Das kleine

TEXBuch

Fritz Cremer

5.7.1993

Vorwort

Warum dieses Buch?

Ich habe mich entschlossen dieses Buch zu schreiben, weil in vielen Texten zum Programm TEX das TEXbook angesprochen wird, es aber kaum ein vernünftiges Buch zu TEX in deutscher Sprache gibt. Außerdem wollte ich den Text in der DFÜ zur Verfügung stellen. Daraus ergaben sich verschiedene Probleme. Zum einen mußte ich die Urheberrechte berücksichtigen, die auf dem TEXbook liegen, zum anderen durfte der Text nicht zu lang werden. Ich habe mich daher für folgendes Vorgehen entschlossen. Ich habe das TEXbook nicht einfach linear übersetzt, das wäre auch fast nicht möglich, da dieses Buch viel zu viele Wortspiele u.a. enthält. Ich habe vielmehr das TEXbook Abschnitt für Abschnitt gelesen und in meinen eigenen Worten wiedergegeben. Dabei konnte ich viel kürzen. So enthält dieser Text keine Übungsaufgaben mehr. Ich habe die Aufgaben, soweit sie vorkamen mit in den Text eingearbeitet, so daß keine Information verloren ging.

An wen richtet sich das Buch?

Dieser Text richtet sich an die Benutzer von TEX und nicht so sehr an die Makrodesigner für TEX, wenn auch dieses Kapitel keineswegs ausgenommen wurde. Ich habe mich nur entschlossen die Teile des Buches wegzulassen, die so speziell auf das Design von Makros ausgerichtet sind, daß sie normalerweise nicht von Interesse sein können. So fehlt z.B. der Anhang 'Dirty tricks', in dem unüblicher Gebrauch von Teilen von TEX beschrieben wird.

Copyright und Ähnliches

Fehler

Dieser Text entstand im Laufe vieler Monate. Es bleibt nicht aus, daß im Verlaufe dieser Zeit bessere und weniger gute Abschnitte entstanden. Auch werden sich noch etliche Rechtschreibfehler in diesem Buch befinden. Verbesserungsvorschläge und Korrekturhinweise bitte ich mir zukommen zu lassen.

Wem dieses Buch gefällt, der empfehle und kopiere es weiter, wems nicht gefällt, der schweige stille.

Fritz Cremer, Südstr. 45, 52064 Aachen, Deutschland

Inhaltsverzeichnis

Kapitel 1

Einführung

Wenn sie einmal ein Buch durchblättern, dann werden sie feststellen, daß es verschiedene Bindestriche gibt, je nachdem, ob Wörter durch Bindestriche verbunden werden (wie bei Tren-nung-en), ob Seitenbereiche angegeben werden (Seiten: 2–13), ob Gedankenstriche — geschrieben werden sollen, oder ob in einer mathematischen Formel ein Minuszeichen auftaucht $(1 - 2)$.

Auch könnte ihnen auffallen, daß manche Buchstaben wie *ein* Zeichen aussehen, so z.B. die Kombination von 'f' und 'i', die dann als 'fi' erscheint. Die Kombination von Zeichen zu *einem* Zeichen heißt übrigens *Ligatur*. Derartige Ligaturen, werden für viele Zeichenkombinationen benutzt, wie z.B. für 'ff', 'fl' oder 'ff'.

Aber es gibt noch weitere Merkwürdigkeiten. Sie finden z.B. auf der Tastatur ihres Computers keine Taste für ein Paar doppelter Anführungszeichen. Das liegt daran, daß es es im ASCII-Zeichensatz keine derartigen Zeichen gibt. Dennoch tauchen in Büchern diese Zeichen "auf". In TEX können sie das erreichen, indem sie einfach zweimal ein einfaches Anführungszeichen, bzw. "Abführungszeichen" benutzen. Falls es auf ihrer Tastatur kein einfaches Anführungszeichen gibt, können sie übrigens auch den Befehl '`\lq`' für das linke, bzw. '`\rq`' für das rechte Anführungszeichen benutzen. Sie müssen in diesem Fall nur ein '`\`' hinter den Befehl schreiben, wenn dem '`\lq`' oder '`\rq`' ein Leerzeichen folgt. Mit:

```
\lq\lq␣Haken\rq\rq\
```

Erhalten sie:

> "Haken"

Der Trick ist natürlich, daß auch die beiden Anführungszeichen zu einem Zeichen zusammengefaßt werden, genau wie bei dem 'fi'. Da nur zwei Anführungszeichen zusammengefaßt werden, ergibt die Kombination von drei Anführungszeichen: "'. So kann man also ein einfaches Anführungszeichen innerhalb von einem doppelten schreiben.

Übrigens, folgendes (' ") erreichen sie nicht so einfach, dazu ist ein weiterer Befehl nötig, geben sie:

```
`\thinspace``
```

ein, um das gewünschte Resultat zu erhalten.

Schließlich finden sich Zeichen, die *untereinandergeschoben* scheinen. So zu sehen, wenn ein 'V' hinter einem 'A' steht. Das sieht dann so: 'VA' aus. Dieses Untereinanderschieben nennt man *Kerning*.

Jetzt die gute Nachricht: Mit TEX brauchen sie sich um derartige Kleinigkeiten nicht mehr zu kümmern; die verschiedenen Bindestriche, die Ligaturen und auch das Kerning wird von TEX völlig selbständig erledigt. Sie können sich also ganz auf ihre eigentliche Arbeit, das Schreiben des Textes konzentrieren.

Kapitel 2

Bis zum ersten Text ...

2.1 TEX kontrollieren

Die Tastatur ihres Computers bietet viel weniger Möglichkeiten, als sie für ein Schriftstück brauchen. Daher muß ein Zeichen als sogenanntes *Escape*zeichen definiert werden, mit dem sie dann Befehle, die die Form oder das Aussehen des Textes verändern, oder die nur ein weiteres Zeichen zur Verfügung stellen. In TEX ist es zwar prinzipiell möglich jedes Zeichen dazu zu verwenden, üblicherweise wird aber der Gegenschrägstrich, '`\`', dazu verwendet. Der Grund hierfür ist einfach, dieses Zeichen kommt in normalen Texten sogut wie nie vor,

TEX versteht jede Eingabe, die mit diesem Gegenschrägstrich beginnt als eine *Kontrollsequenz*. Die Eingabe von:

```
\input␣neu
```

z.B. bedeutet, daß '`\input`' eine Kontrollsequenz ist, und '`neu`' sein Argument (Tatsächlich veranlaßt dieser Befehl TEX als nächstes den File '`neu.tex`' einzulesen und diesen zu bearbeiten).

In der Zeile:

```
Neue␣Eintr\\"age
```

findet sich die Kontrollsequenz '`\\"`'. Diese sorgt dafür, daß über das folgende Zeichen zwei Strichelchen geschrieben werden, das Ergebnis sieht also so 'ä' aus.

Im folgenden wird zwischen zwei Arten von Kontrollsequenzen unterschieden. Zum einen die, bei denen hinter dem Gegenschrägstrich eine Reihe von Buchstaben folgt (Buchstaben sind: '`A...Z`' und '`a...z`', aber nicht '`0...9`'). Die obige Sequenz '`\input`' gehört z.B. zu dieser Kategorie; sie wird *Kontrollwort* genannt. Zum anderen die, bei denen hinter dem Gegenschrägstrich nur ein Zeichen folgt, das *kein* Buchstabe ist. Zur letzten Kategorie gehört auch das '`\\"`'. Diese Sequenzen heißen *Kontrollzeichen.*

Folgt einem Kontrollwort ein Leerzeichen, dann wird es ignoriert. Dieses Leerzeichen dient quasi nur der Begrenzung des Kontrollwortes. Was nun aber, wenn sie ein Leerzeichen hinter einem Kontrollwort schreiben wollen? Da TEX mehrere Leerzeichen immer zu einem zusammenfaßt, wäre es falsch einfach zwei, statt des einen Leerzeichens zu schreiben. Statt dessen gibt es eine Kontrollsequenz, die genau das gewünscht Ergebnis bereithält:[1]

```
\␣
```

Da, wie schon eben gesagt, TEX mehrere Leerzeichen immer zu einem zusammenfaßt, müßten sie, wollten sie drei Leerzeichen hinter einem Kontrollwort erzeugen schon: '`\␣\␣\␣`' schreiben, um das Gewünschte zu erhalten.

[1] Das Zeichen ␣ bezeichnet dabei ein Leerzeichen.

Übrigens, statt des Leerzeichens kann auch ein Tabulatorzeichen, oder ein Zeilentrenner stehen, da TEX letztere in Leerzeichen umwandelt.

Sie fragen sich sicher schon, wieso man hinter einem Kontrollwort ein Leerzeichen schreiben will? Nun, die Antwort ist ganz einfach: Um z.B. die Zeichenfolge 'TEX' zu erhalten, gibt es den Befehl: '`\TeX`'. Wenn sie nun einen Satz wie:

> TEX gefällt mir.

schreiben wollen, dann müssen sie eingeben:

```
\TeX\␣gef\"allt␣mir.
```

Würden sie '`\TeX`' nicht mit einem Gegenschrägstrich abschließen, dann sähe das Ergebnis so aus:

> TEXgefällt mir.

Und das gefiele uns dann nicht mehr. Es darf allerdings ein Kontrollwort nicht in jedem Fall mit einem Gegenschrägstrich beendet werden. Um z.B. 'TEX' zu erhalten, darf nicht:

```
'\TeX\'
```

eingegeben werden, das Ergebnis wäre:

> 'TEX́

Sie können sich vermutlich schon denken, woher der Fehler stammt, '`\'`' ist ein Kontrollzeichen, was die gewünschte Ausgabe verhinderte.

Derartige Kontrollsequenzen gibt es einige in TEX, so z.B. '`\pi`' für den Buchstaben 'π', '`\Pi`' für 'Π' und so weiter. Sie sehen hier schon eine wichtige Eigenschaft von TEX. Die Kontrollsequenzen sind alle so gewählt, daß sie möglichst *klingende* Namen haben. Dies gewährleistet, daß sie die wichtigsten schnell lernen können.

Die Kontrollsequenzen werden weiterhin unterschieden in sogenannte: *Primitive* und *zusammengesetzte* Kontrollsequenzen. Primitive sind nicht aus anderen Kontrollsequenzen zusammengesetzt. Zu dieser Kategorie gehören '`\input`' oder '`\accent`'. Wohingegen '`\\"`' ein zusammengesetzte Kontrollsequenz ist, die u.a. auf '`\accent`' zurückgreift. Sie werden sich fragen, warum man überhaupt zusammengesetzte Kontrollsequenzen zur Verfügung stellt. Nun ganz einfach, die anderen sind einfach zu **primitiv**. Es wäre zu mühsam einen Text nur mit Primitiven zu schreiben, ausserdem kann man mit der Änderung *einer* zusammengesetzten Kontrollsequenz das Aussehen im ganzen Text beeinflußen. Wie man aus TEX Kontrollsequenzen neue zusammensetzt, und wie man die Sequenzen ändert, lesen sie in einem späteren Kapitel.

