nummerierte Listen.

Mit dieser Syntax ist es nicht möglich anzugeben, dass im Label der äußere und der innere Nummerierungszähler wiedergeben werden soll, weil die Symbole immer auf den aktuellen Nummerierungszähler verweisen. Es gibt nur eine Ausnahme: Wenn das Paket mit der Option `pointedenum` oder der Option `pointlessenum` geladen wird, werden Labels, wie jene im nächsten Beispiel, erzeugt.

Bsp. 3-3-10

1. Erste Ebene.
   1.1. Zweite Ebene.
      1.1.1. Dritte Ebene.
   1.2. Wieder zweite Ebene.

```
\usepackage[pointedenum]{paralist}
\begin{compactenum}
\item Erste Ebene.
 \begin{compactenum}
 \item Zweite Ebene.
  \begin{compactenum}\item Dritte Ebene.\end{compactenum}
 \item Wieder zweite Ebene.
 \end{compactenum}
\end{compactenum}
```

Der Unterschied zwischen den beiden Optionen besteht in der An- bzw. Abwesenheit des abschließenden Punktes. Alternativ zu den Optionen können die Befehle `\pointedenum` und `\pointlessenum` verwendet werden. Sie ermöglichen die Definition eigener Umgebungen, die Labels auf diese Weise formatieren, während andere Listenumgebungen Labels in anderen Formaten darstellen. Kompliziertere Labels, wie jene, die verschiedene Nummerierungszähler aus verschiedenen Ebenen beinhalten, müssen manuell erstellt werden; dazu kann man die in Abschnitt 3.3.1 auf Seite 136 beschriebenen Verfahren verwenden.

Diese Syntax im optionalen Argument, welches die Formatierung der Nummerierungslabels festlegt, wurde zunächst im Paket enumerate von

David Carlisle implementiert. Dieses Paket erweitert LaTeXs `enumerate`-Umgebung, um solch ein optionales Argument zu unterstützen. Mit paralist wird das optionale Argument von allen nummerierten Listen unterstützt, einschließlich der Standardumgebung `enumerate` (für welche es eine aufwärtskompatible Erweiterung darstellt).

Wenn eine der Umgebungen für nummerierte Listen ein optionales Argument verwendet, wird der linke Seitenrand nur so breit gewählt, dass die Labels hineinpassen. Genauer gesagt, wird der Einzug an diejenige Breite des Labels angepasst, die es beim Zählerwert sieben einnimmt. Für den Nummerierungsstil „Roman“ wird dabei eine relativ breite Nummer (vii) erzeugt, ansonsten passiert nichts. Dieses Verhalten wird im nächsten Beispiel gezeigt. Für einige Dokumente mag dieses Verhalten richtig sein, aber wenn ein einheitlicherer Einzug gewünscht ist, sollte die Option `neverdecrease` verwendet werden. Sie stellt sicher, dass der linke Seitenrand immer mindestens so breit ist wie die Standardeinstellung.

```
\usepackage{paralist}
Der linke Seitenrand kann variieren, wenn man nicht
aufpasst.
\begin{enumerate}
  \item Ein Listenpunkt in einer normalen
        \texttt{enumerate}-Umgebung.
\end{enumerate}
\begin{compactenum}
  \item Derselbe linke Rand wie
  \item in diesem Fall.
\end{compactenum}
\begin{compactenum}[i)]
  \item Aber ein anderer Einzug \item hier.
\end{compactenum}
```

Bsp. 3-3-11

Der linke Seitenrand kann variieren, wenn man nicht aufpasst.

1. Ein Listenpunkt in einer normalen `enumerate`-Umgebung.

1. Derselbe linke Rand wie
2. in diesem Fall.

i) Aber ein anderer Einzug
ii) hier.

Auf der anderen Seite kann diese Art von Einstellung erzwungen werden, selbst bei Umgebungen ohne optionales Argument; dafür ist die Option `alwaysadjust` zu verwenden.

```
\usepackage[alwaysadjust]{paralist}
Hier wird stets der kleinstmögliche Einzug erzwungen:
\begin{enumerate}
  \item Ein Listenpunkt in einer normalen
        \texttt{enumerate}-Umgebung.
\end{enumerate}
\begin{compactenum}[i)]
  \item Aber ein anderer
  \item Einzug
  \item hier.
\end{compactenum}
\begin{compactenum}[1.]
  \item Derselbe linke Rand wie
  \item in der ersten Liste.
\end{compactenum}
```

Bsp. 3-3-12

Hier wird stets der kleinstmögliche Einzug erzwungen:

1. Ein Listenpunkt in einer normalen `enumerate`-Umgebung.

i) Aber ein anderer
ii) Einzug
iii) hier.

1. Derselbe linke Rand wie
2. in der ersten Liste.

Schließlich wird mit der Option `neveradjust` in allen Fällen der Standardeinzug verwendet. Dabei ragen alle Labels, die zu breit sind, in den linken Seitenrand hinein.

Bsp. 3-3-13

```
\usepackage[neveradjust]{paralist}
Mit dieser Option wird das Label in den Seitenrand
verschoben.
\begin{enumerate}
  \item Ein Listenpunkt in einer normalen
        \texttt{enumerate}-Umgebung.
\end{enumerate}
\begin{compactenum}[{Aufgabe} A)]
  \item Derselbe linke Rand wie
  \item in diesem Fall.
\end{compactenum}
\begin{compactenum}[1)]
  \item Und derselbe Einzug \item hier.
\end{compactenum}
```

Mit dieser Option wird das Label in den Seitenrand verschoben.

1. Ein Listenpunkt in einer normalen `enumerate`-Umgebung.

Aufgabe A) Derselbe linke Rand wie
Aufgabe B) in diesem Fall.

1) Und derselbe Einzug
2) hier.

Aufzählungen

Für Aufzählungen bietet das Paket paralist die Umgebungen `compactitem` (eine kompakte Version der Standardumgebung `itemize`), `asparaitem` (die Listenpunkte als Absätze formatiert) und `inparaitem` (die im Text eingebettete Aufzählungen erzeugt). Die letzte Umgebung wurde hauptsächlich aus Symmetriegründen hinzugefügt. Alle drei Umgebungen haben ein optionales Argument, welches das Label angibt, das für die Listenpunkte verwendet werden soll.

Bsp. 3-3-14

```
\usepackage[neverdecrease]{paralist}
Listen mit besonderen Labels zu erzeugen, ist
recht einfach.
\begin{compactitem}[$\star$]
  \item Dieses Beispiel verwendet die Paketoption
        \texttt{neverdecrease}.
  \item Ohne sie wäre der linke Rand schmaler.
\end{compactitem}
```

Listen mit besonderen Labels zu erzeugen, ist recht einfach.

- ⋆ Dieses Beispiel verwendet die Paketoption `neverdecrease`.
- ⋆ Ohne sie wäre der linke Rand schmaler.

Die drei Optionen `neverdecrease`, `alwaysadjust` und `neveradjust`, mit denen man die Labels ausrichten kann, sind auch für Aufzählungen gültig, wie im vorangegangenen Beispiel zu sehen war. Wenn das Paket paralist geladen ist, wird LaTeXs Umgebung `itemize` erweitert, so dass sie auch diese Art des optionalen Argumentes unterstützt.

Beschreibungen

Für Beschreibungen führt das Paket paralist drei zusätzliche Umgebungen ein: `compactdesc` (die mit der Standardumgebung `description` von LaTeX identisch ist, außer dass alle vertikalen Abstände auf null – oder auf jeden anderen, benutzerdefinierten Wert – reduziert werden), `asparadesc` (die jeden Listenpunkt als Absatz formatiert) und `inparadesc` (die beschreibende Listen innerhalb des laufenden Textes zulässt).

Da description-Umgebungen jedes Label mit dem Befehl \item definieren, benötigen sie kein optionales Argument.

```
\usepackage{paralist}
Wie wär's mit einer beschreibenden Liste im Text?
\begin{compactdesc}
\item[paralist] Ein nützliches Paket, denn es
 unterstützt die Umgebungen
 \begin{inparadesc} \item[compact\ldots]
   die keinen vertikalen Abstand haben,
 \item[aspara\ldots,] die als Absätze
   formatiert werden und
 \item[inpara\ldots] als im Text
    eingebettete Listen.
 \end{inparadesc}
\item[enumerate] Ein jetzt überholtes Paket.
\end{compactdesc}
```

Bsp. 3-3-15

Wie wär's mit einer beschreibenden Liste im Text?

paralist Ein nützliches Paket, denn es unterstützt die Umgebungen **compact…** die keinen vertikalen Abstand haben, **aspara…,** die als Absätze formatiert werden und **inpara…** als im Text eingebettete Listen.

enumerate Ein jetzt überholtes Paket.

Anpassen der Voreinstellungen

Neben der Bereitstellung dieser nützlichen neuen Umgebungen ermöglicht das Paket paralist, die Voreinstellungen von nummerierten Listen und Aufzählungen zu ändern.

Mithilfe der Deklaration \setdefaultitem können die Standardlabels für unterschiedliche Ebenen der Aufzählungen angegeben werden. Sie hat vier Argumente (weil vier Verschachtelungsebenen möglich sind). In jedem Argument wird das gewünschte Label angegeben (so wie im optionalen Argument der Umgebung selbst). Wenn die Voreinstellung für eine Ebene nicht geändert werden soll, wird ein leeres Argument angegeben.

```
\usepackage{paralist}
\setdefaultitem{}{\textbullet}{$\star$}{}
\begin{compactitem}
\item Die äußere Ebene verwendet das voreingestellte Label.
  \begin{compactitem}
  \item Die zweite Ebene verwendet auch einen Punkt.
    \begin{compactitem}
    \item Und auf der dritten Ebene ein Stern.
    \end{compactitem}
  \end{compactitem}
\end{compactitem}
```

Bsp. 3-3-16

- Die äußere Ebene verwendet das voreingestellte Label.
  - Die zweite Ebene verwendet auch einen Punkt.
    - ⋆ Und auf der dritten Ebene ein Stern.

Die geänderten Voreinstellungen gelten für alle nachfolgenden Aufzählungsumgebungen. In der Regel wird solch eine Deklaration in die Präambel eingefügt, sie kann aber auch verwendet werden, um Voreinstellungen in der Mitte des Dokumentes zu ändern. Insbesondere können Umgebungen definiert werden, die eine \setdefaultitem-Deklaration enthalten, die dann nur für diese bestimmte Umgebung gilt – aber dann auch für Listen, die darin verschachtelt sind.

Es ist wahrscheinlich wenig überraschend, dass es eine ähnliche Deklaration für nummerierte Listen gibt. Mit `\setdefaultenum` kann das voreingestellte Layout solcher Umgebungen gesteuert werden. Auch hier gibt es vier Argumente, die den vier möglichen Ebenen entsprechen. Für jede Ebene kann entweder eine Labeldefinition angegeben werden (über die zuvor bereits beschriebene Syntax) oder das Argument leer gelassen werden, wenn der voreingestellte Wert verwendet werden soll.

Bsp. 3-3-17

1) In diesem Beispiel haben alle Ebenen eine schließende Klammer.
   a) Hier Kleinbuchstaben.
      i) Hier römische Ziffern.
      ii) Wirklich!

```
\usepackage{paralist}  \setdefaultenum{1)}{a)}{i)}{A)}
\begin{compactenum}
\item In diesem Beispiel haben alle Ebenen eine
      schließende Klammer.
  \begin{compactenum}
  \item Hier Kleinbuchstaben.
    \begin{compactenum}
    \item Hier römische Ziffern. \item Wirklich!
    \end{compactenum}
  \end{compactenum}
\end{compactenum}
```

Es gibt auch die Möglichkeit, den Einzug der einzelnen Listenebenen zu verändern, und zwar mit der Deklaration `\setdefaultleftmargin`. Dieser Befehl hat jedoch sechs Argumente (denn in den Standard-LaTeX-Klassen gibt es insgesamt 6 Listenebenen), in denen entweder ein Maß angegeben wird, das die Vergrößerung des Einzugs für die jeweilige Ebene festlegt, oder das leer bleibt, wenn der aktuelle Wert übernommen werden soll, der von der Klasse oder anderenorts definiert wurde. Ein weiterer Unterschied zu den vorangegangenen Deklarationen besteht darin, dass es sich in diesem Fall um absolute Listenebenen handelt, und nicht um relative Ebenen, die sich entweder auf Nummerierungen oder Aufzählungen (die untereinander vermischt werden können) beziehen. Ein Vergleich des nächsten Beispiels mit dem vorhergehenden zeigt die Unterschiede.

Bsp. 3-3-18

1) In diesem Beispiel haben alle Ebenen eine schließende Klammer.
   a) Hier Kleinbuchstaben.
      i) Hier römische Ziffern.
      ii) Wirklich!

```
\usepackage{paralist}
\setdefaultenum{1)}{a)}{i)}{A)}
\setdefaultleftmargin{\parindent}{\parindent}
                     {\parindent}{}{}{}
\begin{compactenum}
\item In diesem Beispiel haben alle Ebenen eine
      schließende Klammer.
  \begin{compactenum}
  \item Hier Kleinbuchstaben.
    \begin{compactenum}
    \item Hier römische Ziffern. \item Wirklich!
    \end{compactenum}
  \end{compactenum}
\end{compactenum}
```

Standardmäßig werden die Labels in nummerierten Listen und Aufzählungen linksbündig gesetzt. Diese Einstellung lässt sich mit der Option `flushleft` ändern.

Wie bereits beschrieben, kann das Label einer `description`-Liste durch Modifikation von `\descriptionlabel` angepasst werden; dieser Befehl legt auch die Formatierung des Labels in einer `compactdesc`-Umgebung fest. Bei `inparadesc` und `asparadesc` wird jedoch ein anderer Befehl namens `\paradescriptionlabel` für diesen Zweck verwendet. Da diese Umgebungen ihre Labels leicht unterschiedlich positionieren, benötigt man bei ihnen keine Positionskorrektur um den Betrag von `\labelsep` (siehe zum Vergleich Seite 156). Ihre Standarddefinition lautet daher einfach:

```
\newcommand*\paradescriptionlabel[1]{\normalfont\bfseries #1}
```

Schließlich unterstützt das Paket paralist die Verwendung einer Konfigurationsdatei namens `paralist.cfg`, die standardmäßig geladen wird, sofern sie existiert. Dies kann durch Angabe der Option `nocfg` verhindert werden.

3.3.3 amsthm – Theoremähnliche Strukturen

Der Begriff „headed lists“ (Listen mit Überschrift) beschreibt typographische Strukturen, die wie andere Listen (z.B. Zitate) einen separaten oder abgesetzten Teil eines Abschnitts oder eines Kapitels bilden und bei denen mindestens Anfang und Ende eindeutig zu identifizieren sein müssen. Dies geschieht in der Regel durch Anpassung des vertikalen Abstandes am Anfang oder durch Anfügen einer Linie – und in diesem Fall auch durch Hinzufügen einer Überschrift, ähnlich einer Gliederungsüberschrift. Auch das Ende kann durch eine Linie oder ein anderes Symbol (z.B. im letzten Absatz) und zusätzlichem vertikalen Abstand abgesetzt werden.

Eine weitere Kennzeichnung dieser Listen ist, dass sie häufig nummeriert sind, entweder mit einer eigenen Nummerierung oder in Verbindung mit der Überschriftennummerierung.

Eine der wohl wichtigsten Vertreter von „Listen mit Überschrift“ sind die sogenannten „theoremähnlichen“ Umgebungen. Sie wurden ursprünglich hauptsächlich in mathematischen Schriften und Büchern verwendet, sind aber genauso auf eine Vielzahl anderer hervorgehobener Textpassagen anwendbar. Tatsächlich können alle Arten von Beispielen und Übungen auf diese Art präsentiert werden, auch wenn sie keine mathematischen Formeln enthalten.

Da ihre historischen Ursprünge in der mathematischen Welt liegen, wird das amsthm-Paket [7] von Michael Downes von der amerikanischen Gesellschaft für Mathematik (AMS) stellvertretend für diese Art von Erweiterung beschrieben.[1] Dieses Paket stellt eine verbesserte Version der Standard-LaTeX-Deklaration `\newtheorem` bereit, die theoremähnliche Umgebungen definiert.

Wie bei Standard-LaTeX, haben Umgebungen, die auf diese Art deklariert werden, ein optionales Argument, in dem zusätzlicher Text, sogenannte „Anmerkungen“, zur Überschrift der Umgebung hinzugefügt werden können. Eine Demonstration gibt das Beispiel unten.

[1] Wenn die Pakete amsthm und amsmath zusammen mit einer anderen als einer der AMS-Dokumentenklassen verwendet werden, muss das Paket amsthm *nach* amsmath geladen werden. Die AMS-Dokumentenklassen umfassen beide Pakete.

`\newtheorem*{`*name*`}{`*überschrift*`}`

Die Deklaration `\newtheorem` hat zwei obligatorische Argumente. Im ersten wird der Name der neuen Umgebung angegeben, den der Autor für dieses Element verwenden möchte. Das zweite enthält den Überschriftentext.

Wird anstelle von `\newtheorem` die Sternform `\newtheorem*` verwendet, dann werden für diese Umgebungen keine automatischen Nummern generiert. Diese Form des Befehls kann nützlich sein, wenn nur ein Lemma oder eine Übung vorhanden ist und diese(s) nicht nummeriert werden soll; außerdem eignet sie sich zur Erzeugung einzelner Theoreme, die unter ihrem Namen bekannt sind.