Insgesamt bietet ein nicht schon verändertes TEX ca. 300 Primitive und 600 zusammengesetzte Kontrollsequenzen an. Erschrecken sie nicht vor der großen Zahl. Mit den meisten werden sie nie zu tun haben, und die, die sie brauchen haben größtenteils selbsterklärende Namen.

Wie kann man nun unterscheiden, ob eine TEX Kontrollsequenz zusammengesetzt, oder primitiv ist? Nun, man kann natürlich in diesem Buch nachlesen, wo die Befehle erklärt werden, man kann aber auch TEX veranlassen die Antwort zu geben. Dazu gibt es eine weitere Kontrollsequenz:

```
\show
```

Wenn sie in einen TEX File '`\show\input`' schreiben, dann gibt TEX auf dem Bildschirm aus:

```
>␣\input=\input.
```

Sie erinnern sich daran, daß '`\input`' eine primitive Kontrollsequenz ist. Anders hingegen die Reaktion, wenn sie auf ähnliche Art den Befehl: '`\thinspace`' untersuchen. Sie erhielten das Ergebnis:

```
\> thinspace=macro:
->\kern .16667em.
```

Hierbei ist '`\kern`' allerdings wieder ein Primitiv.

2.2 Die Zeichensätze

2.2.1 Die Schrifttypen

Mitunter wollen sie auf einen anderen Zeichensatz umschalten, z.B. etwas **fett** schreiben, oder durch *schrägstellen* hervorheben. Um dies zu erreichen müssen sie einfach einige Befehle in den Text schreiben, die die Schrift umschalten. Mit der Eingabe von:

```
z.B.\ etwas \bf fett \rm schreiben, oder durch \sl schr\"agstellen \rm her..
```

erhalten sie obiges Ergebnis. TEX arbeitet mit Zeichen, die zu Gruppen von 256 Zeichen zusammengefaßt sind, die sogenannten Zeichensätze. TEX bietet folgende Befehle um auf einen anderen Zeichensatz umzuschalten:

`\rm` schaltet auf die Schriftart 'Roman'	Roman
`\sl` schaltet auf die geneigte Roman Schrift	*Geneigt*
`\it` schaltet auf die Schrift 'Italic'	*Italic*
`\tt` schaltet auf eine Schreibmaschinenschrift	`Schreibmaschine`
`\bf` schaltet auf fette Schrift um	**Fett**

Es gibt einen Unterschied zwischen der geneigten und er Italic-Schrift. Der Unterschied liegt darin, daß es sich bei der geneigten Schrift um die Schriftart 'Roman' handelt, die einfach nur geneigt wurde. Die Schriftart 'Italic' ist dagegen eine ganz andere Schrift. Am augenfälligsten wird das, wenn sie sich mal die ungeneigte Italic Schrift ansehen[2]. Erst mit dem Fortschritt in der Erarbeitung von Zeichensätzen, wurde der Unterschied zwischen der geneigten und der Italic Schrift relevant. Früher wurde kein Unterschied zwischen Texten, die geneigt werden sollten und mathematischen Zeichen gemacht. Mittlerweile hat es sich eingebürgert, Texte in der geneigten, mathematische Formeln aber in der Schrift 'Italic' zu schreiben. Sie kennen die 'Italic' Schrift vermutlich aus mathematischen Büchern, als Bezeichnung für Variablen.

TEX beginnt die Arbeit üblicherweise in der Schriftart 'Roman'. Der Grund hierfür liegt darin, daß die Schriftart 'Roman' speziell auf gute Lesbarkeit ausgelegt ist. Sie sollten sich angewöhnen, die Texthervorhebungen nur in geringem Maße zu benutzen. Längere Texte in einer anderen Schriftart, als Roman, sind nicht gut lesbar.

Nachdem sie auf eine andere Schriftart umgeschaltet haben, müssen sie normalerweise erst wieder auf die Schriftart 'Roman' zurückschalten, bevor sie weiter schreiben können. Dies kann auch einfacher gelöst werden, wenn sie den Text, der hervorgehoben werden soll, zusammen mit dem Zeichensatzbefehl in geschweifte Klammern schreiben. Das obige Beispiel hätte man also auch schreiben können, als:[3]

```
z.B.\ etwas {\bf fett} schreiben, oder durch {\sl schr\"agstellen} her..
```

[2] Wie soeben geschehen.

[3] Bei dieser Art der Schriftartänderung handelt es sich um ein Beispiel für die Gruppierung, die im nächsten Abschnitt vorgestellt wird.

Die geneigten Schriften, also die geneigte Roman Schrift und die Schrift Italic, werfen ein kleines Problem auf. Betrachten sie z.B. den folgenden Satz, der aus geraden und geneigten Zeichen gemischt ist:

Ich sehe *fünf* Indianer.

Dadurch, daß das Wort 'fünf' in einer geneigten Schrift dargestellt wird, erscheint der Abstand zwischen dem zweiten 'f' des Wortes 'fünf' und dem 'I' von 'Indianer' zu klein. Für diesen Fall bietet TeX einen besonderen Befehl '`\/`'. Mit diesem Befehl wird ein zusätzlicher Zwischenraum beim Umschalten von einer geneigten Schrift auf eine gerade Schrift eingefügt. Dieser Zwischenraum ist abhängig vom Buchstaben, also für ein 'f' größer, als z.B. für ein 'e'. Wird der obige Satz als:

```
Ich sehe {\sl f\"unf\/} Indianer.
```

eingegeben, dann erscheint er so:

Ich sehe *fünf* Indianer.

Tatsächlich gibt es für jedes Zeichen, also auch für die Zeichen der geraden Schriften eine sogenannte *Italic Korrektur*, nur ist sie bei den meisten Zeichen gleich Null.

Noch eine Nachbemerkung zu den verschiedenen Schriften. Früher galt es als guter Schriftstil, einen Punkt oder ein Komma in derselben Schriftart zu schreiben, in der auch das Wort davor geschrieben war. Spätestens bei einem 'Italic' Semikolon sieht das aber ziemlich lächerlich aus. Es hat sich daher durchgesetzt, die Satzzeichen in der normalen Schriftart zu schreiben.

Schließlich gibt es bei TeX noch einen Befehl für eine Schriftart, die es gar nicht gibt. '`\nullfont`' schaltet auf einen *leeren* Zeichensatz um, der keine Zeichen enthält. Dieser wird gebraucht, wenn man als Argument einen Zeichensatz angeben muß, aber gar keine Zeichen ausgeben will.

2.2.2 Die Schriftgrößen

Schriften können sich in der Größe genauso voneinander unterscheiden, wie in der Form. Die Schrift, die sie hier lesen hat die Größe von 10pt[4].

Jeder Zeichensatz in TeX steht in Zusammenhang mit einem Kontrollwort. Für die 10pt Roman Schrift gilt z.B. der Befehl '`\tenrm`', für die entsprechende 9pt Schrift '`\ninerm`'. Die Kontrollworte für die entsprechenden geneigten Schriften heißen '`\tensl`' bzw. '`\ninesl`'. Diese Kontrollworte sind nicht in TeX *eingebaut*, sie sind keine Primitive, sie dienen nur der besseren Ansprechbarkeit der Schriftgrößen.

Wenn verschiedengroße Schriften zusammen gebraucht werden, dann orientiert sich TeX an einer *Grundlinie*. Die Eingabe von:

```
\tenrm kleiner \ninerm und kleiner
\eightrm und kleiner \sevenrm und kleiner
\sixrm und kleiner \fiverm und kleiner \tenrm
```

ergibt im Text: kleiner und kleiner und kleiner und kleiner und kleiner und kleiner .

Wir kennen jetzt zwei Befehle um auf die 10pt Roman Schrift umzuschalten '`\tenrm`' und '`\rm`'. Worin besteht der Unterschied? Nun, '`\rm`' schaltet gar nicht auf die 10pt Schrift um, sondern in die derzeit aktuelle Größe der Schriftart Roman. Der Sinn dieses Befehls liegt darin, daß man ihn

[4] Das pt-Maßsystem wird später noch genauer vorgestellt. Hier reicht es zu wissen, daß die Klammern der Schrift 10pt hoch und der Gedankenstrich 10pt breit ist (Übrigens nicht hier in der Fußnote, die Schrift ist kleiner)

benutzen kann, ohne an die augenblicklich gültige Größe der Schrift denken zu müssen. '`\tenrm`' schaltet dagegen tatsächlich auf die 10pt Roman Schriftart um.

Wie bringt man TEX nun bei, welchen Zeichensatz man benutzen möchte. Die Zeichensätze existieren ja als Zeichensatzfiles in ihrem System. Nun, sie geben die Definition:

```
\font\ninerm=cmr9
```

ein, und ab sofort weiß TEX was unter dem Kontrollwort zu verstehen ist, nämlich daß ab da der Zeichensatz zu verwenden ist, der in der Datei *cmr9.xxx* steht (um das 'xxx' kümmern wir uns später)[5].

Außer den verschieden großen Zeichensätzen, kann man einen einzelnen Zeichensatz auch noch vergrößert oder verkleinert benutzen. Das definiert man:

```
\font\cs=<externer Zeichensatzname> at <gew\"unschte Gr\"o\ss{}e>
```

Mit '`\cs`' steht dann der vergrößerte Zeichensatz zur Verfügung, vorausgesetzt, er existiert in ihrem System. Mit:

```
\font\grossfiverm=cmr5 at 10pt
```

erhalten sie z.B. die Schrift 'cmr5' auf 10pt vergrößert, also in doppelter Größe.

Sie werden sich jetzt sicher fragen, wieso man denn dann überhaupt noch eine 10pt Schrift (cmr10) braucht? Der Grund ist einfach, die beiden Schriften sehen unterschiedlich aus[6]. Sie sollten die 10pt Schrift immer dann verwenden, wenn sie davon ausgehen können, daß ihr Text in Originalgröße gelesen werden wird, wenn ihr Text aber, z.B. auf fotographischem Wege, verkleinert werden soll, dann wählen sie die vergrößerte 5pt Schrift. Um den Unterschied deutlich zu machen ein Beispiel:

Hier in 10pt und hier vergrößert

Es gibt noch eine andere Möglichkeit den Vergrößerungsfaktor einer Schrift anzugeben. Mit der Eingabe von:

```
\font\grossfiverm=cmr5 scaled 2000
```

erreichen sie dasselbe Ergebnis wie mit der Angabe von '`at 10pt`', nämlich die Verdoppelung. Die Zahlenangabe hinter dem Wort '`scaled`' gibt den Vergrößerungsfaktor in Promille an. Die Basis der Berechnung ist hier also 1000. 1500 wäre die eineinhalbfache Vergrößerung.

Es hat sich bewährt Zeichensätze mit den Potenzen von 1.2 zu vergrößern. Daher gibt es in TEX auch Kontrollsequenzen, die diese Vergrößerungen einfach bereitstellen. Mit:

```
\font\bigtenrm=cmr10 scaled\magstep2
```

wird die 10pt Schrift in 1.2×1.2-facher Vergrößerung bereitgestellt. Die abkürzende Kontrollsequenz heißt also '`\magstepn`', wobei 'n' üblicherweise für: $0, 1, 2, 3, 4, 5$ steht. Die Vergrößerungen sehen dann so aus:[7]

Hier normal '`\magstep0`'
einmal um 1.2 vergrößert '`\magstep1`'
und zweimal um 1.2 vergrößert '`\magstep2`'

[5]'cmr9' bedeutet übrigens: *Computer modern Roman* mit 9pt

[6]Der Grund für das unterschiedliche Aussehen liegt einfach darin, daß z.B. in der Schrift cmr10 noch Striche von der Stärke eins möglich sind, bei der vergrößerten cmr5 Schrift, wäre hingegen ein Strich der Stärke eins schon auf die Stärke 2 verdoppelt worden.

[7]Es gibt auch noch die Vergrößerungsstufe '`\magstephalf`', die eine Schrift um den Faktor: $\sqrt{1.2}$ vergrößert, also genau zwischen 0 und 1 liegt.

2.3 Gruppierungen

Mitunter ist es erwünscht, wenn ein Teil eines Textes, wie eine Einheit behandelt wird. Dazu stellt TEX die Zeichen '`{`' und '`}`' bereit. Alles, was zwischen diesen Zeichen steht wird als Einheit behandelt. Die beiden Klammern sind also ähnlich spezielle Zeichen, wie das '`\`'.

Die Gruppen in TEX haben eine ähnliche Aufgabe, wie die sogenannte *Blockstruktur* bei Programmiersprachen. Definitionen, die innerhalb einer Gruppe getroffen wurden, gelten nur innerhalb der Gruppe, und verlieren ihre Gültigkeit, sobald die Gruppe endet. In der Sprache der Programmiersprachen sind sie *lokal.*

Im letzten Abschnitt wurde schon ein Beispiel für Gruppen geliefert, hier folgt ein weiteres. Wir haben schon gelernt, daß man bei dem Logo 'TEX' unterscheiden muß, welche Kontrollsequenz man schreibt, je nachdem, ob ein Leerzeichen folgen soll, oder nicht. Man kann dieses Problem auch lösen, indem man:

```
{\TeX}
```

schreibt. Die zweite schließende Klammer beendet die kleine Gruppe, und danach wird alles wieder normal behandelt, d.h. ein Leerezeichen ist dann nicht mehr Begrenzer einer Kontrollsequenz, sondern wieder ein ganz normales Leerzeichen. Auch:

```
\TeX{}
```

hätte denselben Erfolg gebracht, da sich nun der Kontrollsequenz '`\TeX`' eine leere Gruppe anschließt.

Als weiteres Beispiel, was uns dann weiter führen wird dient folgendes:

```
\centerline{Diese Information ist zentriert}
```

Das Beispiel benutzt die Kontrollsequenz '`\centerline`'. Diese Kontrollsequenz zentriert das folgende Zeichen in einer eigenen Zeile. Die Gruppenklammern sind nötig, da sonst nur das 'D' von 'Diese' zentriert würde, der Rest aber in die nächste Zeile geschrieben würde, — nicht im Sinne des Erfinders.

Es ist natürlich auch möglich Gruppen ineinanderzuschachteln. Die Eingabe von:

```
\centerline{Diese Information ist {\it zentriert}}
```

ergibt als Ergebnis:

Diese Information ist *zentriert*

Ich habe oben geschrieben, daß Definitionen, die innerhalb von Gruppen getroffen werden, nur innerhalb der Gruppe Gültigkeit haben. Das gilt auch für alle anderen Befehle innerhalb der Gruppe. Beim letzten Beispiel betraf die Umschaltung auf Italic nur das Wort 'zentriert'. Es gibt mitunter Situationen, wo man innerhalb einer Gruppe etwas verändern möchte und diese Veränderung über die Gruppengrenzen hinaus wirken soll. Das ist möglich, indem man den Befehl '`\global`' vor den Befehl schreibt. Hätten wir im letzten Beispiel:

```
\centerline{Diese Information ist {\global\it zentriert}}
```

geschrieben, dann wäre auch aller nachfolgender Text in Italic geschrieben worden, bis zu einem anderen, auf eine andere Schriftart umschaltenden Befehl. '`\global`' bewirkt, das der unmittelbar darauf folgende Befehl für alle existierenden Gruppen wirksam wird.

Eine weitere Möglichkeit eine Gruppe zu begrenzen besteht mit den Primitiven '`\begingroup`' und '`\endgroup`'. Diese machen es leichter möglich eine Gruppe in einer Kontrollsequenz beginnen und in einer anderen enden zu lassen. Man muß nur darauf achten, daß sie die Gruppen nicht überlappen. Konstruktionen, wie:

```
{ \begingroup } \endgroup
```

sind unzulässig.

2.4 Die Arbeit mit TEX

Sie wissen nun schon genug, um mit TEX, vom Brief angefangen bis zum Buch alles schreiben zu können (wirklich). Das Einzige, was noch fehlt, ist der eigentliche Umgang mit dem Programm TEX selber. In diesem Abschnitt sollen ihnen einige Beispiele den Umgang mit dem Programm TEX verdeutlichen und ihnen das notwendige Rüstzeug zu eigenen Experimenten geben.

2.4.1 Der erste Text

Rufen sie TEX auf[8]. Es müßte sich mit einer Meldung, nicht unähnlich dieser:

```
This is TeX, Version 3.0 (preloaded format=plain 89.7.15)
**
```

melden. Für den Anfang geben sie bitte '`\relax`' ein, was TEX veranlaßt nichts zu tun, TEX müßte sich nun mit *einem* Sternchen wiedermelden. Versuchen sie nun '`Hallo?`', oder etwas ähnliches. Und zum Schluß schreiben sie noch '`\end`'.

TEX sollte nun mit einem '`[1]`' antworten (Es wurde soeben die erste Seite ihres ersten Textes ausgegeben. Der Inhalt steht in einer Datei mit dem Namen: `texput.dvi`[9].

2.4.2 Nun die Eingabe aus einem File

Sie können TEX natürlich *on line* benutzen, das ist aber nicht sehr effektiv. Besser ist es ihre Eingabe in einen File zu schreiben, sie können dazu einen Editor ihrer Wahl benutzen. Diesen File werden wir dann später an TEX übergeben.

Schreiben sie als erstes den folgenden, aus dem TEXBook hinlänglich bekannten Text:

```
\hrule
\vskip 1in
\centerline{\bf A SHORT STORY}
\vskip 8pt
\centerline{\sl by A. U. Thor}
\vskip .6cm
Once upon the time, in a distant
  galaxy called \\"O\\"o\c c,
there lived a computer
named R. J. Drofnats

Mr.~Drofnats---or ``R. J.,'' as
he preferred to be called---
was happiest when he was at work
typesetting beautiful documents.
\vskip 1in
\hrule
\vfill\eject
```

Schreiben sie den Text genauso, wie sie ihn hier sehen, ich beziehe mich im weiteren auf die Zeilennummern.

Sehen wir uns den Text doch noch etwas an, bevor wir ihn an TEX übergeben. In Zeile 1 und 17 steht jeweils der Befehl '`\hrule`', der eine dünne Querlinie mit der Breite einer Seite zieht. Dann

[8]Der Aufruf eines Programms ist von Betriebssystem zu Betriebssystem unterschiedlich, der Programmname von TEX lautet i.a. `virtex`

[9]`dvi` heißt *device independent*, also geräteunabhängig. Fragen sie jemanden, der sich damit auskennt, wie man dieses Ergebnis ansehen, besser noch ausdrucken kann.

gibt es in den Zeilen 2, 4, 6 und 16 noch den Befehl '`\vskip`', der einen zusätzlichen vertikalen Zwischenraum einfügt. Die Größe des Zwischenraums steht unmittelbar hinter dem Befehl, aber um die Maßeinheiten kümmern wir uns später. Weitere Befehle kennen sie ja schon, so z.B. den Befehl '`\centerline`'. Auch die Befehle '`\\"`' sollten ihnen schon bekannt vorkommen, es handelt sich um Umlautbefehle, so daß der Ausdruck: '`\\"O\\"o\c c`' als: '
Ö
öç' erscheint.

Bei genauerem Hinsehen fallen allerdings noch ein paar Kleinigkeiten auf: Zum einen taucht zwischen manchen Wörtern, statt eines Leerzeichens das Zeichen: '`~`' auf. Dieses wird von TEX wie ein Leerzeichen behandelt, nur daß dort die Zeile nicht umgebrochen werden darf. Nützlich z.B. bei Initialen. Weiterhin ist die leere Zeile 11 bemerkenswert. Mit einer derartigen Leerzeile wird TEX mitgeteilt, daß ein Abschnitt zu Ende ist, und danach ein neuer anfängt. Die übliche Neuformatierung, die sie vielleicht von Textverarbeitungsprogrammen gewöhnt sind, entfällt.

Am Schluß unseres Textes stehen die Befehle: '`\vfill`' und '`\eject`'. Diese bewirken, daß die laufende Seite mit leeren Zeilen aufgefüllt wird (`\vfill`) und dann ausgegeben wird (`\eject`).

Jetzt wird es Zeit TEX wieder zu starten, aber nachdem es uns mit den beiden Sternchen begrüßt, geben wir jetzt nicht wieder '`\relax`' ein, sondern '`story`'[10].

Sie haben sich vielleicht schon gewundert, wieso TEX bei der ersten Meldung zwei, später aber nur ein Sternchen ausgibt. Der Grund hierfür ist einfach. Die beiden Sternchen bedeuten, daß jede Eingabe, die nicht mit einem Gegenschrägstrich oder einem Kaufmannsund ('`&`') beginnt, als Filename verstanden wird und TEX deshalb ein '`\input`' vorher ausführt (Sie hätten also auch '`\input story`' schreiben können; '`\input`' beginnt mit einem Gegenschrägstrich).[11] Bei manchen Systemen ist es auch möglich all diese Angaben sofort beim Aufruf von TEX in die Kommandozeile zu schreiben. Der Aufruf sähe dann ungefähr so aus:[12]

```
virtex &plain \input story
```

Und TEX würde sofort mit seiner Arbeit beginnen.