Bsp. 3-3-19

Lemma 1 (Zentrale Aussage)**.** *Der LaTeX-Begleiter vervollständigt jede LaTeX-Einführung.*

Mittelbachs Lemma. *Der LaTeX-Begleiter enthält Pakete aus allen Anwendungsbereichen.*

```
\usepackage{amsthm}
\newtheorem{lem}{Lemma}
\newtheorem*{ML}{Mittelbachs Lemma}
\begin{lem}[Zentrale Aussage] Der \LaTeX{}-Begleiter
  vervollständigt jede \LaTeX{}-Einführung.
\end{lem}
\begin{ML} Der \LaTeX{}-Be\-gleiter enthält
  Pakete aus allen Anwendungsbereichen.
\end{ML}
```

Neben den zwei obligatorischen Argumenten hat `\newtheorem` zwei sich gegenseitig ausschließende optionale Argumente. Sie betreffen die chronologische und hierarchische Abfolge der Nummerierung.

`\newtheorem{`*name*`}[`*verwende-zähler*`]{`*überschrift*`}`
`\newtheorem{`*name*`}{`*überschrift*`}[`*nummer-innerhalb*`]`

Standardmäßig werden die einzelnen theoremähnlichen Umgebungen unabhängig voneinander nummeriert. Wenn z.B. in einem Text Lemmata, Theoreme und Beispiele vorkommen, werden sie in etwa folgendermaßen nummeriert: Beispiel 1, Lemma 1, Lemma 2, Theorem 1, Beispiel 2, Lemma 3, Theorem 2. Wenn Lemmata und Theoreme eine gemeinsame Nummerierung haben sollen, d.h. Beispiel 1, Lemma 1, Lemma 2, Theorem 3, Beispiel 2, Lemma 4, Theorem 5, dann sollte die gewünschte Beziehung in folgender Weise angegeben werden:

```
\newtheorem{thm}{Theorem}     \newtheorem{lem}[thm]{Lemma}
```

Das optionale Argument *verwende-zähler* (Wert `thm`) im zweiten Ausdruck bedeutet, dass die `lem`-Umgebung keine eigene Nummerierung einführen soll, sondern stattdessen auch die `thm`-Nummerierung verwenden soll.

Damit eine Theoremumgebung die Abschnittsnummerierung übernimmt, z.B. Übung 2.1, Übung 2.2 usw. in Abschnitt 2, muss an der letzten Stelle der Name des übergeordneten Zählers in eckigen Klammern angegeben werden:

```
\newtheorem{exa}{Übung}[section]
```

Mit dem optionalen Argument `[section]` wird der Zähler `exa` auf 0 zurückgesetzt, sobald der übergeordnete Zähler `section` hochgezählt wird.

Layoutdefinitionen für theoremähnliche Strukturen

Der Spezifikationsteil des amsthm-Paketes unterstützt die Idee eines aktuellen Theoremlayouts, das die Formatierung für eine Sammlung von \newtheorem-Deklarationen festlegt.[1]

\theoremstyle{*layout*}

Die drei Theoremlayouts, die durch das Paket bereitgestellt werden, heißen plain, definition und remark. Sie legen verschiedene typographische Stile fest, die den Umgebungen je nach ihrer Bedeutung optisch Nachdruck verleihen. Die Details dieser typographischen Gestaltung können, abhängig von der Dokumentenklasse, variieren. In der Regel setzt plain den Haupttext in einem kursiven Schriftschnitt, während die anderen beiden Layouts eine aufrechte Antiqua verwenden.

Neue theoremähnliche Umgebungen in diesen Layouts erstellt man, indem man die \newtheorem-Deklarationen in Gruppen aufteilt. Jeder Gruppe wird der passende \theoremstyle vorangestellt. Wenn man keinen \theoremstyle-Befehl benutzt, wird das Layout plain verwendet. Hier einige Beispiele:

```
\usepackage{amsthm}
\theoremstyle{plain}      \newtheorem{thm}{Theorem}
\theoremstyle{definition}\newtheorem{defn}[thm]{Definition}
\theoremstyle{remark}     \newtheorem*{rem}{Anmerkung}
\begin{defn} Eine typographische Herausforderung ist ein
             Problem, das sich nicht mithilfe des
             \emph{\LaTeX{}-Begleiters} lösen lässt.
\end{defn}
\begin{thm}Es gibt keine typographischen Herausforderungen.
\end{thm}
\begin{rem}Der Beweis bleibt dem Anwender überlassen.
\end{rem}
```

Bsp. 3-3-20

Definition 1. Eine typographische Herausforderung ist ein Problem, das sich nicht mithilfe des *LaTeX-Begleiters* lösen lässt.

Theorem 2. *Es gibt keine typographischen Herausforderungen.*

Anmerkung. Der Beweis bleibt dem Anwender überlassen.

Zu beachten ist, dass die nahe liegende Bezeichnung „def“ für eine „Definition“s-Umgebung nicht funktioniert, weil sie mit dem Low-Level-Befehl von TeX \def kollidiert.

Nummern tauschen

Ein oft verwendetes Layout für Theoremüberschriften ist, die Nummer nicht rechts, sondern links von der Überschrift zu setzen. Da diese Variante normalerweise global angewendet wird, ungeachtet dessen, wie sich einzelne \theoremstyles ändern, kann man den Austausch von Nummern durchführen, indem man zu Beginn der Liste von \newtheorem-Befehlen, auf die sich dieser Wechsel auswirken soll, die Deklaration \swapnumbers hinzufügt.

Erweiterte Anpassung

Umfangreichere Möglichkeiten zur Anpassung bietet das Paket durch die Deklaration \newtheoremstyle und eine Reihe von Paketoptionen, über welche die Definitionen für Theoremlayouts geladen werden.

[1] Dieses Konzept wurde ursprünglich im inzwischen überholten theorem-Paket von Frank Mittelbach eingeführt.

```
\newtheoremstyle{name}{abstand-oben}{abstand-unten}
                 {text-layout}{einzug}{kopf-layout}
                 {kopf-nach-punkt}{kopf-nach-abstand}{spez-kopf}
```

Ein neues Layout für eine theoremähnliche Struktur wird definiert, indem man diese Deklaration mit den neun obligatorischen Argumenten angibt, die unten beschrieben werden. Bei vielen dieser Argumente erhält man den unten aufgeführten voreingestellten Wert, wenn ein leeres Argument angegeben wird.

name Der Name, der für den Verweis auf das neue Layout verwendet wird.

abstand-oben Der vertikale Abstand über der theoremähnlichen Struktur, eine elastische Länge (voreingestellt `\topsep`).

abstand-unten Der vertikale Abstand unter der theoremähnlichen Struktur, eine elastische Länge (voreingestellt `\topsep`).

text-layout Eine Deklaration des Fonts und anderer Aspekte des Layouts, die für den Haupttext der Liste verwendet werden soll (voreingestellt `\normalfont`).

einzug Der zusätzliche Einzug der ersten Zeile der Liste, eine unelastische Länge (voreingestellt ist kein Einzug).

kopf-layout Eine Deklaration des Fonts und anderer Aspekte des Layouts, die für den Text der Listenüberschrift verwendet werden soll (voreingestellt ist `\normalfont`).

kopf-nach-punkt Der Text (normalerweise Satzzeichen), der nach dem Überschriftentext eingefügt werden soll, einschließlich Anmerkungen.

kopf-nach-abstand Der horizontale Abstand, der nach dem Überschriftentext und der Interpunktion eingefügt werden soll; eine elastische Länge. Dieses Argument darf nicht leer sein. Es gibt zwei Sonderfälle: Das Argument kann ein einzelnes Leerzeichen enthalten, das angibt, dass nur ein normaler Wortzwischenraum eingefügt werden soll, oder – was mehr überrascht – den Befehl `\newline`, der angibt, dass vor dem Beginn des Textkörpers der Liste ein Zeilenumbruch eingefügt werden soll.

spez-kopf Dieses Argument enthält die komplette Spezifikation für die Formatierung der Überschrift. Wenn das Argument leer ist, werden die Einstellungen des Theoremlayouts „plain“ verwendet. Einzelheiten hierzu siehe unten.

Layoutspezifikationen, die für die gesamte Umgebung gelten, sollten am besten in das Argument *text-layout* eingefügt werden. Hierbei ist jedoch darauf zu achten, wie es mit den automatischen Einstellungen interagiert. Einstellungen, die nur für die Überschrift gelten, können im *kopf-layout* angegeben werden.

Im Beispiel unten wird das Theoremlayout `break` definiert, das nach der Überschrift eine neue Zeile beginnt. Der Überschriftentext wird in serifenloser Schrift fett formatiert, gefolgt von einem Doppelpunkt und in den Rand um

12 pt ausgerückt. Da die Beispiele im Buch in einem sehr schmalen Satzspiegel gesetzt sind, wurde dem Argument *text-layout* zusätzlich `\raggedright` hinzugefügt.[1]

```
\usepackage{amsthm}
\newtheoremstyle{break}%
  {9pt}{9pt}%              Abstand oben und unten
  {\itshape\raggedright}%  Textlayout
  {-12pt}%                 Überschrift: Einzug
  {\sffamily\bfseries}{:}% Font und Satzzeichen danach
  {\newline}%              Abstand nach Überschrift
  {}%                      Überschriftenspezifikation
    %                      (leer = so wie 'plain' Layout)
\theoremstyle{break}
\newtheorem{exa}{Übung}
```

Übung 1 (Aktiver Autor):
Finde den Autor, von dem die meisten der im LaTeX-Begleiter beschriebenen Pakete stammen.

```
\begin{exa}[Aktiver Autor]
   Finde den Autor, von dem die meisten der im
   \LaTeX{}-Begleiter beschriebenen Pakete stammen.
\end{exa}
```

Bsp. 3-3-21

Überschriftenformat spezifizieren

Das Argument *spez-kopf* wird, wenn es nicht leer ist, zum Ersetzungstext eines internen Befehls. Dieser wird zur Formatierung der (maximal) drei Teile verwendet, die in der Überschrift einer theoremähnlichen Umgebung enthalten sind: die Nummer (sofern vorhanden) der Überschrift, ihr Name und gegebenenfalls Anmerkungen des Autors, die dieser beim Gebrauch der Umgebung (im optionalen Argument) angegeben hat. Daher sollte dieser Ersetzungstext Verweise auf drei Argumente enthalten, die dann folgendermaßen ersetzt werden:

`#1` Der unveränderliche Text, der in der Überschrift verwendet wird (z.B. „Übungen“). Er wird aus `\newtheorem`-Deklaration ausgelesen, die zur Definition der Umgebung verwendet wurde.

`#2` Darstellung der Nummer eines Elementes, sofern dieses nummeriert werden soll. Wird normalerweise leer gelassen, wenn die Umgebung nicht nummeriert werden soll.

`#3` Der optionale Anmerkungstext aus dem optionalen Argument der Umgebung.

Unter der Annahme, dass alle Teile vorhanden sind, könnte der Inhalt des Argumentes *spez-kopf* wie folgt aussehen:

```
#1 #2 \textup{(#3)}
```

Solch eine Deklaration ist natürlich möglich, aber es empfiehlt sich, die drei Argumentreferenzen eher nicht direkt zu verwenden, da dies zu unerwünschtem zusätzlichen Leerraum führen könnte, wenn z.B. die Umgebung nicht nummeriert ist.

Dieser erhöhten Komplexität trägt das Paket mit drei zusätzlichen Befehlen Rechnung, die jeweils ein Argument haben: `\thmname`, `\thmnumber` und `\thmnote`. Diese drei Befehle werden bei jeder Verwendung der Umgebung neu definiert, damit ihre Argumente in der richtigen Weise verarbeitet werden.

[1] Das Beispiel funktioniert nicht, wenn `ragged2e` geladen ist (Stand 2005).

Ihre Standardeinstellung ist einfach „das Argument setzen“. Wenn jedoch z.B. die spezielle Instanz unnummeriert ist, wird `\thmnumber` so definiert, dass sein Argument nicht gesetzt wird. Eine bessere Definition des Argumentes *spez-kopf* wäre also

```
\thmname{#1}\thmnumber{ #2}\thmnote{ \textup{(#3)}}
```

Dieses entspricht der Einstellung, die das Standardlayout `plain` verwendet. Eine besondere Beachtung verdienen die Leerzeichen in den beiden letzten Argumenten: Sie geben den Wortzwischenraum an, der für die einzelnen Teile der Überschrift benötigt wird. Da sie jedoch innerhalb der Argumente platziert wurden, haben sie nur eine Auswirkung, wenn der entsprechende Teil der Überschrift gesetzt wird.

Im folgenden Beispiel wird eine „Theorem“-Variante vorgestellt, bei welcher die komplette Theoremüberschrift als optionale Anmerkung angegeben werden muss, wie z.B. beim Zitieren von Theoremen aus anderen Quellen.

Bsp. 3-3-22

Theorem 3.16 in [92]. *Durch Fokussierung auf kleine Details kann man die tiefere Bedeutung einer Passage verstehen.*

```
\usepackage{amsthm}
\newtheoremstyle{citing}%  Name
  {3pt}{3pt}%              Abstand oben und unten
  {\itshape}%              Textlayout
  {\parindent}{\bfseries}% Überschrifteinzug und Font
  {.}%                     Interpunktion nach Überschrift
  { }%                     Abstand nach Überschrift
  {\thmnote{#3}}%          Anmerkung nur, wenn vorhanden
\theoremstyle{citing}   \newtheorem*{varthm}{}
\begin{varthm}[Theorem 3.16 in \cite{Knuth90}]
Durch Fokussierung auf kleine Details kann man die
tiefere Bedeutung einer Passage verstehen.
\end{varthm}
```

Beweisführungen und das QED-Zeichen

Von speziell mathematischer Natur ist die von `amsthm` definierte `proof`-Umgebung. Diese Umgebung erzeugt die Überschrift „Proof“ (anpassbar über den Parameter `\proofname`) mit passendem Leerraum und Satzzeichen und fügt am Ende automatisch ein „QED-Zeichen“ ein.[1]

Mit einem optionalen Argument der `proof`-Umgebung hat man die Möglichkeit, die Standardbezeichnung „Proof“ durch eine andere zu ersetzen. Wenn die Überschrift z.B. „Beweis des Haupttheorems“ lauten soll, ist sie folgendermaßen zu definieren:

```
\begin{proof}[Beweis des Haupttheorems]
  ...
\end{proof}
```

Das „QED-Zeichen“ (voreingestellt □) wird automatisch an das Ende der `proof`-Umgebung angehängt. Um das Beweisendezeichen durch ein anderes zu ersetzen, kann man mithilfe von `\renewcommand` den Befehl

[1] Die `proof`-Umgebung ist hauptsächlich für kurze Beweise gedacht, die nicht länger als ein oder zwei Seiten sind. Längere Beweise lassen sich im Dokument in der Regel besser als separate `\section` oder `\subsection` realisieren.

\qedsymbol neu definieren. Für einen längeren Beweis, der als \section oder \subsection realisiert wurde, kann man mit dem Befehl \qed das Zeichen und den üblichen Vorabstand erzeugen. Dazu muss der Befehl an der Stelle eingefügt werden, an der das Zeichen erscheinen soll.

Die automatische Platzierung des QED-Zeichens kann problematisch sein, wenn der letzte Teil einer proof-Umgebung z.B. tabellarisch oder eine abgesetzte Gleichung oder Liste ist. In diesem Fall sollte der Befehl \qedhere an einer früheren Stelle eingefügt werden, nämlich dort, wo das QED-Zeichen erscheinen soll. Damit wird sein Erscheinen am logischen Ende der proof-Umgebung unterdrückt. Wenn \qedhere in einer Gleichung eine Fehlermeldung erzeugt, sollte stattdessen \mbox{\qedhere} verwendet werden.

```
\usepackage{amsthm}
\begin{proof}[Beweis (Hinlänglichkeit)]
Dieser Beweis beinhaltet eine Liste:
\begin{enumerate}
 \item Weil der Beweis zweigeteilt ist,
 \item muss \verb|\qedhere| verwendet werden. \qedhere
\end{enumerate}
\end{proof}
```

Bsp. 3-3-23

Beweis (Hinlänglichkeit). Dieser Beweis beinhaltet eine Liste:

1. Weil der Beweis zweigeteilt ist,
2. muss \qedhere verwendet werden. □

3.3.4 Erstellen eigener Listen

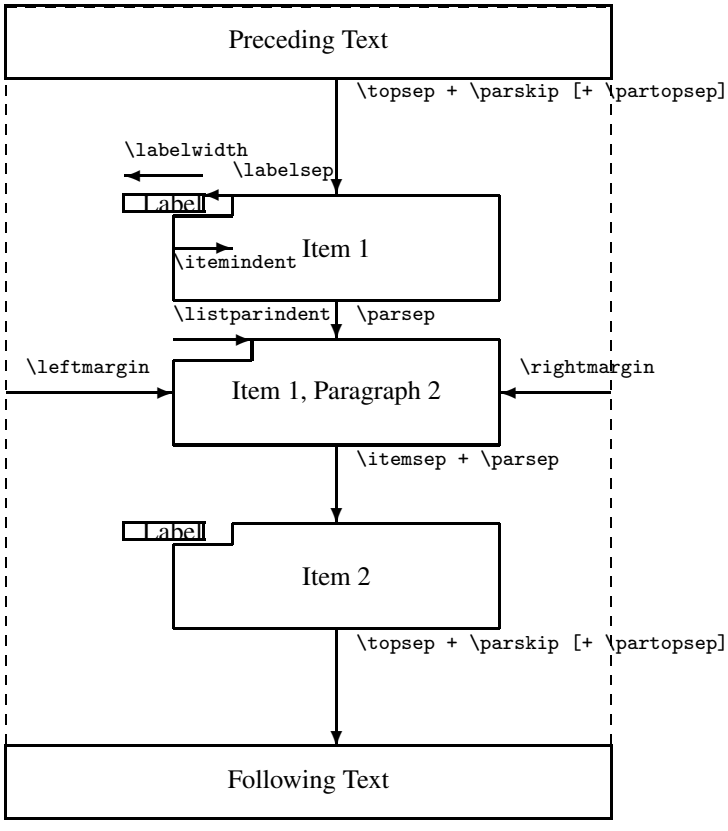
In LaTeX werden die meisten Listen, einschließlich derjenigen, die bereits vorgestellt wurden, intern mit der generischen list-Umgebung erzeugt. Diese hat folgende Syntax:

\begin{list}{*standardlabel*}{*dekl*} *element-liste* \end{list}

Das Argument *standardlabel* gibt den Text an, der als Label verwendet wird, wenn ein \item-Befehl ohne optionales Argument verwendet wird. Das zweite Argument *dekl* stellt die verschiedenen geometrischen Parameter für die list-Umgebung ein, die schematisch in Abbildung 3.3 auf der nächsten Seite dargestellt sind.