Während der Bearbeitung von TEX taucht jetzt die Ausgabe:

```
(story.tex [1])
```

auf, und nachdem sie beim abschließend wieder auftretenden Sternchen wieder ein '`\end`' eingegeben habe, können sie den neuen File '`story.dvi`' wieder ausgeben lassen und sich das Ergebnis ansehen.

2.4.3 Fehler in der Eingabe

Damit ihnen später, wenn es dann mal zu *echten* Fehlern kommt, jede Erfahrung im Umgang mit dieser Situation abgeht, jetzt mal einige Fehlersituationen.[13]

Die Sache mit der Textbreite

Starten sie TEX und auf die Eingabeaufforderung (`**`) geben sie ein:

```
\hsize=4in \input story
```

[10] Ich gehe davon aus, daß sie ihren Text in einer Datei mit dem Namen `story.tex` abgespeichert haben.

[11] Wenn dem doppelten Sternchen ein Kaufmannsund folgt, dann nimmt TEX an, daß es sich um die Angabe eines Formatfiles handelt. Wenn bei ihrem TEX kein Formatfile mitgeladen wird, dann müßte ihre Eingabe lauten: `&plain \input story`.

[12] Bei manchen Betriebssystemen ist es darüber hinaus notwendig das Argument in Anführungszeichen zu schreiben.

[13] D. Knuth schreibt hierzu, daß man sich immer vor Augen halten soll, daß man mit Fehlern nicht etwa den Computer beleidigt, so daß er später die Mitarbeit verweigert, Fehler sind dazu da gemacht zu werden um aus ihnen zu lernen.

Die Kontrollsequenz '`\hsize`' regelt die bedruckbare Breite des Textes, normalerweise wird sie in *Plain* vordefiniert, hier aber auf die Breite von 4 Inch reduziert.[14] Wenn jetzt wieder das Sternchen auftaucht, geben sie ein:

```
\hsize=3in \input story
```

und nachdem sie TEX darüber informiert hat, daß es die zweite Seite des Dokuments auch geschrieben hat (die erste Seite war ja 4 Inch breit, und wird nicht überschrieben, sondern jede weitere Seite wird einfach angehängt), geben sie der Reihe nach auch noch ein:

```
\hsize=2.5in \input story
\hsize=2in \input story
\end
```

Übergehen sie zunächst einmal alle Meldungen, die über ihren Bildschirm huschen; führen sie einfach erst mal alles so aus, wie hier beschrieben. Sie müssen keine Angst haben, daß ihnen etwas entgeht. Alle Fehlermeldungen, die sie auf dem Bildschirm sehen, werden außerdem noch in einem sogenannten Transscriptfile, oder Logfile gesichert, so daß sie sich dort später in Ruhe ansehen können.[15]

Schauen wir uns doch mal eine der Fehlermeldungen an, die sie erhalten haben dürften. Wenn sie nicht genau so aussehen, wie die hier beschriebenen, dann macht das nichts, die Struktur der Fehlermeldung ist die gleiche.

```
Overfull \hbox (0.98807pt too wide) in paragraph at lines 7--11
\tenrm tant galaxy called []O^^?o^^Xc, there lived|
Overfull \hbox (0.4325pt too wide) in paragraph at lines 7--11
\tenrm a com-puter named R. J. Drof-nats. ||
Overfull \hbox (5.32132pt too wide) in paragraph at lines 12--16
\tenrm he pre-ferred to be called---was hap-||
```

Jede übervolle Box liegt an einer bestimmten Stelle im ursprünglichen File (beim erste Fall zwischen den Zeilen 7 und 11) und außerdem läßt sich ablesen, wieviel die Box zu groß ist (im ersten Fall 0.98807pt). Die Fehlermeldung gibt auch noch einen Teil der Box aus, die zu lang ist; hier Zeichen im Zeichensatz *tenrm*. Die beiden Klammern vor dem 'O' bedeuten, daß hier etwas kompliziertes, wie eine Absatzeinrückung, oder wie in diesem Fall die Konstruktion von '
Ö
öç' vorliegt. Die Meldung gibt auch noch an, an welchen Stellen TEX trennen wollte, bevor der Fehler auftrat. Beim zweiten Fehler z.B. wollte TEX 'Drof-nats' trennen. Der Trennalgorithmus von TEX ist ausserordentlich gut, aber natürlich nicht perfekt, so wurde z.B. nicht erkannt, daß 'galaxy' hätte hinter dem 'l' getrennt werden können.[16]

Jetzt aber zum Grund für die Fehlermeldungen. TEX läßt zwischen Worten variablen Wortabstand zu. Betrachten sie die folgenden Zeilen:

Hier mal mit ganz kleinen Abständen
und hier mal mit ganz großen

Es ist deutlich, daß die Wortabstände unterschiedlich sind. TEX berechnet nun für jede Zeile einen Wert, genannt '`badness`'. Dieser gibt an, wie gut oder schlecht die Wort in der Zeile verteilt sind. Dabei geht TEX von einem Idealabstand aus. Wird dieser unter- oder überschritten, dann erhöht sich demgemäß der Wert von `badness`. Normalerweise läßt TEX nur Zeilen mit einer badness

[14] Ein Inch entspricht etwa 2.54 cm.

[15] Tatsächlich ist die Fehlermeldung im Logfile sogar noch etwas ausführlicher.

[16] Man kann TEX etwas auf die Sprünge helfen, indem man entweder an der entsprechenden Stelle '`gal\-axy`' schreibt, oder, wenn man es global für den ganzen Text haben will, zu Beginn des Textes: '`\hyphenation{gal-axy}`'.

von weniger als 200 ohne Fehlermeldung zu. Dieser Wert (200) steht in einer Variablen names '`tolerance`'. Setzen sie diesen Wert doch einfach einmal höher. Starten sie TEX erneut und geben sie

```
\hsize=2in \tolerance=1600 \input story
```

ein. Es wird ihnen keine '`Overfull hbox`' mehr gemeldet. Ok, ok, es gibt da noch ein paar Meldungen wegen '`Underful hbox`'; das liegt daran, daß TEX alle Boxen meldet, die eine badness über einem Wert haben, der in '`\hbadness`' steht. Der vordefinierte Wert für '`\hbadness`' ist 1000.

Auch den letzten Fehler können wir noch beseitigen, indem der Befehl '`\raggedright`' mitangegeben wird. Er veranlaßt TEX sich nicht mehr um den rechten Randausgleich zu kümmern. Bis zum rechten Rand wird mit Leerzeichen aufgefüllt.

Abschließende Bemerkungen zum Thema. Man kann sich z.B. fragen, ob es besser ist, die `tolerance` oder den Wert für `hbadness` hochzusetzen. Tatsächlich haben beide Möglichkeiten ihre Vor- und Nachteile. Wenn sie wirklich ihren Text noch korregieren wollen, dann wählen sie lieber die erste Alternative, TEX gibt ihnen dann noch an, wieweit die Boxen das erlaubte Maß überschreiten, und sie können darauf reagieren. Geht es ihnen darum einfach ihren Text möglichst schnell und fehlerfrei zu setzen, dann wählen sie die zweite Methode.

Es gibt aber auch noch eine Variante: TEX meckert nicht jede zu voll geratene Box an. Wenn eine Box nur *etwas* zu groß ist, erhalten sie keine Fehlermeldung und auch TEX ignoriert das herausragen der Zeile. Der Wert, um den ein Text herausragen darf steht in '`\hfuzz`', und er ist üblicherweise mit 0.1pt vorbesetzt. Sie können ihn aber auch ändern, wie alle Werte in TEX.

Und jetzt mal *richtige* Fehler

Im letzten Teil dieses Abschnitts wollen wir mal ein paar *richtige* Fehler begehen. Ändern sie in ihrem File `story.tex` die Zeile 3 in:

```
\centerline{\bf A SHORT \ERROR STORY}
```

in der zweiten Zeile schreiben sie statt '`\vskip`' einfach '`\vship`'. Um das Maß voll zu machen, schreiben sie statt des Filenamens `story` auch noch `sorry` in die Eingabe von TEX.

TEX wird sie auffordern, einen andern Filenamen einzugeben, da er den von ihnen angegebenen nicht finden kann. Versuchen sie nur sich durch ein neues Drücken der Returntaste aus dieser Lage zu befreien, letztlich ist es doch besser, sie schreiben den richtigen Filenamen. TEX wird dann mit der Arbeit beginnen und nach verhältnismäßig kurzer Zeit mit der Fehlermeldung:

```
! Undefined control sequence.
l.2 \vship
          1in
?
```

aufwarten. Die Fehlermeldung beginnt mit einem '`!`' und der Angabe, worin der Fehler besteht. Es folgt die Angabe der Zeilennummer, und dem Text, soweit TEX ihn bisher bearbeitet hat. Eine Zeile tiefer erscheint der Text, der als nächstes zu bearbeiten wäre. Abgeschlossen wird das Ganze durch ein Fragezeichen, was anzeigt, daß TEX eine Eingabe von ihnen erwartet, wie es weiter gehen soll. Sie haben natürlich keine Ahnung (und wenn sie schon welche haben, dann haben sie vergessen) wie es weitergehen soll. Tippen sie deshalb auch mal ein Fragezeichen, und TEX wird ihnen folgendermaßen antworten:

```
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
```

Es handelt sich dabei um eine Menuauswahl, wobei sie jetzt nur noch entscheiden müssen, wie sie reagieren wollen. Sie haben die folgenden Möglichkeiten:

1. Drücken sie einfach die Returntaste. TEX versucht dann soweit es möglich ist alleine aus den Schwierigkeiten heraus zu kommen.

2. Drücken sie die Taste 'S'. Sie wechseln damit in einen anderen Modus. Ab sofort hält TEX bei Fehlern nicht mehr an, die Fehlermeldungen sausen nur so über den Bildschirm (eventuell) und sie können sie nur noch später im Logfile richtig lesen. TEX verhält sich also so, als würden sie bei diesem und jedem weiteren Fehler immer wieder die Returntaste drücken.

   Denselben Effekt können sie auch erreichen, wenn sie '`\scrollmode`'in ihren File als Befehl hineinschreiben.

3. Drücken sie die Taste 'R'. TEX reagiert wie im letzten Fall, allerdings noch etwas rabiater. TEX hält jetzt nicht einmal mehr an, wenn ein falscher Filename eingegeben wurde.

   Mit dem Befehl '`\nonstopmode`' im Text erreichen sie denselben Effekt.

4. Die Taste 'Q'. Auch hier wieder eine ähnliche Reaktion, wie bei 'R', nur das darüberhinaus auch noch alle Meldungen auf dem Bildschirm unterdrückt werden.

   Hierfür gibt es im Text den Befehl '`\batchmode`'

5. Mit 'I' kann etwas in den Text eingefügt werden. Schreiben sie dazu das, was sie einfügen wollen unmittelbar hinter das 'I'. TEX tut so, als stünde dieser so eingefügte Text vor dem, den es noch lesen muß. Das Return, das sie zur Beendigung der Eingabe eingeben, wird *nicht* als Leerzeichen interpretiert.

6. Wenn sie eine Nummer, die kleiner als 100 sein muß, eingeben, dann ignoriert TEX die entsprechende Anzahl von Buchstaben, bzw. Kontrollsequenzen in ihrem Text. Damit sind die gemeint, die dem Fehler folgen!

7. Mit 'H' erhalten sie eine weitergehende Hilfe. TEX unterscheidet zwischen einer formalen und einer informellen Fehlermeldung. Normalerweise erhalten sie nur die formale, in den Logfile wird allerdings sofort auch die informelle geschrieben. Mit 'H' erhalten sie die volle Fehlermeldung.

8. Mit 'X' verlassen sie TEX an dieser Stelle. Dies bietet sich an, wenn es einfach schon zu viele Fehler geworden sind, oder wenn es zuviele Folgefehler gibt.

9. Auch 'E' verläßt TEX, bereitet aber den Computer auf eine Korrektur des Eingabefiles vor. Dieses Feature wird nicht von allen Rechnern und TEX Versionen unterstützt.

Ach ja, Ich hatte oben von den Befehlen geschrieben, mit denen die Modi auch im Text gesetzt werden können. Zurückgesetzt werden sie Modi durch den Befehl: '`\errorstopmode`'.

Wie begegnen wir nun unserer ersten Fehlermeldung? Schreiben sie doch einfach:

```
I\vskip
```

Die unbekannte Kontrollsequenz '`\vship`' wird ignoriert, und statt dessen die gewollte eingefügt (allerdings nicht in ihrer Textdatei, da müssen sie sie später erst noch wieder ändern). Sie hätten auch '`3`' eingeben können, dann wäre die Angabe '`1in`' auch ignoriert worden. Hätten sie einfach Return gedrückt, stünde ein seltsames '1in' in ihrem Text.

Nun zum nächsten Fehler. TEX beantwortet ihn folgendermaßen:

```
! Undefined control sequence.
<argument> \bf A SHORT \ERROR
                             STORY
```

```
\centerline #1->\line {\hss #1
                              \hss }
l.3 \centerline{\bf A SHORT \ERROR STORY}
|null
?
```

Diese Fehlermeldung ist etwas komplizierter, da der Text:

```
\bf A SHORT \ERROR STORY
```

als Argument an den Befehl '`\centerline`' übergeben worden ist. Der Fehler tritt auf, bei der Bearbeitung von '`\centerline`'. Dieser Befehl ist in *Plain* TEX folgendermaßen definiert:

```
\def\centerline#1{\line{\hss#1\hss}}
```

Das '`#1`' steht dabei stellvertretend für das übergebene Argument.[17] Ich denke mit diesem Wissen braucht die Fehlermeldung nicht weiter erklärt zu werden. Die Verschachtelung sogenannter Makros kann sehr tief sein. Sie brauchen sich aber nicht alle Verschachtelungen in der Fehlermeldung anzusehen, die ihnen meist sowieso nichts sagt, da sie die Makrodefinitionen nicht kennen. Mit dem Befehl:

```
\errorcontextlines=...
```

können sie angeben, wieviele *Zwischen*zeilen sie angezeigt haben möchten. Ist der Wert z.B. gleich 2, dann werden nur die obersten beiden Verschachtelungen angezeigt.

[17] Um die Bedeutung der Befehle: `\hss` und `\line` brauchen wir uns jetzt noch nicht zu kümmern.

Kapitel 3

Die internen Zeichen

3.1 Wie ließt TEX, was sie schreiben?

Es dürfte ihnen schon klar geworden sein, daß ihre Texteingabe nur ungefähr etwas mit der Ausgabe zu tun hat. Zeilenumbrüche etc. werden von TEX gesetzt. Außerdem wird:

- Ein Return wie ein Leerzeichen behandelt,
- Zwei Leerzeichen wie ein Leerzeichen behandelt und
- Eine leere Zeile beendet einen Absatz.

Wenn sie sich die drei Punkte genau ansehen, wird ihnen ein Fehler auffallen. Die Regeln wiedersprechen sich. Eine Leerzeile besteht aus zwei Returns hintereinander, die wie zwei Leerzeichen hintereinander zu einem Leerzeichen zusammengefasst werden müßten. Vielleicht interessieren sie sich irgendwann einmal für die *tatsächlichen* Regeln, Hier zunächst ein kleiner Einblick, was mit ihrer Eingabe alles geschieht, bis sie zur Ausgabe geworden ist.

Die 256 verschiedenen Zeichen, die innerhalb TEX möglich sind gehören 16 verschiedenen Kategorien an. Diese Kategorien sind:

Kategorie	*Bedeutung*	*Standard*
0	Befehlsanfang	`\`
1	Blockanfang	`{`
2	Blockende	`}`
3	Math. Umschaltbefehl	`$`
4	Tabellenspalte	`&`
5	Zeilenende	Return (ASCII CR)
6	Makroargument	`#`
7	Hochstellung	`^`
8	Tiefstellung	`_`
9	Ignorieren	ASCII NULL
10	Leerzeichen	␣
11	Buchstabe	`A..Z` und `a..z`
12	Sonstige Zeichen	Ziffern, Satzzeichen, etc.
13	Aktives Zeichen	`~` Als Makroname verwendbar
14	Kommentarzeichen	`%` Kommentaranfang
15	Ungültiges Zeichen	ASCII DEL

In diesem Text sind schon mehrere Male Zeichen aufgetaucht, die nicht als normale Zeichen von TEX verstanden werden. Nun haben wir alle diese Zeichen auf einem Blick (und es sind dann auch wirklich alle). Aus *Plain* TEX heraus sind etliche dieser Zeichen, die man auch gerne drucken würde, nicht so einfach druckbar, da sie eine andere Bedeutung haben. Es sind dies die Zeichen:

`\ { } $ & # ^ _ % ~`

Einige davon lassen sich dennoch im Text ausgeben, natürlich durch Kontrollsequenzen:

'`\&`' für &, '`\%`' für %, '`\&`' für &, '`\#`' für # und '`\_`' für _.

Innerhalb des mathematischen Modus können außerdem die Kontrollsequenzen '`\{`' und '`\}`' für { und } benutzt werden. Und schließlich gibt es nicht die Akzentbefehle: '`\~`' und '`\^`' z.B. für die Akzente ẽ und â.

Jede Eingabezeile wird von TEX in sogenannte Token zerlegt. Diese Token bestehen aus Zeichen, Kontrollsequenzen, Ziffern etc. So wird die Eingabe von: '`{\hskip 36 pt}`' zerlegt in:

$$\{_1 \quad \boxed{\text{hskip}} \quad 3_{12} \quad 6_{12} \quad \sqcup_1 0 \quad p_{11} \quad t_{11} \quad \}_2$$

Die Indizes deuten dabei die Kategorie an, der das jeweilige Token angehört. $\boxed{\text{hskip}}$ hat keinen Index, da eine Kontrollsequenz keiner Kategorie angehört. Auch das Leerzeichen hinter der Kontrollsequenz wird nicht behandelt, da es nur die Kontrollsequenz abschließt. Nach der Zerlegung in Token, erscheint TEX ihre Eingabe, wie eine unheimlich lange Zeile von Token, es gibt also gar keine Absätze und sonstiges mehr, nur noch Token.

Kontrollsequenzen sind also nur *ein* Token lang, so daß es überhaupt nichts ausmacht, wie lang der Name der Kontrollsequenz ist. Die Kategorie, die einem Token zukommt, kann geändert werden. Wenn sie z.B. INITEX starten, das Programm, mit dem die Formatdateien erstellt werden können etc., dann gibt es für manche Kategorien noch gar keine Zeichen. Die Kategorie können sie einem Zeichen zuweisen, resp. die Kategorie eines Zeichens andern, mit dem Befehl:

```
\catcode`\
```

Hinter diesem Befehl müssen sie dann das Zeichen tippen, dessen Kategorie sie bestimmen oder ändern wollen.

```
\catcode`\a=14
```

wuerde z.B. dafür sorgen, daß Kommentare mit einem 'a' eingeleitet werden (nicht sehr sinnvoll).

Kontrollsequenzen werden ja als *ein* Token behandelt. Trotzdem bietet TEX eine Möglichkeit eine Kontrollsequenz in ihre Zeichen zu zerlegen. Der entsprechende Befehl heißt '`\string`'. Mit der Eingabe von:

```
\string\TeX
```

erhalten sie:

$$\backslash_{12} \quad \mathtt{T}_{12} \quad \mathtt{e}_{12} \quad \mathtt{X}_{12}$$

Umgekehrt kann natürlich auch aus beliebigen Token eine Kontrollsequenz aufgebaut werden. Der zugehörige Befehl lautet:

```
\csname TeX\endcsname
```

Hier werden die Token:

$$\mathtt{T}_{12} \quad \mathtt{e}_{12} \quad \mathtt{X}_{12}$$

zu der Kontrollsequenz: '`\TeX`' zusammengefügt.

Leider gibt es eine Einschränkung. Da zwischen '`\csname`' und '`\endcsname`' zwar beliebige Token, also auch wieder Kontrollsequenzen, stehen dürfen, aber diese müssen sich alle zu Zeichen, und nicht zu Primitiven auflösen lassen. Somit ist die Eingabe von:

```
\csname\TeX\endcsname
```

falsch, da '`\TeX`' auch auf das Primitiv '`\kern`' zurückgreift.

Wenn wir schon bei den Fähigkeiten von TEX sind etwas umzuwandeln. Natürlich ist auch das Escapezeichen '`\`' nichts gottgegebenes in TEX. Tatsächlich steht in der Variablen '`\escapechar`' der Wert des Zeichens, das dann das Escapezeichen ausmacht.

Und noch mehr Fähigkeiten zur Umwandlung: Es gibt noch die beiden Befehle: '`\number`' und '`\romannumber`', welche beide eine Zahl als Ziffernfolge ausgeben.

```
\romannumber24
```

produziert z.B. '`xxiv`'. '`\number`' gibt den Wert einfach als Zahl aus. Hier ist es natürlich nicht sinnvoll eine Zahl anzugeben, aber '`\number`' arbeitet auch mit den Registern von TEX, deren Inhalt man sich mit '`\number`' ausgeben lassen kann.