Die Standardwerte dieser Parameter hängen in der Regel vom Schriftgrad und der Listenebene ab. Die Parameter mit vertikaler Ausrichtung sind elastische Längen, d.h. sie können gestaucht und gedehnt werden. Sie werden von der list-Umgebung wie folgt gesetzt: Wenn die neue Umgebung beginnt, wird der interne Befehl \@list⟨*level*⟩ ausgeführt. Dabei gibt ⟨*level*⟩ die Verschachtelungstiefe der Liste an, die in römischen Ziffern dargestellt wird (z.B. \@listi für die erste Ebene, \@listii für die zweite Ebene, \@listiii für die dritte Ebene usw.). Jeder dieser Befehle, die von der Dokumentenklasse definiert werden, hält für die jeweilige Ebene die entsprechenden Einstellungen bereit. Typischerweise bietet die Dokumentenklasse separate Definitionen für alle durch Optionen wählbare Hauptschriftgrade des Dokumentes. Bei Wahl der Option 11pt werden beispielsweise unter anderem die Standardwerte für Listen geändert. In den Standardklassen geschieht dies durch Laden der Datei size11.clo, welche die Definitionen für die Schriftgröße 11 pt enthält.

Außerdem ändern die meisten Klassen \@listi (d.h. Listenwerte für die erste Ebene) innerhalb der größenändernden Befehle \normalsize, \small



Bsp. 3-3-24

`\topsep` Elastischer Leerraum zwischen erstem Listenpunkt und vorangegangenem Absatz.

`\partopsep` Zusätzlicher Leerraum, der zu dem Parameter `\topsep` hinzugefügt wird, wenn die Umgebung einen neuen Absatz beginnt.

`\itemsep` Elastischer Leerraum zwischen aufeinander folgenden Listenpunkten.

`\parsep` Elastischer Leerraum zwischen Absätzen innerhalb eines Listenpunktes.

`\leftmargin` Abstand zwischen dem linken Rand der umschließenden Umgebung (oder der Seite bei der obersten Verschachtelungsebene) und dem linken Rand dieser Liste. Muss größer oder gleich null sein. Der Wert hängt von der jeweiligen Listenebene ab.

`\rightmargin` Ähnlich wie `\leftmargin`, aber für den rechten Rand. Voreinstellung ist 0 pt.

`\listparindent` Zusätzlicher Einzug am Anfang jedes Absatzes einer Liste, außer bei denjenigen, die mit `\item` beginnen. Kann negativ sein, beträgt aber normalerweise 0 pt.

`\itemindent` Zusätzlicher Einzug, der zum horizontalen Einzug des Textteils der ersten Zeile eines Listenpunktes hinzuaddiert wird. Indem man die Werte von `\labelsep` und `\labelwidth` davon abzieht, wird mithilfe dieses Bezugspunktes berechnet, wo das Label beginnt. Der Wert dieser Länge beträgt normalerweise 0 pt.

`\labelwidth` Die nominale Weite der Labelbox. Wenn die natürliche Breite des Labels $\leq$`\labelwidth` ist, wird das Label per Voreinstellung rechtsbündig in einer Box der Breite `\labelwidth` gesetzt. Ansonsten wird eine Box mit der natürlichen Breite verwendet, welche wiederum einen ebenso großen Einzug für den Text dieser Zeile erzeugt. Die Art, wie das Label formatiert wird, kann mit dem Befehl `\makelabel` geändert werden.

`\labelsep` Abstand zwischen dem Ende der Labelbox und dem Text des ersten Punktes. Der Standardwert beträgt 0.5 em.

Abbildung 3.3: Parameter der `list`-Umgebung (mit layouts erstellt)

und \footnotesize ab. Dies geschieht in der Annahme, dass es Listen innerhalb von Textpassagen gibt, die in solch kleinen Schriftgrößen gesetzt sind. Da aber diese Anpassungen recht unvollständig sind, sind dabei seltsame Effekte möglich, wenn man

- verschachtelte Listen in diesen kleinen Größen verwendet (die inneren Listen erhalten die Standardwerte, die für \normalsize bestimmt sind),
- von \small oder \footnotesize direkt auf eine größere Schrift, wie \huge wechselt (eine Liste erster Ebene erbt jetzt die Voreinstellungen von der kleinen Größe, da \huge diese nicht anpasst).

Mit einer komplexeren Konfiguration könnten diese Mängel beseitigt werden. Da die einfachere Variante jedoch in den meisten Situationen sehr gut funktioniert, bieten die meisten Klassen nur diese eingeschränkte Unterstützung.

Globale Änderungen sind schwierig

Da die Einstellungen für die Listenparameter vom Schriftgrad und der Verschachtelungstiefe abhängen, können die Parameter nicht global in der Dokumentenpräambel geändert werden. Um Parameter global zu ändern, muss man die verschiedenen, oben angesprochenen \@list..-Befehle neu definieren oder eine andere Dokumentenklasse auswählen.

Seitenumbruch vor, in und nach Listen

Seitenumbrüche außerhalb und innerhalb einer Listenstruktur werden über drei TEX-Zähler gesteuert: \@beginparpenalty (für einen Umbruch vor der Liste), \@itempenalty (für einen Umbruch vor einem Listenpunkt innerhalb der Liste) und \@endparpenalty (für einen Umbruch nach einer Liste). Standardmäßig werden alle drei auf einen leicht negativen Wert gesetzt; dies bedeutet, dass es zulässig (und sogar empfehlenswert) ist, eine Seite an diesen Stellen zu umbrechen (anstatt an anderen Stellen). Es kann jedoch sein, dass das Ergebnis trotzdem nicht zufriedenstellend ausfällt. Möglicherweise soll von einem Seitenumbruch vor einer Liste eher abgeraten bzw. dieser unterdrückt werden. Dies erreicht man, indem man \@beginparpenalty einen hohen Wert zuweist (10000 oder mehr verhindert einen Umbruch unter allen Umständen). Ein Beispiel:

```
\makeatletter  \@beginparpenalty=9999  \makeatother
```

TEX-Zähler benötigen diese ungewöhnliche Form der Wertzuweisung, und da alle drei ein @-Zeichen im Namen haben, müssen sie in \makeatletter und \makeatother eingebettet werden, wenn die Zuweisung in der Präambel geschieht.

Viele Umgebungen sind als Listen implementiert

Man muss sich dessen bewusst sein, dass solch eine Einstellung global für alle Umgebungen gilt, die auf der generischen list-Umgebung basieren (sofern sie nicht im Argument *dekl* vorgenommen wurde). Außerdem werden verschiedene LATEX-Umgebungen mithilfe von list definiert (z.B. quote, quotation, center, flushleft und flushright). Diese Umgebungen sind „Listen“ mit einem einzigen Listenpunkt, bei denen der Befehl \item[] in der Definition der Umgebung angegeben wird. Sie werden hauptsächlich deshalb intern als Listen definiert, damit sie denselben vertikalen Abstand bekommen wie andere abgesetzte Objekte. Dies trägt zu einem einheitlichen Layout bei.

Ein Beispiel hierfür ist die Umgebung quote, deren Definition die Ränder links und rechts gleichsetzt. Die einfache Variante Quote, die unten gezeigt wird, ist – bis auf die Anführungszeichen um den Text – identisch mit quote. Zu beachten sind die besonderen Vorsichtsmaßnahmen, die erforder-

lich sind, um unerwünschten Leerraum vor (`\ignorespaces`) und hinter dem Text (`\unskip`) zu vermeiden. Durch Platzierung der Anführungsstriche in Boxen ohne horizontale Ausdehnung laufen diese in den Seitenrand. Diesen Trick sollte man sich merken: Bei einer Box, der man die horizontale Ausdehnung null zuweist, und rechtsbündiger Ausrichtung des Inhalts ragt der Text nach links heraus.

Bsp. 3-3-25

```
\usepackage[ngerman]{babel}
\newenvironment{Quote}%
 {\begin{list}{}%
              {\setlength\rightmargin{\leftmargin}}%
  \item[]\makebox[0pt][r]{"`}\ignorespaces}%
 {\unskip\makebox[0pt][l]{"'}\end{list}}
\ldots\ vorangehender Text.
\begin{Quote}
   Zitierter Text, weiterer zitierter Text.
\end{Quote}
Nachfolgender Text \ldots
```

... vorangehender Text.

"`Zitierter Text, weiterer zitierter Text."'

Nachfolgender Text ...

Im restlichen Teil dieses Kapitels werden eine Reihe von verschiedenen „beschreibenden“ Listen generiert, um die verschiedenen Möglichkeiten zu erläutern, welche die generische `list`-Umgebung bietet. Den Anfang macht die Standarddefinition der `description`-Umgebung, wie sie in den Standardklassen von LaTeX, wie article oder report, vorkommt.[1]

```
\newenvironment{description}
    {\begin{list}{}{\setlength\labelwidth{0pt}%
                    \setlength\itemindent{-\leftmargin}%
                    \let\makelabel\descriptionlabel}}
    {\end{list}}
```

Die Gründe hinter dieser Definition zeigt Abbildung 3.3 auf Seite 153, welche die Beziehung zwischen den verschiedenen Listenparametern erläutert. Die erste Parametereinstellung betrifft `\labelwidth`: Er wird auf null gesetzt, was bedeutet, dass kein Leerraum für das Label reserviert wird. Bei der Formatierung des Labels wird der Text der ersten Zeile damit nach rechts verschoben, damit der erforderliche Leerraum entsteht. Danach wird der Parameter `\itemindent` auf den negierten Wert von `\leftmargin` gesetzt. Im Ergebnis wird das erste Zeichen genau bis an den Rand der umschließenden Umgebung verschoben, alle weiteren Zeilen werden weiterhin um `\leftmargin` eingezogen. Die letzte Deklaration setzt `\makelabel` mit dem Befehl `\descriptionlabel` gleich. Der Befehl `\makelabel` wird von der `list`-Umgebung aufgerufen, sobald das Label eines Listenpunktes formatiert werden muss. Er hat ein Argument (das Label) und dient dazu, eine formatierte Version dieses Argumentes zu erzeugen. Um die Definition der `description`-Umgebung abzuschließen, muss als Letztes eine geeignete Definition für `\descriptionlabel` bereit gestellt werden. Dieser Umweg ist nützlich, weil er eine geänderte Labelformatierung ermöglicht, ohne dass der Rest der Definition der Umgebung modifiziert werden muss.

[1] Ein Blick in `article.cls` oder `report.cls` zeigt eine leicht optimierte Kodierung, die beispielsweise Low-Level-Zuordnungen nutzt, anstelle von `\setlength`. Vom Konzept her sind beide Definitionen jedoch identisch.

Wie sollte \descriptionlabel definiert werden? Der Befehl stellt die Formatierung für das Label bereit. Mit der Standardumgebung description wird das Label normalerweise fett formatiert. Zu beachten ist, dass das Label vom folgenden Text durch einen Abstand der Breite \labelsep getrennt ist. Aufgrund der obigen Parametereinstellungen beginnt dieser Text im Seitenrand. Ohne Korrektur würde das Label in den Rand gerückt (um die Breite von \labelsep). Um dieses Ergebnis zu verhindern, hat die Standarddefinition für den Befehl \descriptionlabel folgende seltsame Definition, durch die das Label erst nach rechts gerückt und dann formatiert wird:

```
\newcommand*\descriptionlabel[1]
    {\hspace{\labelsep}\normalfont\bfseries #1}
```

Um diese Abhängigkeit zu beseitigen, müsste man die Einstellung von \itemindent ändern, um \labelsep zu berücksichtigen, was an sich nicht schwierig wäre. Man könnte dieses Verhalten als historisches Artefakt bezeichnen, aber viele Dokumente verlassen sich auf diese etwas obskure Funktion. Daher ist es schwierig, die Einstellung im LaTeX-Kernel zu ändern, ohne diese Dokumente zu zerstören.

Mit den Parametereinstellungen der Standardumgebung description beginnt der Text der ersten Zeile bei kurzen Labels vor dem Text der übrigen Zeilen. Wenn der Text stets einen gewissen Betrag eingerückt werden soll, kann man eine Definition ähnlich dem folgenden Beispiel verwenden. Hier wird \labelwidth auf 40pt und \leftmargin auf \labelwidth plus \labelsep gesetzt. Dies bedeutet, dass sich \makelabel selbst nur um die Formatierung des Labels kümmern muss. Aber da der Befehl jetzt eine positive nominale Labelbreite generiert, muss definiert werden, was passieren soll, wenn das Label klein ist. Mit \hfil kann man angeben, wo zusätzlicher Weißraum eingefügt werden soll.

```
\usepackage{calc}
\newenvironment{Description}
   {\begin{list}{}{\let\makelabel\Descriptionlabel
      \setlength\labelwidth{40pt}%
      \setlength\leftmargin{\labelwidth+\labelsep}}}%
   {\end{list}}
\newcommand*\Descriptionlabel[1]{\textsf{#1:}\hfil}
\begin{Description}
\item[Beschreibung]
  Rücksprung aus einer Funktion. Bei Verwendung
  im Hauptprogramm endet das Programm.
\item[Fehler]  Keine.
\item[Rückgabewerte]
  \mbox{}\\
  Alle Parameter werden an die aufrufende
  Komponente zurückgegeben.
\end{Description}
```

Beschreibung: Rücksprung aus einer Funktion. Bei Verwendung im Hauptprogramm endet das Programm.

Fehler: Keine.

Rückgabewerte: Alle Parameter werden an die aufrufende Komponente zurückgegeben.

Bsp. 3-3-26

Dieses Beispiel zeigt ein typisches Problem beschreibender Listen, wenn der zu beschreibende *Term* breiter ist als der vorgesehene Raum für das Label. Mit der obigen Definition ragt der Text des Terms in den Text der *Be-*

schreibung hinein. Dieser Effekt ist häufig unerwünscht. Um das optische Erscheinungsbild der Liste zu verbessern, wurde einer der beschreibenden Teile in der nächsten Zeile gestartet. Ein Zeilenumbruch wurde erzwungen, indem eine leere Box, gefolgt vom Befehl '\\', auf dieselbe Zeile platziert wurde.

Im übrigen Teil dieses Abschnitts werden verschiedene Möglichkeiten untersucht, die Breite und gegenseitige Positionierung von Begriff und Beschreibung festzulegen. Zunächst lässt sich die Breite des Labels ändern. Dazu wird die Umgebung mit einem Argument deklariert, das die gewünschte Breite des Labelfeldes angibt (normalerweise die Breite des längsten Begriffs). Man beachte auch die neue Definition des Befehls `\makelabel`, die angibt, wie das Label formatiert wird. Da diese neue Definition innerhalb der Definition[1] für die Umgebung `altDescription` steht, muss der Argumentplatzhalter `#` durch `##` dargestellt werden, um LaTeX anzuzeigen, dass dieser Platzhalter nicht zu dem Argument der äußeren Umgebung, sondern zu dem Argument von `\makelabel` gehört. Der Parameter `\labelwidth` wird auf die Breite des Argumentes der Umgebung gesetzt, nachdem es durch `\makelabel` formatiert wurde. Auf diese Weise werden Formatierungsbefehle für das Label berücksichtigt, die dessen Breite ändern könnten.

```
\usepackage{calc}
\newenvironment{altDescription}[1]
  {\begin{list}{}%
    {\renewcommand\makelabel[1]{\textsf{##1:}\hfil}%
     \settowidth\labelwidth{\makelabel{#1}}%
     \setlength\leftmargin{\labelwidth+\labelsep}}}%
  {\end{list}}

\begin{altDescription}{Rückgabewerte}
\item[Beschreibung]
  Rücksprung aus einer Funktion. Bei Verwendung
  im Hauptprogramm endet das Programm.
\item[Fehler]  Keine.
\item[Rückgabewerte]
  Alle Parameter werden an die aufrufende
  Komponente zurückgegeben.
\end{altDescription}
```

Beschreibung: Rücksprung aus einer Funktion. Bei Verwendung im Hauptprogramm endet das Programm.

Fehler: Keine.

Rückgabewerte: Alle Parameter werden an die aufrufende Komponente zurückgegeben.

Bsp. 3-3-27

Eine ähnliche Umgebung (aber unter Verwendung eines optionalen Argumentes) wird in Beispiel A-1-9 auf Seite 883 gezeigt. Verschiedene Listen auf der gleichen Seite, die unterschiedlich breite Labelfelder haben, können jedoch nach typographischen Gesichtspunkten unakzeptabel sein. Daher besteht eine weitere Möglichkeit darin, die Länge des Begriffs auszuwerten. Wenn der Begriff länger ist als `\labelwidth`, wird zusätzlich eine leere Box eingefügt, so dass die Beschreibung erst in der nächsten Zeile beginnt. Dies entspricht dem üblichen Verfahren für die Darstellung von Befehlsoptionen in UNIX-Handbüchern.

Zur Demonstration dieses Verfahrens wird im nächsten Beispiel die in Beispiel 3-3-26 definierte `Description`-Umgebung wiederverwendet, jedoch

[1] Dies geschieht aus Demonstrationszwecken. In der Regel ist die Lösung mit externem Namen vorzuziehen, wie bei `\Descriptionlabel` in Beispiel 3-3-26 auf der vorherigen Seite.

mit einer abgewandelten Definition für den Befehl \Descriptionlabel.

```
\usepackage{calc,ifthen}   \newlength{\Mylen}
% Definition der Description-Umgebung wie zuvor
\newcommand*\Descriptionlabel[1]{%
  \settowidth\Mylen{\textsf{#1:}}%   Breite bestimmen
  \ifthenelse{\lengthtest{\Mylen > \labelwidth}}%
      {\parbox[b]{\labelwidth}%  Term  > Labelbreite
         {\makebox[0pt][l]{\textsf{#1:}}\\\mbox{}}}%
      {\textsf{#1:}}%            Term <= Labelbreite
  \hfill}
\begin{Description}
\item[Beschreibung]
  Rücksprung aus einer Funktion. Bei Verwendung
  im Hauptprogramm endet das Programm.
\item[Fehler]  Keine.
\item[Rückgabewerte]
  Alle Parameter werden an die aufrufende
  Komponente zurückgegeben.
\end{Description}
```

Bsp. 3-3-28

Beschreibung: Rücksprung aus einer Funktion. Bei Verwendung im Hauptprogramm endet das Programm.

Fehler: Keine.

Rückgabewerte: Alle Parameter werden an die aufrufende Komponente zurückgegeben.

Der Befehl \Descriptionlabel weist der Längenvariable \Mylen die Breite des Labels zu. Anschließend wird diese Länge mit \labelwidth verglichen. Wenn das Label nicht breiter ist als \labelwidth, wird es in derselben Zeile gesetzt wie die Beschreibung; andernfalls wird es in einer Box der Breite 0 gesetzt, so dass der Inhalt so weit wie notwendig über den rechten Rand der Box herausragt. Es wird einer am unteren Rand ausgerichteten \parbox platziert, gefolgt von einem erzwungenem Zeilenumbruch, so dass die Beschreibung eine Zeile tiefer beginnt. Dieses etwas komplizierte Verfahren ist notwendig, weil \makelabel, und damit auch \Descriptionlabel, in einem streng horizontalem Kontext ausgeführt werden, in dem vertikale Abstände oder \\-Befehle keine Wirkung haben.

Eine weitere Möglichkeit ist, mehrzeilige Labels zuzulassen.

```
\usepackage{calc}
% Definition der Description-Umgebung wie zuvor
\newcommand*\Descriptionlabel[1]
   {\raisebox{0pt}[1ex][0pt]%
      {\makebox[\labelwidth][l]%
         {\parbox[t]{\labelwidth}%
                    {\hspace{0pt}\textsf{#1:}}}}}
\begin{Description}
\item[Beschreibung]
  Rücksprung aus einer Funktion. Bei Verwendung
  im Hauptprogramm endet das Programm.
\item[Fehler]  Keine.
\item[Rückgabewerte]
  Alle Parameter werden an die aufrufende
  Komponente zurückgegeben.
\end{Description}
```

Bsp. 3-3-29

Beschreibung: Rücksprung aus einer Funktion. Bei Verwendung im Hauptprogramm endet das Programm.

Fehler: Keine.

Rückgabewerte: Alle Parameter werden an die aufrufende Komponente zurückgegeben.

Im vorangegangenen Beispiel wurde als Grundlage wieder die Umgebung `Description` und dazu eine weitere neue Definition des Befehls `\Descriptionlabel` verwendet. Dieser Befehl soll bewirken, dass längere Labels über mehrere Zeilen umbrochen werden können. Um eine Trennung des ersten Wortes in einem Absatz zu ermöglichen, müssen bestimmte Vorsichtsmaßnahmen beachtet werden. Dazu wurde in der Definition der Befehl `\hspace{0pt}` benutzt. Der Text wird in einer Absatzbox der richtigen Breite `\labelwidth` gesetzt, die dann am oberen und linken Rand in einer weiteren Box ausgerichtet wird. Diese wird wiederum selbst in einer Box platziert, welche die Höhe 1ex und keine Tiefe hat. Auf diese Weise erkennt LaTeX nicht, dass sich der Text unter die erste Zeile erstreckt.

Das letzte Beispiel befasst sich mit der Definition von nummerierten Listen. Wenn man den Befehl `\usecounter` in die Deklaration der `list`-Umgebung einbettet, kann man eine Umgebung mit einem automatisch erhöhten Zähler erstellen. Diese Funktionalität wird anhand der Umgebung `Notes` gezeigt, die eine Reihe von Anmerkungen erzeugt. In diesem Fall stellt das erste Argument der `list`-Umgebung den automatisch generierten Text für den Begriff zur Verfügung.

Nachdem der Zähler `notes` deklariert ist, wird das Standardlabel der Umgebung `Notes` definiert. Es besteht aus dem in Kapitälchen gesetzten Wort ANMERKUNG, gefolgt von dem Wert des Zählers `notes`, der als arabische Ziffer mit darauf folgendem Punkt dargestellt wird. Als Nächstes wird `\labelsep` auf einen relativ hohen Wert gesetzt, während die Parameter `\itemindent`, `\leftmargin` und `\labelwidth` so angepasst werden, dass das Label trotzdem am linken Rand beginnt. Schließlich sorgt die bereits erwähnte `\usecounter`-Deklaration dafür, dass der Zähler `notes` bei jedem `\item`-Befehl hochgesetzt wird.

```
\newcounter{notes}
\newenvironment{Notes}
    {\begin{list}{\textsc{Anmerkung} \arabic{notes}.}%
                 {\setlength\labelsep{10pt}%
                  \setlength\itemindent{10pt}%
                  \setlength\leftmargin{0pt}%
                  \setlength\labelwidth{0pt}%
                  \usecounter{notes}}}%
    {\end{list}}
\begin{Notes}
\item Dies ist der Text der ersten Anmerkung.
\item Dies ist der Text der zweiten Anmerkung.
\end{Notes}
```

ANMERKUNG 1. Dies ist der Text der ersten Anmerkung.

ANMERKUNG 2. Dies ist der Text der zweiten Anmerkung.

Bsp. 3-3-30

3.4 Wortwörtlicher Text

Oft muss man Informationen unverändert, also „wie über die Tastatur eingegeben“, darstellen. Diese Funktionalität wird über die Standardumgebung `verbatim` von LaTeX realisiert. Um den Benutzer zu leiten, kann es jedoch nützlich sein, dabei einige Textteile besonders hervorzuheben, z.B. durch Nummerierung der Zeilen. Nach und nach wurden eine Reihe von Paketen entwickelt,

die sich mit der einen oder anderen Sonderfunktion befassen – leider jedoch jedes mit einer eigenen Syntax.

In diesem Abschnitt werden ein paar solcher Pakete kurz beleuchtet. Da sie in der Vergangenheit intensiv genutzt wurden, findet man sie in älteren Quelldokumenten im Internet oder hat das eine oder andere auch selbst schon verwendet. Der dann folgende Teil konzentriert sich aber auf das Paket fancyvrb von Timothy Van Zandt, das all diese Funktionen und viele andere unter dem Dach eines einzigen, leicht anpassbaren Paketes vereint.

Anschließend folgt eine Besprechung des Paketes listings, das eine vielseitige Umgebung bereitstellt, in der Computerlistings in vielen Computersprachen auf ansprechende Weise ausgegeben werden können.

3.4.1 Einfache Verbatim-Erweiterungen

Das Paket alltt (von Leslie Lamport) definiert die Umgebung `alltt`. Diese verhält sich wie die Umgebung `verbatim`, bis auf die Tatsache, dass der Backslash „\“ und die Klammern „{“ und „}“ ihre übliche Bedeutung behalten. Dadurch können andere Umgebungen und Befehle innerhalb der Umgebung `alltt` verwendet werden. Eine ähnliche Funktionalität bietet das Schlüsselwort `commandchars` der Umgebung fancyvrb (siehe Seite 169).

```
\usepackage{alltt}
\begin{alltt}
Man kann den \emph{Font} wechseln.
Einige Sonderzeichen: # $ % ^ & ~ _
\end{alltt}
```

Bsp. 3-4-1

```
Man kann den Font wechseln.
Einige Sonderzeichen: # $ % ^ & ~ _
```

In Dokumenten, in denen zahlreiche `\verb`-Befehle benötigt werden, ist das Quelldokument bald schwer zu lesen. Aus diesem Grund führt das Paket doc, das in Kapitel 14 beschrieben wird, eine Abkürzung ein, mit der über ein Sonderzeichen Anfang und Ende der wortwörtlichen Ausgabe gekennzeichnet werden können, ohne dass diese ständig wieder mit `\verb` eingeleitet werden muss. Diese Funktion ist auch in einem separaten Paket namens shortvrb verfügbar. Dieselbe Funktionalität bietet auch fancyvrb, jedoch wird hier eine etwas andere Syntax verwendet (siehe Seite 177).

```
\usepackage{shortvrb}
\MakeShortVerb{\|}
Mit |\MakeShortVerb| werden Quelldateien
leichter lesbar.
\DeleteShortVerb{\|}\MakeShortVerb{\+}
Und mit der Deklaration +\DeleteShortVerb{\|}+
kann man das Zeichen  +|+ wieder auf die
normale Bedeutung zurücksetzen.
```

Bsp. 3-4-2

Mit `\MakeShortVerb` werden Quelldateien leichter lesbar. Und mit der Deklaration `\DeleteShortVerb{\|}` kann man das Zeichen | wieder auf die normale Bedeutung zurücksetzen.

Die Variante `\MakeShortVerb*` führt dieselbe Abkürzung für den Befehl `\verb*` ein. Dies wird im nächsten Beispiel gezeigt.

```
\usepackage{shortvrb}  \MakeShortVerb*{\+}
Anstelle von \verb*/ / kann man jetzt + + schreiben.
```

Bsp. 3-4-3

Anstelle von ␣ kann man jetzt ␣ schreiben.

Das Paket verbatim (von Rainer Schöpf) reimplementiert die LaTeX-Umgebungen `verbatim` und `verbatim*`. Einer seiner größten Vorteile ist, dass es unformatierte Texte beliebiger Länge zulässt, was mit den gleichnamigen Standardumgebungen von LaTeX nicht möglich ist. Außerdem stellt es die Umgebung `comment` zur Verfügung, die jeglichen Text zwischen den Befehlen `\begin{comment}` und `\end{comment}` übergeht. Weiterhin bietet das Paket die Möglichkeit, eigene Erweiterungen zu implementieren, so dass man angepasste, Verbatim-ähnliche Umgebungen definieren kann.

Ein paar dieser Erweiterungen werden im Paket moreverb (von Angus Duggan) realisiert. Es bietet einige interessante Verbatim-ähnliche Befehle zum Schreiben und Lesen von Dateien sowie verschiedene Umgebungen zum Erzeugen von Auflistungen und zum Setzen von Tabulatoren. Alle diese Erweiterungen sind auch in konsistenter Art und Weise mit dem fancyvrb-Paket verfügbar, deshalb folgt an dieser Stelle nur ein einziges Beispiel, das einen Eindruck der Syntax vermittelt, die das Paket moreverb verwendet.

Bsp. 3-4-4

Text vor der Listenumgebung.

```
Die␣listing-Umgebung␣nummeriert
4  die␣in␣ihr␣enthaltenen␣Zeilen.
   Sie␣hat␣ein␣optionales␣Argument,
6  welches␣den␣Zählschritt␣zwischen
   nummerierten␣Zeilen␣angibt␣(die
8  Zeile␣mit␣der␣Nummer␣1␣wird␣immer
   nummeriert),␣und␣ein
10 obligatorisches␣Argument␣--␣die
   Nummer␣der␣Startzeile.␣␣Die
12 Sternform␣macht␣Leerzeichen
   sichtbar.
```

Text zwischen Listenumgebungen.

```
14 Die listingcont-Umgebung fährt da
   fort, wo die vorige
16 listing-Umgebung aufgehört hat.
   Beide Umgebungen setzen Tabstopps
18 mit dem voreingestellten Abstand
   von 8 Leerzeichen ein.
```

Text nach Listenumgebungen.

```
\usepackage{verbatim,moreverb}
Text vor der Listenumgebung.
\begin{listing*}[2]{3}
Die listing-Umgebung nummeriert
die in ihr enthaltenen Zeilen.
Sie hat ein optionales Argument,
welches den Zählschritt zwischen
nummerierten Zeilen angibt (die
Zeile mit der Nummer 1 wird immer
nummeriert), und ein
obligatorisches Argument -- die
Nummer der Startzeile.  Die
Sternform macht Leerzeichen
sichtbar.
\end{listing*}
Text zwischen Listenumgebungen.
\begin{listingcont}
Die listingcont-Umgebung fährt da
fort, wo die vorige
listing-Umgebung aufgehört hat.
Beide Umgebungen setzen Tabstopps
mit dem voreingestellten Abstand
von 8 Leerzeichen ein.
\end{listingcont}
Text nach Listenumgebungen.
```

3.4.2 upquote – Anführungsstriche in Programmlistings

Die Schrift Computer Modern Typewriter, die standardmäßig zum Formatieren von „Verbatim“, d.h. der wortgetreuen Wiedergabe, von Texten verwendet wird, ist eine sehr gut lesbare dicktengleiche Schrift. Durch ihre kleine Lauflänge eignet sie sich gut für Computerprogramme und ähnliche Texte. Ein Vergleich dieser Schrift mit anderen dicktengleichen Schriften ist in Abschnitt 7.7.4 aufgeführt.

Ein potentielles Problem gibt es jedoch, wenn diese Schrift zur Darstellung von Programmlistings und Ähnlichem verwendet wird: Die meisten Leute erwarten, dass rechte Anführungsstriche durch einen senkrechten Anführungsstrich (d.h. ') und linke oder rückwärtige Anführungsstriche durch ein Grave-Akzentzeichen (d.h. `) dargestellt werden. Die Schrift Computer Modern Typewriter zeigt jedoch echte linke und rechte geschweifte Anführungsstriche an (wie man sie in einer normalen Textschrift erwarten würde). Tatsächlich verhalten sich auch die meisten anderen Typewriter-Schriften so, wenn sie mit LaTeX verwendet werden. Dies führt zu etwas ungewöhnlichen Ergebnissen, die manche Leute schwer verstehen können. Das folgende Beispiel zeigt das Standardverhalten von drei bekannten Typewriter-Schriften: LuxiMono, Courier und Computer Modern Typewriter.

```
\usepackage[scaled=0.85]{luximono}
\raggedright
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
\par \renewcommand\ttdefault{pcr}
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
\par \renewcommand\ttdefault{cmtt}
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
```

```
TEST=‘ls -l |awk ’{print $3}’‘
TEST=‘ls -l |awk ’{print $3}’‘
TEST=‘ls -l |awk ’{print $3}’‘
```

Bsp. 3-4-5

Dieses Verhalten lässt sich ändern, indem man das Paket upquote (von Michael Covington) lädt, das in \verb oder der verbatim-Umgebung anstelle der üblichen geschweiften linken und rechten Anführungsstriche die Zeichen \textasciigrave und \textquotesingle aus dem textcomp-Paket verwendet. In normaler Schreibmaschinenschrift können trotzdem noch geschweifte Anführungsstriche verwendet werden, vgl. hierzu den englischen Text in der letzten Zeile des Beispiels.

```
\usepackage[scaled=0.85]{luximono}
\usepackage{upquote}
\raggedright
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
\par \renewcommand\ttdefault{pcr}
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
\par \renewcommand\ttdefault{cmtt}
\verb+TEST=‘ls -l |awk ’{print $3}’‘+
\par \texttt{but ‘text’ is unaffected!}
```

```
TEST=`ls -l |awk '{print $3}'`
TEST=`ls -l |awk '{print $3}'`
TEST=`ls -l |awk '{print $3}'`
but ‘text’ is unaffected!
```

Bsp. 3-4-6

Wie dieses Kapitel zeigt, arbeitet das Paket sehr gut mit den „Verbatim“-Erweiterungen zusammen. Einzige Ausnahme bildet das listings-Paket, mit dessen Scan-Mechanismus es kollidiert. Wenn diese Art der Anführungsstriche trotzdem mit listings verwendet werden soll, können sie mit dem Schlüsselwort upquote des Befehls \lstset aktiviert werden.

```
\usepackage{textcomp}
\usepackage{listings} \lstset{upquote}
\begin{lstlisting}[language=ksh]
TEST=‘ls -l |awk ’{print $3}’‘
\end{lstlisting}
```

```
TEST=`ls␣-l␣|awk␣'{print␣$3}'`
```

Bsp. 3-4-7

3.4.3 fancyvrb – Anpassungsfähige Verbatim-Umgebungen

Das Paket fancyvrb von Timothy Van Zandt (das inzwischen von Denis Girou und Sebastian Rahtz gepflegt wird) bietet eine Reihe von sehr anpassbaren Umgebungen und Befehle, mit denen wortwörtliche Ausgaben formatiert und verändert werden können.

Die Arbeitsweise des Paketes basiert auf einer zeilenweisen syntaktischen Analyse der Umgebung oder der Datei (das Konzept stammt ursprünglich vom verbatim-Paket), wodurch Zeilen auf unterschiedliche Weise aufbereitet werden können. Durch Integration verschiedener Funktionen aus anderen Paketen bietet es eine echte universelle Produktionsumgebung mit gemeinsamen Syntaxregeln.

Die wichtigste Umgebung, die das Paket bereitstellt, ist die Umgebung `Verbatim`, die, wenn sie ohne Anpassungen benutzt wird, LaTeXs Standardumgebung `verbatim` ähnelt. Der Hauptunterschied besteht darin, dass sie ein optionales Argument besitzt, dem in Form einer Schlüssel-Wert-Syntax Informationen zur Anpassung der Umgebung mitgegeben werden können. Es gibt jedoch eine Einschränkung, die man nicht vergessen sollte: Die linke Klammer des optionalen Argumentes muss in derselben Zeile stehen wie `\begin`. Ansonsten wird das optionale Argument nicht erkannt und stattdessen wortwörtlich ausgegeben.

Es sind mehr als 30 Schlüsselwörter verfügbar, deren Verwendung und mögliche Werte im Folgenden ausführlich besprochen werden.