3.2 Die Sache mit den Zeichen

Etwas weiter vorne konnten sie lesen, daß TEX intern 256 Zeichen verarbeiten kann. Diese große Menge von Zeichen kann allerdings auf den meisten Tastaturen nicht direkt eingegeben werden. Es stellt sich die Frage, wie man denn am besten vorgeht. Um z.B. das 37te Zeichen eines Zeichensatzes ansprechen zu können, geben sie einfach:

```
\char37
```

ein, also '`\char`' gefolgt von der gewünschten Nummer. Auf diese Art lassen sich alle Zeichen ansprechen. So ergibt z.B.:

```
\char70 ritz
```

meinen Vornamen: 'Fritz'. Sie vermuten ganz richtig, daß das 'F' das 70. Zeichen im aktuell gültigen Zeichensatz ist. Die Nummerierung der Zeichen hängt vom sogenannten 'ASCII' Zeichensatz ab, nach dem sich TEX orientiert. Diese Orientierung findet übrigens *innerhalb* von TEX statt, unabhängig davon, welchen Zeichensatz ihr Rechner benutzt.

Innerhalb der Tabelle von Zeichen, die zur Verfügung stehen, werden die Zeichen oft mit ihrem oktalen, oder hexadezimalen Wert angesprochen. In TEX können diese Werte auch angegeben werden, wenn sie die dezimalen nicht kennen oder ausrechnen wollen. Vor den oktalen Zahlen muß ein (rechtes) Häkchen '`'`' stehen, vor den hexadezimalen ein Anführungszeichen '`\"`'. Die folgenden Angaben sind also äquivalent:

```
\char98  \char'142  \char\"62
```

Tatsächlich brauchen sie nicht einmal die Nummer des Zeichens zu kennen, dessen Wert sie angeben wollen. Das Token $`_{12}$ (linkes Häkchen) liefert den Wert des unmittelbar folgenden Buchstaben oder der Kontrollsequenz, die nur aus einem Buchstaben besteht (dann natürlich den Wert dieses Buchstabens). Man kann also angeben: '``\char`b``' oder '``\char`\b``'.[1]

Den '`\char`' Befehl dürfen sie überall benutzen um ein Zeichen anzusprechen, allerdings nicht innerhalb einer Kontrollsequenz. Wozu ist dieser Befehl denn nun eigentlich nützlich? Nun, es gibt Zeichen, wie z.B. das Zeichen 'Γ', welches im normalen Text nicht geschrieben werden kann. Da es aber das 0. Zeichen des Zeichensatzes ist, kann es einfach als '`\char0`' angesprochen werden.

Es gibt noch zwei weitere interessante Möglichkeiten ein Zeichen aus einem Zeichensatz anzusprechen und zwar durch:

```
^^
```

[1] Auf diese Art wird übrigens das Kommentarzeichen ausgegeben. Man definiert die zugehörige Kontrollsequenz als: ``\def\%{\char`\%}``. Etwas eleganter geht es aber auch mit dem Befehl: `\chardef`.

Wenn sie hinter diese beiden Zeichen ein weiteres Zeichen schreiben, dann reagiert TEX folgendermaßen: Hat das folgende Zeichen einen inneren Wert zwischen 64 und 127, dann werden 64 subtrahiert, hat es einen Wert zwischen 0 und 63, dann werden 64 addiert. Um das obige Zeichen darzustellen hätte ich also auch:

```
^^@
```

schreiben können, da '@' den inneren Wert 64 hat. '`^^@`' wird also als das 0. Zeichen gelesen.

Folgt den beiden '`^^`' eine *kleingeschriebene* hexadezimale Zahl, dann wird das entsprechende Zeichen, dem diese hexadezimale Zahl entspricht ausgegeben. '`^^7f`' ergibt also das 127. Zeichen.

Die Realisation der Zeichenordnung innerhalb von TEX erlaubt eine ungeheure Sprachunabhängigkeit. So ist es z.B. möglich auch noch mit Tastaturen zu arbeiten, die Zeichen, wie 'æ' bereitstellen, doch hier verweise ich auf das TEXBook, das diese Möglichkeit viel genauer beschreibt.

Kapitel 4

TeX's Ausgabe

Dieses Kapitel beschäftigt sich mit dem, was TeX in die `dvi` Files ausgibt. Dazu gehören, neben den Zeichen, auch die Maßeinheiten, um Positionierungen durchführen zu können.

4.1 TeX's Roman Zeichensätze

Ich habe schon früher die möglichen Zeichensätze angesprochen, die TeX ausgeben kann. Hier nun ein etwas systematischerer Zugang. Zunächst einmal die Zeichen des Roman Zeichensatzes. Da sind zunächst folgende Zeichen möglich:

> Die Buchstaben: `A` bis `Z` und `a` bis `z`
> die Ziffern `0` bis `9`
> die Satzzeichen: `:  ;  !  ?  ( ) [ ] ‘ ’ - * / .  , @`

Außerdem gibt es die Ligaturen:

`ff`	→	ﬀ	`ffi`	→	ﬃ	`‘‘`	→	“	<	→	¡
`fi`	→	ﬁ	`ffl`	→	ﬄ	`’’`	→	”	>	→	¿
`fl`	→	ﬂ	`--`	→	–	`---`	→	—			

Auch die Zeichen ‘+’ und ‘=’ sind möglich, sollten aber besser im mathematischen Modus verwendet werden, der durch ein Dollarzeichen ‘`$`’ ein- bzw. ausgeleitet wird. Sie sehen dann besser aus. Das gleiche gilt für ‘-’ und ‘/’.

Weiterhin gibt es die reservierten Zeichen:

`$  #  %  &`

die aber durch die Kontrollzeichen:

`\$  \#  \%  \&`

erhalten werden können. Außerdem reserviert *Plain* TeX noch die Zeichen:

`\  {  }  ^  _  ~`

Schließlich bleiben im sichtbaren ASCII-Code noch vier Zeichen übrig, die sie aber in normalem Text nicht verwenden sollten, resp. durch andere Zeichen(kombinationen) ersetzen sollten:

`\"  |  <  >`

TeX bietet aber auch einiges, was sie in der normalen ASCII-Codierung nicht finden. An erster Stelle die Akzente:

Eingabe	*Ausgabe*
`` \`o ``	ò
`\'o`	ó
`\^o`	ô
`\"o`	
ö	
`\~o`	õ
`\=o`	ō
`\.o`	ȯ
`\u o`	ŏ
`\v o`	ǒ
`\H o`	ő
`\t oo`	o͡o

Die Akzente lassen sich natürlich auch auf andere Buchstaben, als das 'o' setzen. Achten sie allerdings darauf, daß bei den letzten vier Beispielen ein Zwischenraum zwischen dem Kontrollzeichen und dem Argument steht. Alternativ können sie auch '`\H{o}`' schreiben.

Weiterhin kennt TEX auch Akzente *unter* den Buchstaben:

Eingabe	*Ausgabe*
`\c o`	o̧
`\d o`	ọ
`\b o`	o̲

Darüberhinaus kennt TEX auch noch einige Spezialzeichen europäischer Schriften:

Eingabe	*Ausgabe*	*Bezeichnung*
`\oe, \OE`	œ, Œ	Französische Ligatur
`\ae, \AE`	æ, Æ	Lateinische, skandinavische Ligatur
`\aa, \AA`	å, Å	Skandinavisches A mit Kreis
`\o, \O`	ø, Ø	Skandinavisches O mit Strich
`\l, \L`	ł, Ł	Polnisches L mit Strich
`\ss`	ß	Deutsches Es-Zet

Die Roman Schrift enthält außerdem noch die Buchstaben 'i' und 'j' ohne den Punkt. So ergeben '`\i`': ı und '`\j`': ȷ. Der Grund hierfür ist, daß der Punkt bei diesen Zeichen entfernt werden muß, bevor ein anderer Akzent auf die Zeichen gesetzt werden kann.

Die Schriften: '`\sl`', '`\it`' und '`\bf`' haben die gleichen Zeichen. Nur der Zeichensatz '`\tt`' verhält sich etwas anders. Zum einen kennt dieser Zeichensatz keine Ligaturen. Selbst `---` wird als `---` ausgegeben. Die Akzente: '`\H`'...'`\L`' können nicht benutzt werden, dafür aber die Zeichen: ; `|`, `<`, `>`.

Schließlich gibt es noch vier Zeichen, die in allen Zeichensätzen gleich aussehen:

Eingabe	*Ausgabe*
`\dag`	†
`\ddag`	‡
`\S`	§
`\P`	¶

Im Anhang wird noch genauer dargestellt, daß alle Akzente von TEX auf das Primitiv: '`\accent`' zugreifen. So ist z.B. '`\'#1`' äquivalent zu: '`{\accent19 #1}`'. Generell wird der Akzent mit einer Nummer angesprochen. Die Akzente werden immer richtig über die nachfolgenden Zeichen gesetzt. So würde ein Akzent über einem 'o' niedriger gesetzt, als über einem 'f', aber auch Akzente über gar keinem Zeichen sind möglich. Ahnen sie wie? '`\'{}`' ergibt: ´.

4.2 Die Maßeinheiten

4.2.1 Die eigentlichen Maßeinheiten

Im englischsprachigen Raum haben sich die *klassischen* Maßeinheiten 'Point' und 'Pica' für Drucker durchgesetzt. Neben diesen kennt TEX aber auch noch einige andere, um die Benutzung von TEX komfortabeler zu gestalten.

Abkürzung	*Bedeutung*
`pt`	Point
`pc`	Pica (1pc=12pt)
`in`	Inch (1in=72.27pt)
`bp`	Big Point (72bp=1in)
`cm`	Zentimeter (2.54cm=1in)
`mm`	Millimeter (10mm=1cm)
`dd`	Didot Punkt (1157dd=1238pt)
`cc`	Cicero (1cc = 12dd)
`sp`	Skalierter Punkt (65536sp=1pt)

Die Maßeinheiten werden immer nach folgenden Regeln angegeben:

<optionales Vorzeichen><Nummer><Maßeinheit>, oder
<optionales Vorzeichen><Ziffern> `.` <Ziffern><Maßeinheit>

Dabei können zwischen der Zahl und der Maßeinheit Leerzeichen eingefügt werden. Leerzeichen zwischen den Ziffern der Zahl, oder den Buchstaben der Maßeinheit sind dagegen nicht erlaubt.

Intern arbeitet TEX mit dem `sp`. Diese Maßeinheit ist so klein,[1] daß auch trotz der Verwendung von Integerarithmetik keine erkennbaren Fehler auftreten. Allerdings begrenzt sich dadurch die maximal angebbare Größe in TEX auf 2^{30}sp, was aber auch nicht weiter stört, da diese Größe einer Strecke von 5.7583 Metern entspricht.

Im weiteren werden des öfteren Dimensionsangaben gebraucht. Statt dort immer eine explizite Angabe zu verwenden, wird die Abkürzung: <dimen> verwendet, z.B.:

`\hsize=`<dimen>

4.2.2 Maßeinheiten und Vergrößerung

Zu Beginn eines Textes können sie eine globale Vergrößerung oder Verkleinerung angeben. Dies geschieht mit dem Befehl:

```
\magnifikation=1200
```

Die Zahl am Ende des Befehls variiert natürlich von Fall zu Fall. Bei der Zahlangabe handelt es sich um eine Promilleangabe, so daß eine Angabe von 1000 zu keiner Veränderung führt. Der Befehl:

```
\magnifikation=2000
```

verdoppelt die Größe ihrer Ausgabe. Vergewissern sie sich allerdings *vorher*, ob die entsprechenden Zeichensätze auf ihrem System verfügbar sind.