Gegen Ende dieses Abschnitts werden eine Reihe von alternativen Umgebungen und Befehlen erörtert. Auch sie können über das Schlüssel-Wert-Verfahren verändert werden. Zum Schluss wird auf die Möglichkeit eingegangen, eigene Varianten zu definieren.

Schlüsselwörter zur Anpassung der Formatierung

Zur Änderung der Fonts, die von den Verbatim-Umgebungen des Paketes fancyvrb verwendet werden, stehen vier Schlüsselwörter zur Verfügung, die den vier Achsen des NFSS entsprechen. Das Schlüsselwort `fontfamily` gibt die zu verwendende Schriftfamilie an. Voreingestellt ist `Computer Modern Typewriter`, so dass sich die Umgebungen ohne Verwendung von Schlüsselwörtern ähnlich der Standardumgebung `verbatim` von LaTeX verhalten. Der Wert des Schlüsselwortes kann jedoch jede Schriftfamilie in NFSS-Schreibweise sein, z.B. `pcr` für `Courier` oder `cmss` für Computer Modern Sans, auch wenn letztere keine dicktengleiche Schrift ist, wie sie normalerweise für eine wortwörtliche Ausgabe verwendet würde. Das Schlüsselwort erkennt auch die speziellen Werte `tt`, `courier` und `helvetica` und transferiert sie intern in die NFSS-Nomenklatur.

Die Formatierung von wortwörtlichen Ausgaben umfasst auch Sonderzeichen, wie „\“; hier ist darauf zu achten, dass solche Zeichen im gewählten Zeichensatz tatsächlich vorhanden sind. Dies sollte kein Problem darstellen, wenn eine Fontkodierung wie `T1` verwendet wird (diese kann man mithilfe des Paketes fontenc laden). Bei der Standardfontkodierung `OT1` von LaTeX ist dies jedoch nicht so einfach, denn in dieser Kodierung enthalten nur ein paar dicktengleiche Schriften, wie die voreingestellte Typewriter-Schrift, alle diese

Sonderzeichen. Welche Form von falscher Ausgabe erzeugt werden könnte, zeigt die zweite Zeile im nächsten Beispiel.

```
\usepackage{fancyvrb}
\usepackage[OT1,T1]{fontenc}
\fontencoding{OT1}\selectfont
\begin{Verbatim}[fontfamily=tt]
‘tt’ funktioniert in OT1: \sum_{i=1}^n
\end{Verbatim}
\begin{Verbatim}[fontfamily=helvetica]
‘Helvetica’ geht in OT1 nicht: \sum_{i=1}^n
\end{Verbatim}
\fontencoding{T1}\selectfont
\begin{Verbatim}[fontfamily=helvetica]
...  aber in T1 schon: \sum_{i=1}^n
\end{Verbatim}
```

```
‘tt’ funktioniert in OT1: \sum_{i=1}^n

‘Helvetica’ geht in OT1 nicht: “sum¨–i=1´ˆn

...  aber in T1 schon: \sum_{i=1}^n
```

Bsp. 3-4-8

Da alle Beispiele in diesem Buch mit der T1-Kodierung formatiert sind, wird dieses Problem nicht an anderen Stellen im Buch auftreten. Trotzdem sollte man sich dieser Gefahr bewusst sein. Sie ist ein weiterer guter Grund, um anstelle von TEXs Original-Fontkodierung T1 zu verwenden; eine ausführliche Erörterung gibt Abschnitt 7.2.4 auf Seite 347 wieder.

Die anderen drei Schlüsselwörter, die sich auf die Fonteinstellungen beziehen, sind `fontseries`, `fontshape` und `fontsize`. Sie erben die aktuellen NFSS-Einstellungen aus dem umgebenden Text, wenn nichts anderes angegeben ist. Die ersten beiden erwarten Werte, die an `\fontseries` bzw. `\fontshape` weitergereicht werden können (z.B. `bx` für eine fette Schriftserie mit großer Laufweite oder `it` für einen *kursiven* Schriftschnitt), während `fontsize` sich anders verhält. Es erwartet als Wert einen der höheren NFSS-Befehle für die Angabe der Schriftgröße – z.B. `\small`. Wenn das Paket relsize verfügbar ist, kann alternativ eine Schriftgrößenänderung relativ zur aktuellen Schriftgröße des Textes als Wert verwendet werden, wie etwa `\relsize{-2}` im folgenden Beispiel.

```
\usepackage{relsize,fancyvrb}
\begin{Verbatim}[fontsize=\relsize{-2}]
  \sum_{i=1}^n
\end{Verbatim}
Eine Textzeile für die Darstellung des Textkörpers.
\begin{Verbatim}[fontshape=sl,fontsize=\Large]
  \sum_{i=1}^n
\end{Verbatim}
```

```
\sum_{i=1}^n

Eine Textzeile für die Darstellung des Textkörpers.

\sum_{i=1}^n
```

Bsp. 3-4-9

Eine allgemeinere Form, die Formatierung anzupassen, bietet das Schlüsselwort `formatcom`, das jeglichen LATEX-Code akzeptiert und diesen am Anfang der Umgebung ausführt. Um beispielsweise die wortwörtliche Ausgabe einzufärben, könnte man ihm einen Wert wie `\color{blue}` zuweisen. Es ist auch möglich, jede Textzeile separat zu bearbeiten, indem der Befehl `\FancyVerbFormatLine` in geeigneter Weise neu definiert wird. Dieser Befehl wird für jede Zeile ausgeführt, wobei er den Text aus der Zeile als Argument

erhält. Im nächsten Beispiel wird jede zweite Zeile blau eingefärbt. Dieses Ergebnis resultiert aus der Überprüfung des Zählers `FancyVerbLine`, der von der Umgebung automatisch bereitgestellt wird und die aktuelle Zeilennummer enthält.

Bsp. 3-4-10

```
Diese Zeile sollte blau werden,
 während  diese schwarz ist. Und
er sieht man, dass gobble nicht
r Leerzeichen löscht.
```

```
\usepackage{ifthen,color,fancyvrb}
\renewcommand\FancyVerbFormatLine[1]
  {\ifthenelse{\isodd{\value{FancyVerbLine}}}%
              {\textcolor{blue}{#1}}{#1}}
\begin{Verbatim}[gobble=2]
  Diese Zeile sollte blau werden,
   während  diese schwarz ist. Und
hier sieht man, dass gobble nicht
nur Leerzeichen löscht.
\end{Verbatim}
```

Wie das vorangegangene Beispiel zeigt, kann das Schlüsselwort `gobble` verwendet werden, um (bis zu neun) Zeichen oder Leerzeichen am Anfang jeder Zeile zu entfernen. Dies ist dann sehr nützlich, wenn alle Zeilen in den Umgebungen eingezogen sind und man die zusätzlichen Leerzeichen löschen möchte, die durch den Einzug entstanden sind. Manchmal ist das Gegenteil erwünscht: Jede Zeile sollte um einen bestimmten Betrag eingezogen werden. Zum Beispiel sind alle Verbatim-Umgebungen in diesem Buch um `24pt` eingerückt. Dieser Einzug wird von dem Schlüsselwort `xleftmargin` gesteuert. Daneben gibt es das Schlüsselwort `xrightmargin`, das den rechten Einzug festlegt. Es wird allerdings nur selten benötigt, weil die Zeilen von wortwörtlichen Ausgaben nicht umbrochen werden. Ihre visuelle Auswirkung lässt sich vermutlich nur anhand von „overfull box"-Meldungen[1] nachvollziehen (es sei denn, man verwendet Rahmen, wie unten erörtert). Diese zeigen an, dass der Text in den rechten Rand hineinläuft. Möglicherweise nützlicher ist das boolesche Schlüsselwort `resetmargins`, welches festlegt, ob existierende Einzüge von äußeren Umgebungen ignoriert werden sollen.

Bsp. 3-4-11

- Normaler Einzug links:

```
Wortwörtliche Ausgabe!
```

- Auf beiden Seiten kein Einzug:

```
Wortwörtliche Ausgabe!
```

```
\usepackage{fancyvrb}
\begin{itemize}  \item  Normaler Einzug links:
\begin{Verbatim}[frame=single,xrightmargin=1pc]
Wortwörtliche Ausgabe!
\end{Verbatim}
  \item Auf beiden Seiten kein Einzug:
\begin{Verbatim}[resetmargins=true,
                 frame=single]
Wortwörtliche Ausgabe!
\end{Verbatim}
\end{itemize}
```

Das vorangegangene Beispiel demonstriert eine typische Verwendung des Schlüsselwortes `frame`: einen Rahmen um die wortwörtliche Ausgabe ziehen.

[1] Ob überlaufende Zeilen innerhalb einer Verbatim-Umgebung angezeigt werden, wird vom Schlüsselwort `hfuzz` gesteuert, das den Standardwert `2pt` hat. Eine Warnung wird nur ausgegeben, wenn die Boxen um einen größeren Betrag in den Rand hineinragen, als den, der als Wert des Schlüsselwortes angegeben ist.

Mit weiteren Werten lassen sich andere Rahmen erzeugen. Voreingestellt ist `none`, d.h.: kein Rahmen. `topline`, `bottomline` oder `leftline` erzeugen jeweils eine einzelne Linie auf der angegebenen Seite.[1] Der Wert `lines` erzeugt oben und unten eine Linie und `single` zeichnet, wie schon Beispiel 3-4-11 zeigte, einen vollständigen Rahmen. In allen Fällen kann die Dicke der Linien durch Angabe eines Wertes im Schlüsselwort `framerule` geändert werden (voreingestellt ist `0.4pt`). Der Abstand zwischen Linien und Text ist über `framesep` steuerbar (voreingestellt ist der aktuelle Wert von `\fboxsep`).

Wenn das color-Paket verfügbar ist, können die Linien eingefärbt werden. Dazu wird das Schlüsselwort `rulecolor` benötigt (Standardeinstellung: schwarz). Bei Verwendung eines kompletten Rahmens kann auch der Freiraum zwischen Text und Rahmen eingefärbt werden, und zwar mit `fillcolor`.

```
Gerahmte Ausgabe!
```

```
\usepackage{color,fancyvrb}
\begin{Verbatim}[frame=single,rulecolor=\color{blue},
  framerule=3pt,framesep=1pc,fillcolor=\color{yellow}]
Gerahmte Ausgabe!
\end{Verbatim}
```

Bsp. 3-4-12

Leider gibt es keine direkte Methode, um den gesamten Hintergrund einzufärben. Die größte Näherung erreicht man, indem man `\colorbox` innerhalb von `\FancyVerbFormatLine` verwendet. Dieser Ansatz hinterlässt jedoch schmale weiße Linien zwischen den Zeilen und führt auch zu unterschiedlich breiten farbigen Blöcken – es sei denn, dass eine gleiche Zeilenbreite erzwungen wird, z.B. mit `\makebox`.

```
Wortwörtliche Ausgabe mit
Hintergrundfarbe.

Wortwörtliche Ausgabe mit
Hintergrundfarbe.
```

```
\usepackage{color,fancyvrb}
\renewcommand\FancyVerbFormatLine[1]
  {\colorbox{green}{#1}}
\begin{Verbatim}
Wortwörtliche Ausgabe mit
Hintergrundfarbe.
\end{Verbatim}
\renewcommand\FancyVerbFormatLine[1]
  {\colorbox{yellow}{\makebox[\linewidth][l]{#1}}}
\begin{Verbatim}
Wortwörtliche Ausgabe mit
Hintergrundfarbe.
\end{Verbatim}
```

Bsp. 3-4-13

Man kann Text als Teil eines Rahmens formatieren, indem man ihn als Wert des Schlüsselwortes `label` angibt. Wenn dieser Text Sonderzeichen enthält, wie etwa Klammern, Gleichheitszeichen oder Kommas, müssen diese maskiert werden, indem sie in ein Klammerpaar eingefasst werden. Ansonsten werden sie fälschlicherweise als Teil der Syntax interpretiert. Der Text erscheint standardmäßig am oberen Rand, wird aber nur ausgedruckt, wenn die Einstellung für den Rahmen an dieser Stelle eine Linie erzeugt. Andere Positionen können mithilfe von `labelposition` festgelegt werden, welches den

[1] Es gibt keinen Wert, um eine Linie auf der rechten Seite zu erzeugen.

Wert `none`, `topline`, `bottomline` oder `all` annehmen kann. In letzterem Fall wird der Rahmentext oberhalb und unterhalb ausgegeben. Wenn der Rahmentext ungewöhnlich lang ist, muss der Abstand zwischen Rahmen und wortwörtlicher Ausgabe möglicherweise vergrößert werden, dies geschieht mithilfe des Schlüsselwortes `framesep`. Eine vorangegangene Einstellung für den Text kann mit dem Wert `none` gelöscht werden. Soll der Rahmentext „none“ ausgegeben werden, muss dieser in Klammern eingefasst werden.

Bsp. 3-4-14

```
Gerahmte Ausgabe
Beispielcode
```

```
\usepackage{fancyvrb}
\begin{Verbatim}[frame=single,label=\fbox{Beispielcode},
                 framesep=5mm,labelposition=bottomline]
Gerahmte Ausgabe
\end{Verbatim}
```

Es ist sogar möglich, verschiedene Texte für oben und unten zu definieren. Dazu muss der Text für die obere Position in eckige Klammern eingefasst werden, wie das nächste Beispiel zeigt. Damit diese Anordnung klappt, muss man `frame` entweder auf `single` oder auf `lines` setzen.

Bsp. 3-4-15

```
Anfang des Codes
  Eine Zeile Programmcode
Ende des Codes
```

```
\usepackage{fancyvrb}
\begin{Verbatim}[frame=lines,framesep=5mm,
        label={[Anfang des Codes]Ende des Codes}]
  Eine Zeile Programmcode
\end{Verbatim}
```

Standardmäßig kann die fertig gestellte Ausgabe von `Verbatim`-Umgebungen am Seitenende von LaTeX umbrochen werden, wenn sie nicht auf eine einzelne Seite passt. Dies gilt auch in den Fällen, in denen der Text eingerahmt ist. Um einen Seitenumbruch zu verhindern, muss das boolesche Schlüsselwort `samepage` auf `true` gesetzt werden.

Der vertikale Abstand zwischen Zeilen in einer `Verbatim`-Umgebung entspricht dem von normalem Text. Bei Bedarf kann er mithilfe des Schlüsselwortes `baselinestretch` vergrößert werden. Zeilen zu stauchen, so dass sie überlappen, ist nicht möglich. Den Standardzeilenvorschub erhält man wieder, indem man als Wert die Zeichenfolge `auto` angibt.

Bsp. 3-4-16

```
Dieser Text zeigt etwa doppelten
Zeilenvorschub. Siehe auch die
Besprechung des setspace-Paketes an
anderer Stelle.
```

```
\usepackage{fancyvrb}
\begin{Verbatim}[baselinestretch=1.6]
Dieser Text zeigt etwa doppelten
Zeilenvorschub. Siehe auch die
Besprechung des setspace-Paketes an
anderer Stelle.
\end{Verbatim}
```

Bei der Darstellung von Computerlistings ist es häufig hilfreich, einige oder alle Zeilen zu nummerieren. Hier hilft das Schlüsselwort `numbers`, das den Wert `none`, `left` oder `right` annehmen kann, wodurch die Position der Nummern festgelegt wird. Der Abstand zwischen der Nummer und der wortwörtlichen Ausgabe beträgt standardmäßig `12pt`, er kann aber über das

Schlüsselwort `numbersep` geändert werden. In der Regel beginnt die Nummerierung mit jeder neuen Umgebung wieder bei 1. Man kann jedoch beim Schlüsselwort `firstnumber` explizit einen beliebigen ganzzahligen Startwert angeben, der sogar negativ sein darf. Alternativ kann in diesem Schlüsselwort `last` angegeben werden, welches festlegt, dass die Nummerierung dort fortgesetzt werden soll, wo sie in der vorhergehenden `Verbatim`-Umgebung aufgehört hat.

```
\usepackage{fancyvrb}
\begin{Verbatim}[numbers=left,numbersep=6pt]
Zeilen können entweder links
oder rechts nummeriert werden.
\end{Verbatim}
Text dazwischen\ldots
\begin{Verbatim}[numbers=left,
                 firstnumber=last]
Wie hier zu sehen, kann die
Nummerierung fortgesetzt werden.
\end{Verbatim}
```

```
1 Zeilen können entweder links
2 oder rechts nummeriert werden.
```

Text dazwischen…

```
3 Wie hier zu sehen, kann die
4 Nummerierung fortgesetzt werden.
```

Bsp. 3-4-17

Manche ziehen es vor, nur einige Zeilen zu nummerieren. Auf diese Wünsche geht das Paket mit dem Schlüsselwort `stepnumber` ein. Wenn diesem Schlüsselwort eine positive ganze Zahl zugewiesen wird, werden nur jene Zeilennummern ausgedruckt, die ein Vielfaches dieser Zahl sind. Wie bereits erwähnt, heißt der Zähler, der intern zur Zählung von Zeilen verwendet wird, `FancyVerbLine`. Es überrascht daher wenig, dass das Erscheinungsbild der Nummern durch den Befehl `\theFancyVerbLine` gesteuert wird. Durch Veränderung dieses Befehls können Spezialeffekte erzielt werden. Eine Möglichkeit, bei der die aktuelle Kapitelnummer vorangestellt wird, zeigt das nächste Beispiel. Es demonstriert auch die Verwendung des booleschen Schlüsselwortes `numberblanklines`, das festlegt, ob Leerzeilen nummeriert werden (voreingestellt ist `false`, d.h. sie werden nicht nummeriert).

```
\usepackage{fancyvrb}
\renewcommand\theFancyVerbLine
   {\footnotesize
    \thechapter.\arabic{FancyVerbLine}}
\begin{Verbatim}[numbers=left,stepnumber=2,
                 numberblanklines=true]
Normalerweise haben Leerzeilen
in einer Verbatim-Umgebung keine
Nummern -- hier haben sie welche!

Zugegeben, die mit FancyVerbLine
hinzugefügten Gliederungsnummern
sehen etwas seltsam aus, wenn man
"‘stepnumber"’ verwendet.
\end{Verbatim}
```

```
Normalerweise haben Leerzeilen
3.2 in einer Verbatim-Umgebung keine
    Nummern -- hier haben sie welche!
3.4

    Zugegeben, die mit FancyVerbLine
3.6 hinzugefügten Gliederungsnummern
    sehen etwas seltsam aus, wenn man
3.8 "‘stepnumber"’ verwendet.
```

Bsp. 3-4-18

In manchen Situationen ist es hilfreich, Leerräume zu kennzeichnen, indem alle Leerzeichen als ␣ dargestellt werden. Dies lässt sich mit dem boole-

schen Schlüsselwort showspaces oder alternativ mit der Verbatim*-Variante der Umgebung erreichen.

Ein weiteres Zeichen zur Darstellung von Leerräumen, nämlich der Tabulator, spielt in einigen Programmiersprachen eine wichtige Rolle, so dass es manchmal notwendig sein kann, diesen zu kennzeichnen. Hierzu steht das boolesche Schlüsselwort showtabs zur Verfügung. Das dargestellte Tabzeichen wird durch den Befehl \FancyVerbTab definiert und kann, wie unten gezeigt, neu definiert werden. Standardmäßig entspricht ein Tabstopp acht Leerzeichen. Dieser Wert kann mit dem Schlüsselwort tabsize verändert werden. Weist man dem booleschen Schlüsselwort obeytabs den Wert true zu, dann erzeugt der Tabstopp jedoch genau die benötigte Anzahl an Leerzeichen, um auf das nächste ganzzahlige Vielfache von tabsize zu gelangen. Im nächsten Beispiel enthalten alle Zeilen Tabstopps, die in der Eingabe als Leerzeichen mit einem voreingestellten tabsize-Wert von 8 dargestellt werden. Man beachte vor allem in der letzten Zeile den Unterschied zwischen der Ein- und Ausgabe.

```
\usepackage{fancyvrb}
\begin{Verbatim}[showtabs=true]
12345678901234567890123456789O
Zwei    Standard        Tabulatoren
\end{Verbatim}
\begin{Verbatim}[obeytabs=true,showtabs=true]
Zwei    echte   Tabulatoren
\end{Verbatim}
\renewcommand\FancyVerbTab{$\triangleright$}
\begin{Verbatim}[obeytabs=true,showtabs=true]
Zwei    neue    Tabulatoren
\end{Verbatim}
\begin{Verbatim}[obeytabs=true,tabsize=3,showtabs=true]
Und     eine    spezielle Tab   einstellung
\end{Verbatim}
```

```
12345678901234567890123456789O
Zwei       ⇥Standard       ⇥Tabulatoren

Zwei   ⇥echte  ⇥Tabulatoren

Zwei   ▷neue   ▷Tabulatoren

Und  ▷eine ▷spezielle Tab ▷einstellung
```

Bsp. 3-4-19

Wenn innerhalb der wortwörtlichen Ausgabe Befehle ausgeführt werden sollen, müssen diese durch ein spezielles Zeichen markiert werden (um den Beginn eines Befehls zu kennzeichnen). Zwei weitere Zeichen werden als Begrenzungssymbole für Argumente benötigt (d.h. sie übernehmen die Funktion, die in LaTeX normalerweise geschweifte Klammern haben). Solche speziellen Zeichen können mit dem Schlüsselwort commandchars definiert werden (siehe unten); allerdings dürfen diese Zeichen dann natürlich nicht in der

wortwörtlichen Ausgabe vorkommen. Die Zeichen werden durch Voranstellen eines Backslashs markiert. Damit wird jegliche Bedeutung, die sie in LaTeX möglicherweise haben, maskiert. Mit dem Schlüsselwort `commentchar` kann ein Kommentarzeichen definiert werden, was dazu führt, dass alles bis zum nächsten Zeilenumbruch ignoriert wird. Wenn dieses Zeichen beispielsweise in der Mitte einer Zeile eingefügt wird, werden diese und die nächste Zeile zusammengefasst. Die Einstellung für `commandchars` oder `commentchar` kann mithilfe des Schlüsselwortes „`none`“ aufgehoben werden.

```
\usepackage{fancyvrb}
\begin{Verbatim}[commandchars=\|\[\],commentchar=\!]
Text kann |emph[hervorgehoben] werden.
! siehe oben (diese Zeile ist unsichtbar)
Eine Zeile mit Label|label[linea] ! entfernt Zeilenvorschub
ist hier zu sehen.
\end{Verbatim}
In Zeile~\ref{linea} sieht man\ldots
```

```
Text kann hervorgehoben werden.
Eine Zeile mit Label ist hier zu sehen.

In Zeile 2 sieht man...
```

Bsp. 3-4-20

Wenn man wie im vorangegangenen Beispiel innerhalb der `Verbatim`-Umgebung den Befehl `\label` verwendet, bezieht sich dieser auf die interne Zeilennummer, ungeachtet dessen, ob die Nummer angezeigt wird oder nicht. Hierzu muss das Schlüsselwort `commandchars` angegeben werden, was manchmal ungünstig sein kann, weil dadurch die gewählten Zeichen nicht mehr in der wortwörtlichen Ausgabe verwendet werden können.

Es gibt zwei weitere Schlüsselwörter, mit denen die Art und Weise verändert werden kann, wie der Inhalt einer `Verbatim`-Umgebung analysiert und manipuliert wird: `codes` und `defineactive`. Mit diesen beiden Schlüsselwörtern sind einige nette Tricks möglich, ihr Einsatz ist jedoch nicht so einfach zu erklären. Man braucht ein gutes Verständnis für die Funktionsweise von TeX. Eine gute Beschreibung liefert die Dokumentation des fancyvrb-Paketes.

Teilweise Darstellung der Eingabe

Normalerweise werden innerhalb der `Verbatim`-Umgebung alle Zeilen gesetzt. Man kann jedoch die Ausgabe der Zeilen mit einer Reihe von Schlüsselwörtern begrenzen. `firstline` und `lastline` definieren die erste und, sofern notwendig, letzte darzustellende Zeile. Alternativ können auch bestimmte Zeichenketten innerhalb der Umgebung als Anfang und Ende definiert werden, was dazu führt, dass alle Zeilen zwischen diesen (aber diesmal *exklusive* der angegebenen Zeilen) ausgegeben werden. Die Zeichenketten werden in den Makros `\FancyVerbStartString` und `\FancyVerbStopString` angegeben. Hier muss man jedoch aufpassen: Die Makros müssen mit den Befehlen `\newcommand*` bzw. `\renewcommand*` (re)definiert werden. Die Verwendung

von \newcommand funktioniert *nicht*! Solch eine Deklaration aufzuheben, ist sogar noch komplizierter, sie muss mittels \let rückgängig gemacht werden:

```
\let\FancyVerbStartString\relax
```

oder aber lokal begrenzt sein, so dass LaTeX die Definition am Ende der Klammerngruppe automatisch rückgängig macht – alles andere funktioniert nicht.

Bsp. 3-4-21

```
\usepackage{fancyvrb}
\newcommand*\FancyVerbStartString{START}
\newcommand*\FancyVerbStopString{STOP}
\begin{Verbatim}
  Eine nicht dargestellte wortwörtliche Ausgabe.
START
  Nur diese Zeile wird angezeigt.
STOP
  Aber der Rest wird ausgelassen.
\end{Verbatim}
```

```
Nur diese Zeile wird angezeigt.
```

Man mag sich fragen, warum es solch eine Funktion überhaupt gibt, denn man könnte die Zeilen, die nicht gesetzt werden, doch einfach weglassen. In einer Umgebung wie Verbatim sind die Einsatzmöglichkeiten tatsächlich begrenzt. Bei Verwendung mit anderen Funktionen des Paketes, die Daten in Dateien schreiben und sie wieder auslesen, bietet diese Funktion jedoch eine hervorragende Lösung für ansonsten unlösbare Probleme.

Wie die Beispiele im Buch erzeugt wurden

So nutzen etwa alle Beispiele in diesem Buch dieses Verfahren. Das Beispiel wird zusammen mit einer Dokumentenpräambel und weiterem Material in eine Datei geschrieben, so dass diese Datei ein durch LaTeX verarbeitbares Dokument wird. Dieses Dokument wird anschließend extern formatiert und das Ergebnis als EPS-Graphik in das Buch eingebunden. An seiner Seite wird der Beispielcode dargestellt, indem die generierte Datei noch einmal eingelesen wird, wobei jedoch nur jene Zeilen dargestellt werden, die sich zwischen den Zeichenketten \begin{document} und \end{document} befinden. Dies entspricht den Beispielzeilen, die in schwarz gesetzt sind. Der blau dargestellte Präambelteil wird auf ähnliche Weise erzeugt. Hierfür werden die Start- und Ende-Zeichenketten neu definiert, so dass nur die Zeilen zwischen den Zeichenketten \StartShownPreambleCommands und \StopShownPreambleCommands berücksichtigt werden. Bei der externen Verarbeitung dieses Beispiels werden diese beiden Befehle einfach ignoriert, d.h. sie werden von der im Beispiel verwendeten Klasse (die ansonsten der Dokumentenklasse article ähnelt) so definiert, dass sie nichts tun. Damit entspricht der Beispielcode (glücklicher oder unglücklicher Weise) immer dem angezeigten Ergebnis.[1]

Um Daten wortwörtlich in eine Datei zu schreiben, kann man die Umgebung VerbatimOut verwenden. Sie hat ein obligatorisches Argument: den Namen der Datei, in welche die Daten geschrieben werden sollen. Es gibt jedoch ein logisches Problem, wenn man versucht, solch eine Umgebung innerhalb der Definition einer neuen Umgebung zu verwenden. Sobald

[1] In der ersten Ausgabe entstanden leider eine Reihe von Fehlern dadurch, dass Code im Text angezeigt wurde, der nicht direkt verwendet wurde.

\begin{VerbatimOut} ausgeführt ist, wird das gesamte nachfolgende Material ohne weitere Verarbeitung eingelesen, so dass das Ende der neuen Umgebung (in der \end{VerbatimOut} versteckt ist) nicht erkannt wird. Als Lösung bietet das Paket fancyvrb den Befehl \VerbatimEnvironment. Wenn er innerhalb des \begin-Codes einer Umgebung ausgeführt wird, stellt er sicher, dass die korrespondierende Ende-Marke innerhalb „wortwörtlicher Ausgabe“ erkannt wird und ihr Code ausgeführt wird.

Um Daten wortwörtlich aus einer Datei auszulesen, kann man den Befehl \VerbatimInput verwenden. Er hat ein optionales Argument, das dem der Umgebung Verbatim ähnelt (d.h. er akzeptiert alle zuvor besprochenen Schlüsselwörter) sowie ein obligatorisches Argument, das angibt, aus welcher Datei die Daten ausgelesen werden sollen. Die Variante \BVerbatimInput gibt den darzustellenden Text in einer Box ohne Leerraum oben und unten aus. Das nächste Beispiel zeigt einige der Möglichkeiten: Es definiert die Umgebung example, die zunächst ihren Inhalt in eine Datei schreibt, die erste Zeile wieder liest und diese blau darstellt, die Datei ein zweites Mal liest, wobei sie dieses Mal in der zweiten Zeile startet, und die Zeilen beginnend bei 1 durchnummeriert. Wie oben erläutert, wird zum Erstellen der Beispiele in diesem Buch eine ähnliche, wenn auch kompliziertere Definition verwendet.

```
\usepackage{fancyvrb,color}
\newenvironment{example}
  {\VerbatimEnvironment\begin{VerbatimOut}{test.out}}
  {\end{VerbatimOut}\noindent
   \BVerbatimInput[lastline=1,%
                   formatcom=\color{blue}]{test.out}%
   \VerbatimInput[numbers=left,%
                  firstnumber=1,firstline=2]{test.out}}
\begin{example}
Eine blaue Zeile.
Zwei Zeilen
mit Nummern.
\end{example}
```

```
Eine blaue Zeile.

1 Zwei Zeilen
2 mit Nummern.
```

Bsp. 3-4-22

Eine Anzahl interessanter Beispielumgebungen enthält das Paket fvrb-ex von Denis Girou, das auf den Funktionen von fancyvrb aufsetzt.

Umgebungs- und Befehlsvarianten

Bisher haben alle Beispiele die Verbatim-Umgebung verwendet. Es gibt jedoch noch eine Reihe von Varianten, die in bestimmten Situationen nützlich sind. BVerbatim ähnelt Verbatim, setzt die wortwörtliche Ausgabe aber in eine Box. Einige der oben besprochenen Schlüsselwörter (vorzugsweise jene, die sich mit Rahmen befassen) werden nicht unterstützt. Dafür gibt es zwei zusätzliche Schlüsselwörter. Das erste, baseline, gibt den Justierungspunkt für die Box an. Es kann die Werte t (für top, zu deutsch: oben), c (für center, zu deutsch: zentriert) und b (für bottom, zu deutsch: unten) annehmen. Letzterer ist der voreingestellte Wert. Das zweite Schlüsselwort boxwidth gibt die gewünschte Breite der Box an. Wenn es fehlt oder der angegebene Wert auto ist, wird die Box so breit wie die längste Zeile in der Umgebung. Auch

das schon angesprochene \BVerbatimInput unterstützt diese zusätzlichen Schlüsselwörter.

Bsp. 3-4-23

```
\usepackage{fancyvrb}
\begin{BVerbatim}[boxwidth=4pc,baseline=t]
erste Zeile
zweite Zeile
\end{BVerbatim}
\begin{BVerbatim}[baseline=c]
erste Zeile
zweite Zeile
\end{BVerbatim}
```

```
erste Zeile
zweite Zeile

erste Zeile
zweite Zeile
```

Alle Umgebungen und Befehle für den Satz von wortwörtlicher Ausgabe gibt es auch in Sternform, die, wie die Standardumgebungen von LaTeX, Leerzeichen als ␣ darstellen. Mit anderen Worten: Sie setzen intern das Schlüsselwort showspaces auf true.

Definition eigener Varianten

Benutzerdefinierte Varianten von Verbatim-Befehlen und -Umgebungen zu definieren, ist recht einfach. Zunächst können die Standardeinstellungen des Paketes mithilfe des Befehls \fvset verändert werden. Er hat ein Argument: eine durch Kommas getrennte Liste von Schlüssel-Wert-Paaren. Diese werden auf alle Verbatim-Umgebungen oder -Befehle angewendet. Natürlich können diese neuen Standardwerte immer noch mit dem optionalen Argument des Befehls oder der Umgebung überschrieben werden. Wenn z.B. alle Verbatim-Umgebungen um zwei Leerzeichen eingerückt sind, möchte man diese vielleicht global entfernen, anstatt bei jeder Instanz gobble zu verwenden.

Bsp. 3-4-24

```
\usepackage{fancyvrb}   \fvset{gobble=2}
\noindent Eine Textzeile, die den linken Rand zeigt.
\begin{Verbatim}
  Der neue 'Normal'-Fall.
\end{Verbatim}
\begin{Verbatim}[gobble=0]
Jetzt muss gobble bei Bedarf
explizit deaktiviert werden!
\end{Verbatim}
```

Eine Textzeile, die den linken Rand zeigt.

```
Der neue 'Normal'-Fall.

Jetzt muss gobble bei Bedarf
explizit deaktiviert werden!
```

Möglicherweise ist es jedoch gar nicht erwünscht, dass \fvset auf alle Umgebungen und Befehle angewendet wird. Daher bietet das Paket vier Deklarationen, mit denen eigene Verbatim-Befehle und -Umgebungen definiert werden können bzw. das Verhalten von vordefinierten verändert werden kann.

```
\CustomVerbatimEnvironment      {neu-umg}{basis-umg}{schlüssel-wert-liste}
\RecustomVerbatimEnvironment{ändere-umg}{basis-umg}{schlüssel-wert-liste}
\CustomVerbatimCommand          {neu-befehl}{basis-befehl}{schlüssel-wert-liste}
\RecustomVerbatimCommand    {ändere-befehl}{basis-befehl}{schlüssel-wert-liste}
```

Diese Deklarationen haben jeweils drei Argumente: den Namen der zu definierenden beziehungsweise zu ändernden Umgebung, den Namen der Umge-

bung, auf der diese basieren soll, und eine durch Kommas getrennte Liste von Schlüssel-Wert-Paaren, die das neue Verhalten definiert. Die Deklarationen für Befehle sind analog aufgebaut, man achte aber darauf, dass *basis-befehl* den Befehlsnamen ohne vorangestellten Backslash erwartet.

Zur Definition neuer Strukturen kann `\CustomVerbatimEnvironment` beziehungsweise `\CustomVerbatimCommand` verwendet werden. Um das Verhalten bestehender Befehle und Umgebungen (vordefinierter als auch selbstdefinierter) zu ändern, stehen `\RecustomVerbatimEnvironment` bzw. `\RecustomVerbatimCommand` zur Verfügung. Wie im folgenden Beispiel zu sehen, können die Standardwerte, die im dritten Argument gesetzt sind, wie üblich mit dem optionalen Argument überschrieben werden, wenn die Umgebung oder der Befehl instantiiert wird.

```
\usepackage{fancyvrb}
\CustomVerbatimEnvironment{Xverbatim}{Verbatim}
        {numbers=left,frame=lines,framerule=2pt}
\begin{Xverbatim}
Der Normalfall mit dicken
Linien und Nummern links.
\end{Xverbatim}
\begin{Xverbatim}[numbers=none,framerule=.6pt]
Die Ausnahme ohne Nummern
und dünneren Linien.
\end{Xverbatim}
\RecustomVerbatimEnvironment{Xverbatim}{Verbatim}
        {numbers=left,frame=none,showspaces=true}
\begin{Xverbatim}
Und ab hier verhält sich
die Umgebung wieder anders.
\end{Xverbatim}
```

```
1 Der Normalfall mit dicken
2 Linien und Nummern links.
```

```
Die Ausnahme ohne Nummern
und dünneren Linien.
```

```
1 Und␣ab␣hier␣verhält␣sich
2 die␣Umgebung␣wieder␣anders.
```

Bsp. 3-4-25

Verschiedene Funktionen

LaTeXs Standardbefehl `\verb` kann normalerweise nicht in Argumenten verwendet werden, weil dort die Analyse fehlschlägt und zu falschen Ergebnissen oder Fehlermeldungen führt. Eine Lösung dieses Problems besteht darin, die Daten für die wortwörtliche Ausgabe außerhalb des Argumentes zu verarbeiten, diese abzuspeichern und die analysierten Daten später an solchen gefährlichen Stellen zu verwenden. Zu diesem Zweck bietet das Paket fancyvrb die Befehle `\SaveVerb` und `\UseVerb`.