Mit der Vergrößerung oder Verkleinerung des Textes werden auch alle Maßangaben geändert, mit Ausnahme des `sp`. Wenn sie in ihrem ursprünglichen Text eine Lücke von 2cm gelassen hatten, dann erhalten sie jetzt eine 4cm große Lücke (entsprechend dem letzten Beispiel). Wenn sie das verhindern wollen, dann müssen sie *vor* der Maßeinheit das Wort '`true`' einfügen. Haben sie obige Lücke mit:

```
\vskip 2 true cm
```

definiert, dann ist sie auch in der Vergrößerung nur 2cm groß. Der Grund für dieses Verhalten liegt darin, daß alle Maßeinheiten mit einer Grundgröße in '`\mag`' verrechnet werden. Diese Berechnung verhindern sie mit der Angabe von '`true`'.

[1] Die Wellenlänge des sichtbaren Lichtes inst ungefähr 100sp

4.2.3 Relative Maßeinheiten

Jeder Zeichensatz verfügt über zwei relative Maßeinheiten:

> `em` ist die Breite eines '`\quad`' der entsprechenden Schrift
> `ex` ist die Höhe des 'x' in der entsprechenden Schrift.

Die tatsächlichen Ausmaße dieser Maßeinheiten ist von Zeichensatz zu Zeichensatz verschieden.

4.2.4 Abschließende Bemerkungen

Ihnen ist vielleicht aufgefallen, daß die Maßeinheiten nicht mit einem Gegenschrägstrich eingeleitet werden. Das liegt daran, daß TEX einige Schlüsselworte kennt. Die Maßeinheiten gehören auch dazu. Hier eine vollständige Liste der Schlüsselworte von TEX: `at`, `bp`, `by`, `cc`, `cm`, `dd`, `depth`, `em`, `ex`, `fil`, `height`, `in`, `l`, `minus`, `mm`, `mu`, `pc`, `plus`, `pt`, `scaled`, `sp`, `spread`, `to`, `true`, `width`.

Kapitel 5

Interna des Textsatzes

In diesem Kapitel werden die wichtigsten Elemente des Textsatzes von TeX dargestellt. Für die normale Anwendung ist das Wissen um diese Zusammenhänge nicht unbedingt nötig, wenn sie also nur *normale* Texte mit TeX schreiben wollen, können sie dieses Kapitel überspringen.

5.1 Boxen

Auch, wenn sich TeX des modernen Hilfsmittel *Computer* bedient, beruht die Methode, mit der Texte gesetzt werden letztlich auf der guten alten Satzkunst, die spätestens mit Guttenberg ihren Einzug in die schönen Künste gefunden hat. Früher war es nötig einzelne Zeichen, die als erhabene Grate auf Bleiklötzchen vorgegeben waren, auf einer Schiene zusammenzusetzen, und diese Schienen dann ihrerseits wieder zu größeren Komponenten zusammenzufügen. An dieser Idee hat TeX nichts geändert, nur daß statt der Bleiklötzchen nun sogenannte 'Boxen' verwendet werden.[1] Es gibt zwei Arten von Boxen: die einen können Zeichen enthalten, die anderen nicht, sind insofern also noch einfacher.

5.1.1 Die *Rule*boxen

Die Ruleboxen[2] sind eigentlich nur gefüllte Rechtecke. Je nachdem, ob sie breiter als hoch, oder umgekehrt sind, werden sie mit den Befehlen: '`\hrule`' oder '`\vrule`' angesprochen. Quadratische Boxen dieser Art können mit beiden Befehlen erzeugt werden. Der generelle Aufruf der Befehle lautet:

```
\xrule width <dimen> height <dimen> depth <dimen>
```

Zu den Größen '`width`', '`height`' und '`depth`' komme ich im nächsten Abschnitt (Das 'x' muß natürlich gegen 'v' oder 'h' ausgetauscht werden). Zu diesem Boxtyp abschließend ein Beispiel. Die Eingabe von:

```
\hrule width 10pt height 2pt
```

ergibt:

5.1.2 Die Zeichenboxen

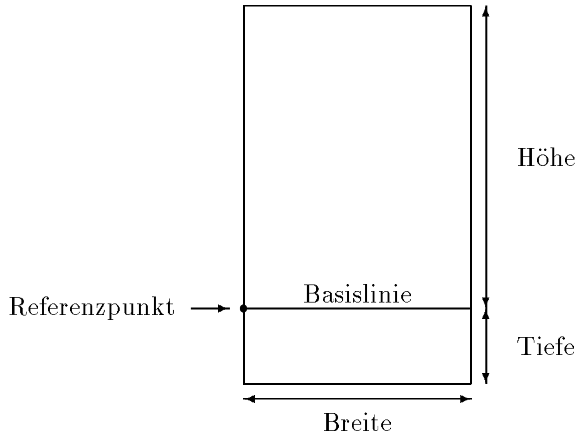
Jedes Zeichen, mit dem TeX arbeitet, stellt sich für das Programm als eine Box dar, deren Abmessungen vom Zeichensatzdesigner festgelegt wurden.[3] Bei den Abmessungen werden folgende Werte

[1] Natürlich nur intern. Der *normale* Anwender bekommt davon nichts mit.

[2] Der Begriff kommt von der üblichen Verwendung dieser Boxen. Mit ihnen werden normalerweise Linien gezogen.

[3] Genauste Darstellung im METAFONTBook.

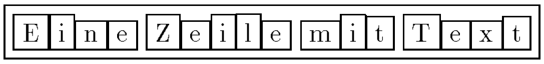
unterschieden: Breite (width), Höhe (height) und Tiefe (depth). Eine Box hat somit folgendes Aussehen:



Die Bedeutung der Basislinie und des Referenzpunktes wird sofort deutlich. Wenn nämlich die Zeichen zu einer Zeile zusammengefügt werden, dann befinden sich die Basislinien aller Zeichen auf einer Höhe.[4] Die Zeichen werden dabei zu einer übergeordneten Box, einer sogenannten '`hbox`' (horizontale Box) zusammengefügt. Die einzelnen Zeilen, jetzt zusammengefügt in ihren `hboxen` werden dann übereinandergefügt, und zwar so, daß die Referenzpunkte der `hboxen` jeweils auch wieder genau übereinander liegen. TeX wäre nicht TeX, wenn sich so etwas nicht auch *manuell* bewerkstelligen ließe. Um eine *fertige* `hbox` zu erhalten braucht man nur den richtigen Befehl und den darin enthaltenen Test. Die Sequenz:

```
\hbox{Eine Zeile mit Text}
```

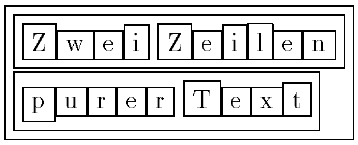
ergibt eine vollständige `hbox`. Um deutlicher zu machen, wie TeX das nun setzt, sollen die Boxen einmal mitgezeichnet werden:



Der entsprechende Befehl für die übereinandergesetzten `hboxen` heißt übrigens `vbox` (für vertikale Box, wie kaum anders zu erwarten war. Der entsprechende Befehl dafür lautet:

```
\vbox{\hbox{Zwei Zeilen}\hbox{purer Text}}
```

und das gesetzte Ergebnis sieht dann folgendermaßen aus:



Bei diesem zweiten Beispiel ist auch eine Teilbox erkennbar, die eine *echte* Tiefe besitzt. Sehen sie sich dazu bitte den Rahmen um das 'p' des Wortes 'purer' einmal genau an. Sie können sehen, daß der Rahmen tiefer hinunterreicht, als die Rahmen der anderen Buchstaben. Auch die, je nach Buchstabe, unterschiedliche Breite der Boxen ist deutlich erkennbar.

Normalerweise brauchen sie die Boxbefehl nie, sie erlauben aber absolute Kontrolle über das Aussehen des Ergebnisses. Es sind z.B. auch negative Dimensionsangaben für die Breite einer Box zulässig, so daß auch Buchstaben übereinander gedruckt werden können.

Boxen können auch in internen Registern gespeichert werden. Die Angabe von:

```
\setbox0=\hbox{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125em X}
```

speichert das TeX Logo in der `box0`, einem der internen Register von TeX. Sie können sich auch den Inhalt einer derartigen Box ansehen. Mit '`\showbox0`' erhalten sie:

[4] Die Leerzeichen sind *keine* Boxen, doch dazu im nächsten Abschnitt mehr.

```
\hbox(6.83331+2.15277)x18.6108
.\tenrm T
.\kern -1.66702
.\hbox(6.83331+0.0)x6.80557, shifted 2.15277
..\tenrm E
.\kern -1.25
.\tenrm X
```

allerdings nur in ihrem Logfile. Die Angaben im Einzelnen. Zunächst sagt ihnen TeX, daß es sich um eine Box mit der Höhe 6.83331pt, Tiefe 2.15277pt und der Breite 18.6108pt handelt. In der folgenden Zeile zeigt der führende Punkt an, daß sie sich sozusagen schon im Inneren der Box befinden. Der erste Inhalt ist das Zeichen 'T' in '`\tenrm`' als Schriftart. Jetzt folgt ein negativer Abstand von -1.66702pt, das 'E' wird etwas an das 'T' herangerückt. Gefolgt von einer weiteren Box, die 6.83331pt hoch ist, 0pt tief und 6.80557pt breit, außerdem ist sie um 2.15277pt nach unten verschoben, was die `shifted` Angabe aussagt. Der Inhalt dieser Box (angezeigt durch die *zwei* Punkte) ist das 'E'. Den Rest können sie sich jetzt sicher selber erklären. Eine derartige Aufschlüsselung ist sicher nur zu Diagnosezwecken sinnvoll.