`\SaveVerb`[*schlüssel-wert-liste*]{*label*}=*daten*=
`\UseVerb*`[*schlüssel-wert-liste*]{*label*}

Der Befehl `\SaveVerb` hat ein obligatorisches Argument, ein *label*, das den Speicherbehälter angibt, in dem die analysierten Daten gesichert werden sollen. Dieses wird gefolgt von den *daten* der wortwörtlichen Ausgabe, die in zwei identische Zeichen eingeschlossen werden (in der Beispielsyntax oben =), auf dieselbe Weise, in der `\verb` sein Argument begrenzt. Um diese Daten zu verwenden, muss `\UseVerb` mit dem obligatorischen Argument *label* aufgerufen werden. Da die Daten von `\SaveVerb` zwar analysiert, aber nicht gesetzt

wurden, kann die Formatierung mithilfe einer Liste von Schlüssel-Wert-Paaren oder, wie bei anderen Verbatim-Befehlen und -Umgebungen, einer Sternform geändert werden. Natürlich ist nur ein Teil der Schlüsselwörter sinnvoll, die irrelevanten werden stillschweigend ignoriert. Der Befehl `\UseVerb` ist (unnötigerweise) zerbrechlich, daher muss er mit `\protect` in bewegten Argumenten geschützt werden.

Bsp. 3-4-26

```
\usepackage{fancyvrb}
\SaveVerb{danger}=Reale \Gefahr=

\tableofcontents

\section{\protect\UseVerb{danger}}

Die \UseVerb*{danger} ist nicht länger
problematisch und
kann\marginpar{\UseVerb[fontsize=\tiny]
                        {danger}}
beliebig oft wiederverwendet werden.
```

Inhaltsverzeichnis

1 `Reale \Gefahr` **6**

1 Reale \Gefahr

`Reale␣\Gefahr` Die `Reale␣\Gefahr` ist nicht länger problematisch und kann beliebig oft wiederverwendet werden.

Solche Speicherbehälter können wieder verwendet werden, wenn sie nicht mehr benötigt werden. Bei Verwendung von `\UseVerb` in Befehlen, die ihre Argumente über eine große Entfernung verteilen, muss man jedoch aufpassen, dass der Speicherbehälter tatsächlich noch den gewünschten Inhalt enthält, der schließlich gesetzt werden soll. Im vorangegangenen Beispiel wurde `\SaveVerb` in die Präambel eingefügt, weil die Verwendung des Speicherbehälters im `\section`-Befehl letzten Endes dazu führt, dass `\UseVerb` im Befehl `\tableofcontents` ausgeführt wird.

`\SaveVerb` besitzt außerdem ein optionales Argument, in dem eine Schlüssel-Wert-Liste eingegeben werden kann. Aber auch hier sind nur ein paar Schlüssel relevant (z.B. jene, die sich mit der Analyse befassen). Es gibt ein zusätzliches Schlüsselwort namens `aftersave`, das den Code enthält, der unmittelbar nach dem Abspeichern der wortwörtlichen Ausgabe in den Speicherbehälter ausgeführt wird. Das nächste Beispiel zeigt eine Anwendung dieses Schlüsselwortes: die Definition einer speziellen Variante des Befehls `\item`, in welchem eine wortwörtliche Ausgabe angegeben werden kann, die in einer `description`-Umgebung dargestellt wird. Außerdem unterstützt er ein optionales Argument, das aus einer Schlüssel-Wert-Liste besteht, welche die Formatierung steuert. Es lohnt sich, diese Definition genauer zu betrachten, auch wenn die Unmenge an Klammern zunächst kompliziert erscheint. Sie sind notwendig, um sicherzustellen, dass die richtigen eckigen Klammern von `\SaveVerb`, `\item` und `\UseVerb` als Argumentbegrenzer erkannt werden – das übliche Problem, dass sich eckige Klammern in TEX nicht wie geschweifte verschachteln lassen.[1] Man beachte auch die Verwendung von `\textnormal`, welcher notwendig ist, um `\bfseries` zu deaktivieren, der implizit von `\item` verwendet wird. Ansonsten hätte der Befehl `\emph` keine

[1] Der Autor gesteht, dass er drei Anläufe brauchte (bis kurz vor Mitternacht), bevor dieses Beispiel funktionierte.

Wirkung, da es keinen fettkursiven Computer Modern Schriftschnitt gibt.

\ddanger Gefährliches Biest; zu finden im TEXbook.

\danger Sein kleiner Bruder, immer noch gefährlich.

\dddanger{*arg*} Der ultimative Horror.

```
\usepackage{fancyvrb}
\newcommand\vitem[1][]{\SaveVerb[commandchars=\|\<\>,%
   aftersave={\item[\textnormal{\UseVerb[#1]{vsave}}]}]{vsave}}
\begin{description}
\vitem+\ddanger+   Gefährliches Biest; zu finden im \TeX{}book.
\vitem[fontsize=\tiny]+\danger+ Sein kleiner Bruder,
                                immer noch gefährlich.
\vitem+\dddanger{|emph<arg>}+   Der ultimative Horror.
\end{description}
```

Bsp. 3-4-27

Auf dieselbe Weise können komplette Verbatim-Umgebungen gespeichert werden, indem man die Umgebung `SaveVerbatim` verwendet. Sie hat ein obligatorisches Argument, in dem der Name des Speicherbehälters angegeben wird. Um die Verbatim-Umgebung zu setzen, kann man `\UseVerbatim` oder `\BUseVerbatim` (für die Ausgabe in Boxen) mit den üblichen Schlüssel-Wert-Paaren verwenden.

In Fußnoten sind Verbatim-Befehle und -umgebungen normalerweise nicht zugelassen, trotzdem benötigt man kein `\SaveVerb` oder Ähnliches, um an solchen Stellen wortwörtliche Ausgabe zu erzeugen. Stattdessen wird der Befehl `\VerbatimFootnotes` am Anfang des Dokumentes (nach der Präambel) eingefügt, und von da ab können Verbatim-Befehle in Fußnoten verwendet werden. Dies gibt es jedoch nur für Fußnoten – für andere Befehle, wie `\section`, wird immer noch das oben beschriebene, kompliziertere Verfahren mit Speicherbehälter benötigt.

Ein bisschen Text als Anlass[1] für eine Fußnote. War dies gut genug?

[1] Hier ist der Beweis: `\danger{%_^}`

```
\usepackage{fancyvrb}
\VerbatimFootnotes
Ein bisschen Text als Anlass\footnote{Hier ist
der Beweis: \verb=\danger{%_^}=} für eine Fußnote.
War dies gut genug?
```

Bsp. 3-4-28

Die fancyvrb-Version von `\verb` heißt `\Verb`. Sie unterstützt alle anwendbaren Schlüsselwörter, die ihr wie üblich mittels eines optionalen Argumentes übergeben werden können. Das Beispiel erzeugt `\verbx` als Variante von `\Verb` mit einer speziellen Einstellung von `commandchars`, so dass Befehle innerhalb seines Argumentes ausgeführt werden können. Zu diesem Zweck muss `\CustomVerbatimCommand` verwendet werden, da `\verbx` ein neuer Befehl ist, der in Standard-LaTeX nicht verfügbar ist.

`\dddanger{|emph<arg>}`
`\dddanger{`*arg*`}`

```
\usepackage{fancyvrb}
\CustomVerbatimCommand\verbx{Verb}{commandchars=\|\<\>}
\Verb [fontfamily=courier]+\dddanger{|emph<arg>}+ \par
\verbx[fontfamily=courier]+\dddanger{|emph<arg>}+
```

Bsp. 3-4-29

Wie bereits erwähnt, bietet fancyvrb die Möglichkeit, den Anfang und das Ende einer wortwörtlichen Ausgabe mit einem bestimmten Zeichen zu kennzeichnen, ohne ihr den Befehl `\verb` voranzustellen. Der Befehl zur Deklaration dieser Begrenzungssymbole heißt `\DefineShortVerb`. Wie ande-

re fancyvrb-Befehle akzeptiert er ein optionales Argument, in dem Schlüssel-Wert-Paare spezifiziert werden können. Sie wirken sich auf die Formatierung und die Analyse aus, wobei dieses Mal die gewählten Einstellungen in den einzelnen Instanzen nicht überschrieben werden können. Alternativ kann `\fvset` verwendet werden, da dieser Befehl für alle Verbatim-Befehle und -Umgebungen in seinem Gültigkeitsbereich funktioniert. Um die spezielle Bedeutung eines Zeichens aufzuheben, das mit `\DefineShortVerb` deklariert wurde, ist `\UndefineShortVerb` zu verwenden.

Bsp. 3-4-30

Die Verwendung von `\DefineShortVerb` kann Quellen viel lesbarer machen!

Und mit `\UndefineShortVerb{\|}` erhält das Zeichen wieder seine normale Bedeutung.

```
\usepackage{fancyvrb}
\DefineShortVerb[fontsize=\tiny]{\|}
Die Verwendung von |\DefineShortVerb|
kann Quellen viel lesbarer machen! \par
\UndefineShortVerb{\|}\DefineShortVerb{\+}
\fvset{fontfamily=courier}
Und mit  +\UndefineShortVerb{\|}+ erhält
das Zeichen wieder seine normale Bedeutung.
```

Es gibt die Möglichkeit, die beliebtesten Erweiterungen oder Anpassungen in einer Datei namens `fancyvrb.cfg` zusammenzufassen. Sobald fancyvrb geladen ist, sucht das Paket automatisch nach dieser Datei. Für die Verwendung einer solchen Datei spricht, dass die Erweiterungen nicht mehr in alle Dokumente eingefügt werden müssen, wenn die Datei an zentraler Stelle platziert wird. Der Nachteil ist, dass die Dokumente nicht mehr portierbar sind, es sei denn, man legt die Datei bei der Verteilung der Dokumente stets bei.

3.4.4 listings – Schön gesetzter Programmcode

Eine wichtige Anwendung der wortwörtlichen Ausgabe ist die Darstellung von Programmcode. Man kann zwar einfach und erfolgreich ein Paket wie fancyvrb einsetzen, um diese Aufgabe zu bewältigen, es ist jedoch häufig vorteilhaft, die Darstellung durch besondere Formatierung bestimmter Programmelemente (wie Schlüsselwörter, Bezeichner und Kommentare) zu verbessern.

Hierfür gibt es zwei Hauptansätze: Entweder kennzeichnet man mit Befehlen die logischen Aspekte des Algorithmus oder der Programmiersprache, oder die Anwendung analysiert den Programmcode hinter den Kulissen. Der Vorteil des ersten Verfahrens besteht darin, dass man im Prinzip eine bessere Kontrolle über die Darstellung hat. Auf der anderen Seite ist der Programmcode mit TEX-Befehlen durchsetzt und damit schwer zu pflegen, für die direkte Verarbeitung nicht zu gebrauchen und häufig im Quelltext schlecht lesbar. Pakete, die in diese Kategorie fallen, sind beispielsweise alg und algorithmic. Hier ein Beispiel:

Bsp. 3-4-31

if $i \leq 0$ **then**
  $i \leftarrow 1$
else
  if $i \geq 0$ **then**
    $i \leftarrow 0$
  end if
end if

```
\usepackage{algorithmic}
\begin{algorithmic}
\IF {$i\leq0$} \STATE $i\gets1$ \ELSE
\IF {$i\geq0$} \STATE $i\gets0$ \ENDIF
\ENDIF
\end{algorithmic}
```

ABAP (R/2 4.3, R/2 5.0, R/3 3.1, R/3 4.6C, R/3 6.10)
ACSL
Ada (83, 95)
Algol (60, 68)
Assembler (x86masm)
Awk (gnu, POSIX)
Basic (Visual)
C (ANSI, Objective, Sharp)
C++ (ANSI, GNU, ISO, Visual)
Caml (light, Objective)
Clean
Cobol (1974, 1985, ibm)
Comal 80
csh
Delphi
Eiffel
Elan
erlang
Euphoria
Fortran (77, 90, 95)
GCL
Gnuplot
Haskell
HTML
IDL (*leer*, CORBA)
Java (*leer*, AspectJ)
ksh
Lisp (*leer*, Auto)
Logo
Make (*leer*, gnu)
Mathematica (1.0, 3.0)
Matlab
Mercury
MetaPost
Miranda
Mizar
ML
Modula-2
MuPAD
NASTRAN
Oberon-2
OCL (decorative, OMG)
Octave
Pascal (Borland6, Standard, XSC)
Perl
PHP
PL/I
POV
Prolog
Python
R
Reduce
S (*leer*, PLUS)
SAS
Scilab
SHELXL
Simula (67, CII, DEC, IBM)
SQL
tcl (*leer*, tk)
TeX (AlLaTeX, common, LaTeX, plain, primitive)
VBScript
Verilog
VHDL (*leer*, AMS)
VRML (97)
XML

blau gibt den Standarddialekt an

Tabelle 3.7: Von listings unterstützte Sprachen (Winter 2003)

Der zweite Ansatz wird exemplarisch am Paket listings[1] von Carsten Heinz veranschaulicht. Dieses Paket analysiert zunächst den Code, zerlegt ihn in seine Bestandteile und formatiert diese dann nach selbst definierbaren Regeln. Der Parser ist recht allgemein und kann so angepasst werden, dass er die Syntax vieler verschiedenen Sprachen erkennt (siehe Tabelle 3.7). In regelmäßigen Abständen kommen neue Sprachen hinzu. Wenn die gewünschte Zielsprache nicht aufgeführt ist, empfiehlt es sich, auf CTAN die neueste Version des Paketes zu überprüfen. Alternativ kann man die erforderlichen Deklaration auch selbst bereitstellen. Das kostet etwas Arbeit, ist aber nicht sehr schwierig.

Die Anwenderbefehle und -Umgebungen in diesem Paket haben viele Gemeinsamkeiten mit jenen in fancyvrb. Die Aspekte zur syntaktischen Analyse und zur Formatierung werden über Schlüssel-Wert-Paare gesteuert, die in einem optionalen Argument angegeben werden und die Einstellungen für das gesamte Dokument oder größerer Teile davon können über `\lstset` definiert werden (der entsprechende fancyvrb-Befehl heißt `\fvset`). Wann immer möglich, verwenden beide Pakete dieselben Schlüsselwörter, so dass Benutzer des einen Paketes recht leicht auch das andere beherrschen lernen.

Nach dem Laden des Paketes empfiehlt es sich, mit `\lstloadlanguages` alle im Dokument benötigten Programmiersprachen anzugeben (als durch

[1] Die hier beschriebene Paketversion ist 1.0. Frühere Versionen verwendeten eine etwas andere Syntax. Falls Funktionen nicht so arbeiten wie angegeben, sollte man das Paket aktualisieren.

Kommas getrennte Liste). Solch eine Deklaration wählt keine Sprache aus, sondern lädt nur die notwendige unterstützende Information und beschleunigt die Verarbeitung.

Programmteile werden in einer `lstlisting`-Umgebung eingebettet. Die Sprache des Programmteils wird mit dem Schlüsselwort `language` angegeben. Im folgenden Beispiel wird das Schlüsselwort per `\lstset` auf `C` gesetzt und später im optionalen Argument der zweiten `lstlisting`-Umgebung überschrieben.

Eine „for“-Schleife in C:

```
int sum;
int i; /* Schleifenvariable */
sum=0;
for (i=0;i<n;i++) {
  sum += a[i];
}
```

Nun dieselbe Schleife in Ada:

```
Sum: Integer;
-- keine Deklaration für I nötig
Sum := 0;
for I in 1..N loop
  Sum := Sum + A(I);
end loop;
```

Bsp. 3-4-32

```
\usepackage{listings}
\lstloadlanguages{C,Ada}
\lstset{language=C,commentstyle=\scriptsize
        ,extendedchars=true}
Eine "‘for"’-Schleife in C:
\begin{lstlisting}[keywordstyle=\underbar]
int sum;
int i; /* Schleifenvariable */
sum=0;
for (i=0;i<n;i++) {
  sum += a[i];
}
\end{lstlisting}
Nun dieselbe Schleife in Ada:
\begin{lstlisting}[language=Ada]
Sum: Integer;
-- keine Deklaration für I nötig
Sum := 0;
for I in 1..N loop
  Sum := Sum + A(I);
end loop;
\end{lstlisting}
```

Das Schlüsselwort `extendedchars` wird benötigt, um die Umlaute in den Kommentaren korrekt darzustellen; weitere Informationen dazu finden sich auf Seite 184. Obiges Beispiel verwendet auch das Schlüsselwort `commentstyle`, welches das generelle Erscheinungsbild von Kommentaren steuert. Das Paket erkennt die sprachabhängigen Syntaxstile für Kommentare automatisch. Es gibt weitere solcher Schlüsselwörter – u.a. `basicstyle`, welches das gesamte Erscheinungsbild des Programmlistings steuert, `stringstyle`, für Zeichenketten in der Programmiersprache, und `directivestyle`, das Compilerdirektiven formatiert.

Um die Schlüsselwörter einer Programmiersprache zu formatieren, werden `keywordstyle` und `ndkeywordstyle` (zweiter Ordnung) verwendet. Weitere Bezeichner werden gemäß der Einstellung von `identifierstyle` formatiert. Die Werte aller „Stil“-Schlüsselwörter (außer `basicstyle`) können als letztes Element einen LaTeX-Befehl mit einem Argument enthalten, wie etwa `\textbf`. Dies funktioniert, weil der „Bezeichnername“ intern in geschweifte Klammern gesetzt wird und deshalb als Argument eines Befehls interpretiert werden kann.

Auf diese Weise können Schlüsselwörter, Bezeichner und andere Elemente automatisch und doch ganz individuell hervorgehoben werden. Manchmal sollen zusätzlich bestimmte Variablen, Funktionen oder Schnittstellen

betont werden. Zu diesem Zweck kann man die Schlüsselwörter `emph` und `emphstyle` einsetzen. Ersteres erhält eine Liste der Namen, die hervorgehoben werden sollen, letzteres gibt an, wie diese gesetzt werden sollen.

```
\usepackage{listings,color}
\lstset{emph={Sum,N},emphstyle=\color{blue},
        emph=[2]I,emphstyle=[2]\underbar}
\begin{lstlisting}[language=Ada]
Sum: Integer;   Sum := 0;
for I in 1..N loop
  Sum := Sum + A(I);
end loop;
\end{lstlisting}
```

```
Sum: Integer;   Sum := 0;
for I in 1..N loop
  Sum := Sum + A(I);
end loop;
```

Bsp. 3-4-33

Wenn Codefragmente im normalen Text gesetzt werden sollen, kann man den Befehl `\lstinline` verwenden. Der Code wird auf dieselbe Weise begrenzt wie beim `\verb`-Befehl, was bedeutet, dass jedes Zeichen (außer der öffnenden eckigen Klammer) als Begrenzungssymbol ausgewählt werden kann, das nicht im Codefragment vorkommt. Die öffnende eckige Klammer kann nicht verwendet werden, weil der Befehl auch ein optionales Argument akzeptiert, in dem eine Liste von Schlüssel-Wert-Paaren angegeben werden kann.

```
\usepackage{listings}  \lstset{language=C}
Die \lstinline[keywordstyle=\underbar]!for!-Schleife
ist definiert als \lstinline!i=0;i<n;i++!.
```

Die for-Schleife ist definiert als i=0;i<n;i++.

Bsp. 3-4-34

Natürlich ist es auch möglich, den Inhalt kompletter Dateien zu formatieren; zu diesem Zweck gibt es den Befehl `\lstinputlisting`. Er hat ein optionales Argument, in dem Schlüssel-Wert-Paare angegeben werden können, sowie ein obligatorisches Argument, in dem die zu verarbeitende Datei angegeben wird. Im folgenden Beispiel identifiziert das Paket Schlüsselwörter einer Sprache korrekt, die Groß- und Kleinschreibung nicht unterscheidet – selbst wenn sie eine ungewöhnliche SChrEibWeiSe haben.

```
\usepackage{listings}
\begin{filecontents*}{pascal.src}
for i:=1 to maxint do
begin
  WrItE('Dies ist albern');
end.
\end{filecontents*}
\lstinputlisting[language=Pascal]{pascal.src}
```

```
for i:=1 to maxint do
begin
  WrItE('Dies␣ist␣albern');
end.
```

Bsp. 3-4-35

Leerzeichen in Zeichenketten werden standardmäßig als ␣ dargestellt. Dieses Verhalten kann durch Setzen des Schlüsselwortes `showstringspaces` auf `false` deaktiviert werden, wie das nächste Beispiel zeigt. Die gleiche Darstellung aller Leerzeichen erreicht man auch, indem man das Schlüsselwort `showspaces` auf `true` setzt. Auf ähnliche Weise können Tabzeichen sichtbar gemacht werden, und zwar mithilfe des booleschen Schlüsselwortes `showtabs`.

Auch Zeilennummerierung ist möglich; dafür werden dieselben Schlüsselwörter verwendet wie bei fancyvrb: `numbers` kann entweder den Wert `left`,

`right` oder `none` annehmen (welches die Nummerierung ein- oder ausschaltet). `numberblanklines` legt fest, ob Leerzeilen in Hinsicht auf die Nummerierung berücksichtigt werden (voreingestellt ist `false`), `numberstyle` definiert das gesamte Erscheinungsbild der Nummern, `stepnumber` gibt an, welche Zeilennummern angezeigt werden (0 bedeutet keine Nummerierung), und `numbersep` legt den Abstand zwischen Nummer und dem Zeilenanfang fest. Standardmäßig beginnt die Zeilennummerierung mit jedem `\lstinputlisting` bei 1, diese Einstellung kann jedoch mithilfe des Schlüsselwortes `firstnumber` geändert werden. Wenn `last` als spezieller Wert von `firstnumber` angegeben wird, beginnt die Nummerierung dort, wo die letzte Nummerierung aufhört.

Bsp. 3-4-36

Vorangehender Text ...

```
10 for i:=1 to maxint do
   begin
12   WrItE('Dies ist albern');
   end.
```

```
\usepackage{listings}
% pascal.src wie vorher definiert
\lstset{numberstyle=\tiny,numbers=left,
   stepnumber=2,numbersep=5pt,firstnumber=10,
   xleftmargin=12pt,showstringspaces=false}
\noindent Vorangehender Text \ldots
\lstinputlisting[language=Pascal]{pascal.src}
```

Ein globaler Einzug kann, wie im vorigen Beispiel gezeigt, mit dem Schlüsselwort `xleftmargin` definiert werden. Eine bestimmte Anzahl von Zeichen (hoffentlich nur Leerzeichen) können mit `gobble` am Anfang jeder dargestellten Zeile entfernt werden. In der Regel werden Einzüge von übergeordneten Umgebungen, wie `itemize`, berücksichtigt. Diese Funktion kann mit dem booleschen Schlüsselwort `resetmargin` ausgeschaltet werden. Die Schlüsselwörter lassen sich natürlich kombinieren. Wenn nur ein Teil der Programmcode-Zeilen formatiert werden soll, kann man mit `firstline` und `lastline` die erste und/oder letzte Zeile definieren; zum Beispiel würden bei `lastline=10` höchstens 10 Programmcode-Zeilen gesetzt.

Eine andere Möglichkeit, eine fortlaufende Nummerierung zu erhalten, besteht über das Schlüsselwort `name`. Wenn eine Umgebung mit diesem Schlüsselwort einen „Namen“ erhalten hat, wird die Nummerierung automatisch fortgesetzt, d.h. sie startet dort, wo die letzte Umgebung mit diesem Namen endete. Dies ermöglicht bei Bedarf eine unabhängige Nummerierung einzelner Codefragmente.

Bsp. 3-4-37

Sum: Integer;

Das zweite Fragment setzt die Nummerierung fort.

```
Sum := 0;                    2
for I in 1..N loop           3
  Sum := Sum + A(I);         4
end loop;                    5
```

```
\usepackage{listings} \lstset{language=Ada,numbers=right,
   numberstyle=\tiny,stepnumber=1,numbersep=5pt}
\begin{lstlisting}[name=Test]
Sum: Integer;
\end{lstlisting}
Das zweite Fragment setzt die Nummerierung fort.
\begin{lstlisting}[name=Test]
Sum := 0;
for I in 1..N loop
  Sum := Sum + A(I);
end loop;
\end{lstlisting}
```

Wenn ein Programmlisting sehr lange Zeilen enthält, passen diese möglicherweise nicht in den verfügbaren Platz. In diesem Fall erzeugt listings überlaufende Zeilen, die in den rechten Seitenrand hineinragen, genauso wie bei einer `verbatim`-Umgebung. Mit dem Schlüsselwort `breaklines` kann man jedoch Zeilen auch an Leerzeichen oder Satzzeichen umbrechen lassen. Die Folgezeilen werden um 20pt eingerückt, wobei dieser Wert mit dem Schlüsselwort `breakindent` verändert werden kann.

Bei Bedarf kann man Material vor (Schlüsselwort `prebreak`) und/oder nach (Schlüsselwort `postbreak`) dem Umbruch einfügen, das darauf hinweist, dass das Programmlisting künstlich umbrochen wurde. Im folgenden Beispiel wird diese Funktionalität verwendet, um mit kleinen Pfeilen und später mit der Zeichenkette „(Forts.)“ in kleiner Schrift zu experimentieren. Beide Schlüsselwörter sind intern als TEX-Befehle `\discretionary` realisiert, was bedeutet, dass sie nur bestimmte Eingaben akzeptieren (Zeichen, Boxen und Unterschneidungen[1]). Komplexeres Material sollte besser in einer `\mbox` eingebettet werden (die Material in `\discretionary` verbirgt), wie es im Beispiel geschehen ist. Bei Farbänderungen reicht selbst das nicht aus. Hier benötigt man ein weiteres Klammernpaar, um zu verhindern, dass die Wirkung des `\special`-Befehls zur Farbeinstellung über die Box hinausgeht (siehe Besprechung in Anhang A.2.5).

Das Beispiel zeigt eine weitere Eigenschaft der Umbruchfunktion: Leerzeichen oder Tabzeichen, die vor dem umbrochenen Material stehen, werden standardmäßig in den Folgezeilen wiederholt. Wenn dies nicht erwünscht ist, muss das Schlüsselwort `breakautoindent` auf `false` gesetzt werden (wie im zweiten Teil des Beispiels).

```
Text am linken ↘
→Seitenrand
        /*Ein ↘
        →mehrfach ↘
        →umbrochener↘
        → Kommentar↘
        →! */

        /*Ein ↘
(Forts.) mehrfach ↘
(Forts.) umbrochener ↘
(Forts.) Kommentar! */
```

```
\usepackage{color,listings}
\lstset{breaklines=true,breakindent=0pt,
        prebreak=\mbox{\tiny$\searrow$},
        postbreak=\mbox{{\color{blue}\tiny$\rightarrow$}}}
\begin{lstlisting}
Text am linken Seitenrand
        /*Ein mehrfach umbrochener Kommentar! */
\end{lstlisting}
\begin{lstlisting}[breakautoindent=false,
                   postbreak=\tiny (Forts.)\,]
        /*Ein mehrfach umbrochener Kommentar! */
\end{lstlisting}
```

Bsp. 3-4-38

Mit dem Schlüsselwort `frame` können Rahmen oder Linien um den Programmcode herum gezogen werden; es hat dieselben Werte wie in fancyvrb (z.B. `single`, `lines`). Zusätzlich kann es eine Untermenge von `trblTRBL` als Wert annehmen. Die großgeschriebenen Buchstaben stehen für doppelte Linien, die kleingeschriebenen für einfache Linien. Es gibt ein halbes Dutzend weiterer Schlüsselwörter: zur Festlegung der Linienbreite, des Abstandes vom Text, Erzeugung von abgerundeten Ecken usw. Sie sind alle kompatibel mit fancyvrb, wenn die Funktionalität dieselbe ist.

[1] Leerraum, der durch den TEX-Befehl `\kern` erzeugt wird.

Bsp. 3-4-39

```
for␣i:=1␣to␣maxint␣do
begin
␣␣WrItE('Dies␣ist␣albern')
end.
```

```
\usepackage{listings}
% pascal.src wie vorher definiert
\lstset{frame=trBL,framerule=2pt,framesep=4pt,
        rulesep=1pt,showspaces=true}
\lstinputlisting[language=Pascal]{pascal.src}
```

Einzelne Programmlistings können mithilfe des Schlüsselwortes `caption` eine Legende erhalten. Die Legenden werden standardmäßig nummeriert und durch die Zeichenkette `Listing` eingeführt, die in `\lstlistingname` gespeichert wird. Als Zähler wird `lstlisting` verwendet, dessen Erscheinungsbild mit `\thelstlisting` verändert werden kann. Die Legende wird entweder über (voreingestellt) oder unter dem Listing platziert, wobei diese Einstellung mit dem Schlüsselwort `captionpos` geändert werden kann.

Eine formatierte Liste aller Legenden erhält man, indem man den Befehl `\lstlistoflistings` an geeigneter Stelle im Dokument einfügt. Er erzeugt eine Überschrift aus den Wörtern, die in `\lstlistlistingname` gespeichert sind (voreingestellt ist `Listings`). Wenn sich der Legendentext im Dokument vom Text im Listingverzeichnis unterscheiden soll, muss ein optionales Argument verwendet werden (siehe nächstes Beispiel). Zu beachten ist, dass in diesem Fall der Wert in geschweifte Klammern einzufassen ist, um die rechte eckige Klammer zu verbergen. Wenn die Legende nicht im Listingverzeichnis erscheinen soll, muss dem Schlüsselwort `nolol` der Wert `true` zugewiesen werden. Durch Verwenden des Schlüsselwortes `label` kann ein Label definiert werden, um per `\ref` auf die Listingnummer verweisen zu können. Voraussetzung ist, dass die Nummer nicht unterdrückt wird.

Listings

1 Pascal-Listing 6

Der Pascalcode in Listing 1 zeigt...

Bsp. 3-4-40

```
for i:=1 to maxint do
begin
  WrItE('Dies␣ist␣albern');
end.
```

Listing 1: Pascal

```
\usepackage{listings}
% pascal.src wie vorher definiert
\lstset{frame=single,frameround=tftt,
        language=Pascal,captionpos=b}
\lstlistoflistings
          %
\bigskip  % normalerweise steht obiges
\noindent % im Vorspann, aber hier ...
          %
Der Pascalcode in Listing~\ref{foo} zeigt\ldots
\lstinputlisting
   [caption={[Pascal-Listing]Pascal},label=foo]
   {pascal.src}
```

Mit dem Schlüsselwort `frameround`, das im vorangegangenen Beispiel verwendet wurde, können Ecken abgerundet werden, indem für jede Ecke, beginnend bei der rechten oberen Ecke, im Uhrzeigersinn der Wert `t` für „true“ (wahr) und `f` für „false“ (falsch) eingegeben wird. Diese Funktion ist nicht für fancyvrb-Rahmen verfügbar.

Anstatt Programmlistings im Text zu formatieren, können sie mithilfe des Schlüsselwortes `float` in Gleitobjekte umgewandelt werden, typischerweise in Kombination mit dem Schlüsselwort `caption`. Der Wert von `float` ist eine Teilmenge von `htbp`, die angibt, wohin sich das Gleitobjekt bewegen

darf (ohne Angabe eines Wertes entspricht er `tbp`). Man sollte jedoch vermeiden, gleitende und nicht gleitende Listings zu vermischen, weil dadurch die Nummerierung durcheinander geraten könnte, wie im Beispiel 6-3-5 auf Seite 305 zu sehen ist.

8-Bit-Eingabe

Standardmäßig behandelt listings nur Eingabezeichen im ASCII-Bereich; bei unerwarteten 8-Bit-Eingaben können sehr seltsame Ergebnisse entstehen, wie die Buchstabendreher im folgenden Beispiel. Wenn man `extendedchars` auf `true` setzt, werden 8-Bit-Zeichen zugelassen, was dem Paket mehr Leistung abverlangt, aber (normalerweise) zu richtigen Ergebnissen führt. Bei Verwendung eines erweiterten Zeichensatzes empfiehlt es sich natürlich, das Schlüsselwort in die Deklaration `\lstset` einzufügen, anstatt es bei jeder Umgebung erneut anzugeben, wie in Beispiel 3-4-32 auf Seite 179 geschehen. Spezielle Eingabekodierungen für Programmfragmente, die sich von der Eingabekodierung für das restliche Dokument unterscheiden, kann man auch über das Schlüsselwort `inputencoding` definieren. Dieses Schlüsselwort kann nur verwendet werden, wenn das Paket inputenc geladen ist.

```
\usepackage[ansinew]{inputenc}
\usepackage{listings}
\lstset{language=C,commentstyle=\scriptsize}
\begin{lstlisting}
int i; /*für die äußere Schleife*/
\end{lstlisting}
\begin{lstlisting}[extendedchars=true]
int i; /*für die äußere Schleife*/
\end{lstlisting}
```

```
int i; /*für die äußere Schleife*/
int i; /*für die äußere Schleife*/
```

Bsp. 3-4-41

Das Paket bietet viele weitere Schlüssel, um die Art der Darstellung festzulegen. Für spezielle Formatierungstricks kann man beliebigen LaTeX-Code ausführen, man kann Tabulatoren oder Zeichen für den Papiervorschub anzeigen, bestimmte Bezeichner indexieren oder ein Interface zum hyperref-Paket aktivieren, so dass beim Klicken auf einen Bezeichner auf die vorherige Stelle gesprungen wird, an der dieser auftrat. Einige dieser Funktionen werden noch als experimentell betrachtet und müssen über ein optionales Argument angefordert werden, wenn das Paket geladen wird. Alle diese Schlüssel sind ausführlich im (etwa 50 Seiten umfassenden) Handbuch beschrieben, das zusammen mit dem Paket verteilt wird.

Als abschließendes Beispiel für die Art von Schätzen, die in diesem Handbuch zu finden sind, siehe auch das folgende Beispiel. Es zeigt die Formatierung des Programmcodes im Stil von Donald Knuths Veröffentlichungen zu „Literate Programming“.

```
\usepackage{listings}
\lstset{literate={:=}{{$\gets$}}1
 {<=}{{$\leq$}}1 {>=}{{$\geq$}}1 {<>}{{$\neq$}}1}
\begin{lstlisting}[gobble=2]
  var i:integer;
  if (i<=0) i := 1;
  if (i>=0) i := 0;
  if (i<>0) i := 0;
\end{lstlisting}
```

```
var i:integer;
if (i≤0) i ← 1;
if (i≥0) i ← 0;
if (i≠0) i ← 0;
```

Bsp. 3-4-42