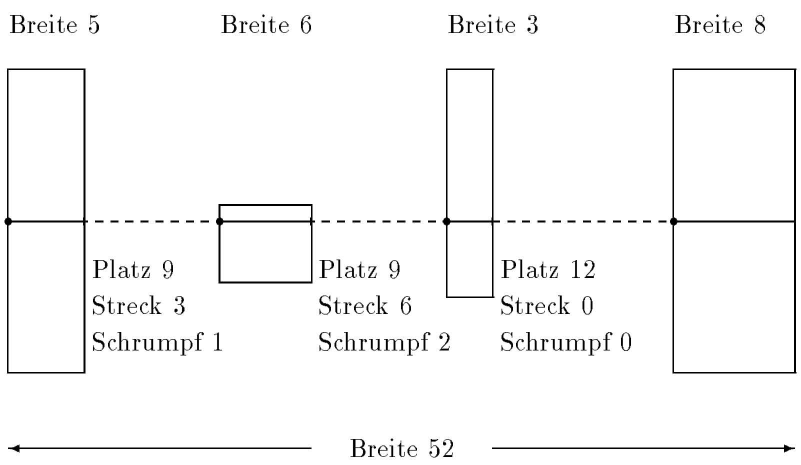
Jetzt kennen sie fast schon den ganzen Weg von ihrer Eingabe bis zur Ausgabe durch TeX. Sie beginnen mit Token, die intern in Boxen umgewandelt werden. Diese Boxen werden zu einer größeren `hbox` zusammengefügt, die `hboxen` zu einer, oder mehreren `vbox` und diese wieder zu einer Seite, die natürlich auch wieder eine Box ist. Alles was ihnen noch zum Verständnis fehlt ist das, was die Boxen zusammenhält, der *Leim*, englisch: *Glue*.

5.2 Leim

5.2.1 Eigenschaften von Leim

Neben den Boxen gibt es noch etwas in Texten, die mit TeX gesetzt wurden, und das heißt *Leim*[5]. Die Zeilen dieses Textes haben z.B. einen derartigen Zwischenraum, daß der Abstand der Grundlinien zweier Zeilen immer 12pt beträgt.

Was macht nun diesen Leim aus? Er hat drei Eigenschaften: Eine *natürliches* Ausmaß, ein *Schrumpfmaß* und ein *Streckmaß*. Wenn z.B. kleinere Boxen zu einer größeren horizontalen Box zusammengefaßt werden, dann wird zwischen den kleinen Boxen Leim eingefügt. Um zu verstehen, wie das funktioniert, ein kleines Beispiel:

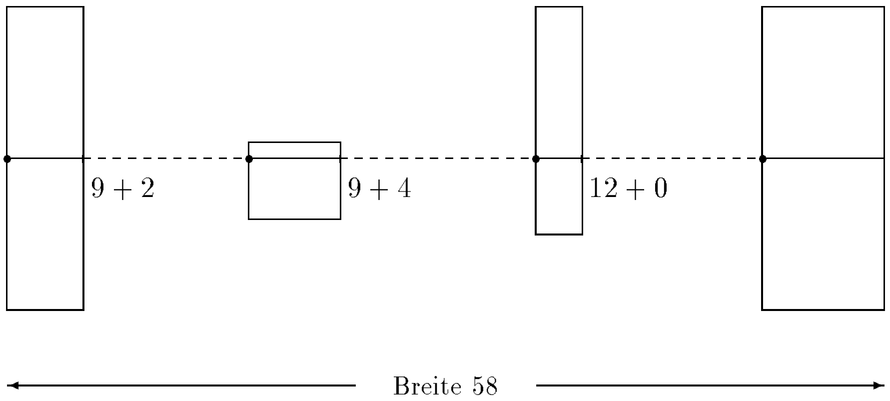


[5] Dieser Leim dürfte Lesern der Bücher von H. Kopka bekannt sein, der Leim heißt dort: *elastische Maße*. Die Wichtigkeit dieser elastischen Maße wird erst hier richtig klar.

Das erste Leimstück hat z.B. eine natürliche Breite von 9pt, eine Schrumpfmöglichkeit von 1pt und eine Streckmöglichkeit von 3pt. Die Gesamtbreite der ganzen, großen Box ergibt sich normalerweise als die Breite der Einzelboxen, plus die natürliche Breite des Leimes, also:

$$5 + 9 + 6 + 9 + 3 + 12 + 8 = 52$$

Angenommen, TEX würde aufgefordert diese Box auf eine Breite von 58pt zu strecken (z.B. wegen der Rechtsbündigkeit), d.h. die Box um 6pt zu strecken. Nun, es gibt insgesamt $3 + 6 + 0 = 9$pt als Gesamtstreckmöglichkeit. TEX multipliziert nun jede Streckmöglichkeit mit $6/9$ um die neuen Leimgrößen zu errechnen. Damit ergibt sich für die erste Lücke: $9 + (6/9) \times 3 = 11$pt und für die zweite Lücke: $9 + (6/9) \times 6 = 13$pt. Die letzte Lücke ist nicht weiter streckbar, so daß sie 12pt breit bleibt. Das Gesamtbild sieht nun folgendermaßen aus:



Wäre TEX, umgekehrt angewiesen worden die Box nur 51pt breit zu machen, dann würde sie erste Lücke um $1/3$, die zweite um $2/3$ ihrer Schrumpfmöglichkeit verkleinert.

Egal wie, der Leim definiert wurde, so hat TEX dennoch nicht die Möglichkeit eine Box beliebig weit zusammenzuschrumpfen. Im vorherigen Beispiel ist daher 49pt die geringste Breite, die die horizontale Box annehmen kann. Hingegen kann der Leim beliebig weit gestreckt werden, wenn man ihm die entsprechenden Angaben beigibt.

Es fragt sich nur, welche Angaben gibt man denn dem Leim? Nun, man sollte die Maße immer so wählen, daß der Text mit dem natürlichen Maß des Leimes am besten aussieht, und die maximale Schrumpf- bzw. Streckmaße so, daß er gerade eben noch gut aussieht. Normalerweise brauchen sie sich allerdings nicht darum zu kümmern, da TEX ihnen diese Arbeit abnimmt. Um z.B. die Abstände zwischen Absätzen zu bestimmen, gibt es in *Plain* TEX den Befehl: '`\smallskip`'. Dieser Befehl wird automatisch immer hinter einem Absatz eingefügt, sie kennen also schon genug Beispiele für die Anwendung. Es ist immer besser einen derartigen Befehl einzusetzen, als den Abstand explizit anzugeben. Nur, wenn es unbedingt sein muß, können sie auf den Leim zurückgreifen. Stellt sich die Frage, *wie* man den nun eine Leimangabe macht. Üblicherweise wird Leim in TEX folgendermaßen spezifiziert:

<dimen> `plus` <dimen> `minus` <dimen>

Die Angaben von `plus` <dimen> und `minus` <dimen> sind optional, und werden als Null angenommen, wenn sie nicht angegeben werden. Die Größe, die hinter `plus` steht repräsentiert sie Streckbarkeit, die hinter `minus` die Schrumpfbarkeit. Die natürliche Größe *muß* angegeben werden, auch wenn sie Null ist. Der Befehl '`\medskip`' wird z.B. in *Plain* TEX folgendermaßen definiert:

```
\vskip6pt plus2pt minus2pt
```

Auch horizontaler Platz wird entsprechend definiert. Der Befehl '`\enskip`' wird z.B. definiert als:

```
\hskip.5em\relax
```

Es wird also ein halbes 'em' Platz gelassen, allerdings ohne die Möglichkeit der Schrumpfung oder Streckung. Der Befehl '`\relax`' hinter der Angabe sorgt dafür, daß der Befehl auch dann richtig arbeitet, wenn im Text hinter dem Befehl eine 'plus' oder 'minus' steht.

Eine wichtige Eigenschaft des Leimes ist, daß er auch *unendlich* weit gestreckt werden kann. Nehmen sie z.B. an, im letzten Beispiel wäre der zweite Leimabschnitt unendlich weit streckbar. Dann würde nur der zweite Zwischenraum gestreckt, wärend die andern unverändert bleiben würden.

Das wird erst interessant, wenn sie daran denken, daß der Befehl '`\centerline`' nichts anderes ist, als eine hbox von der Breite '`\hsize`', deren Text von zwei unendlich streckbaren Leimstücken eingerahmt ist. Auch in der kleinen Geschichte, die wir in einem früheren Kapitel kennengelernt haben wurde derartiger Leim benutzt. Die Geschichte endete mit den Befehlen: '`\vfill`' und '`\eject`'. '`\eject`' sorgt nur für die Ausgabe des Textes an den Ausgabefile, '`\vfill`' hingegen bedeutet: Füge vertikalen Zwischenraum mit der natürlichen Länge 0 ein, der aber unendlich weit streckbar ist. Mit anderen Worten, fülle die restliche Seite mit Leerraum auf.

Es gibt nun in TEX verschiedene Arten von *Unendlichkeit*, die sich voneinander unterscheiden. Bei der vertikalen Streckung gibt es die Befehle: '`\vfil`' und '`\vfill`', wobei der zweite wesentlich stärker wirkt. Wenn '`\vfil`' alleine benutzt wird, dann wird es soweit, wie nötig gestreckt. Tritt es allerdings zusammen mit '`\vfill`' auf, dann wird es nicht gestreckt, sondern es wird da gestreckt, wo '`\vfill`' steht. Denken sie sich das einfach so, als hätte '`\vfil`' eine Streckfähigkeit von einem Kilometer, und '`\vfill`' von einer Million Kilometer.

Analog gibt es auch noch die Befehl '`\hfil`' und '`\hfill`' und die Befehle '`\hss`' und '`\vss`', die sowohl Streckung, als auch unendliche Schrumpfung zulassen.

In diesen Zusammenhang gehört auch noch der Befehl '`\hfilneg`', bzw. '`\vfilneg`'. Diese Befehle heben die Streckbarkeit von '`\hfil`' und '`\vfil`' auf. Den Grund für einen derartigen Befehl werden wir erst später kennenlernen.

Um derartige unendlich streckbare Maße erstelle zu können, können sie auf die Primitive '`fil`', '`fill`' oder '`filll`' zurückgreifen. Die Befehle '`\vfil`', '`\vfill`', '`\vss`' und '`\vfilneg`' werden z.B. definiert als:

```
\vskip 0pt plus 1fil
\vskip 0pt plus 1fill
\vskip 0pt plus 1fil minus 1fil
\vskip 0pt plus -1fil
```

Die dritte Alternative: '`filll`' sollten sie nicht benutzen, auch *Plain* TEX macht keinen Gebrauch davon. Sie ist für absolute Notfälle vorbehalten.

5.2.2 Verwendung des Leims durch TEX

Wie wird der Leim von *Plain* TEX verwendet? Es gibt einige interessante Anwendungen. TEX fügt z.B. hinter einem Satz einen etwas größeren Zwischenraum ein, als zwischen normalen Worten. Außerdem wird die Streckbarkeit des Zwischenraums an diesen Stellen erhöht. Betrachten wir folgenden Satz:[6]

```
``Oh, oh!'' cried Baby Sally. Dick and Jane laughed.
```

In natürlicher Breite sieht der Satz so aus:

“Oh, oh!” cried Baby Sally. Dick and Jane laughed.

[6] Originalbeispiel aus dem TEXBook.

Was aber passiert, wenn man ihn um 5pt, 10pt, 15pt oder mehr auseinanderzieht? Sehen wir es uns an:

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.
"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Der Leim hinter dem Komma wird um das $1\frac{1}{4}$fache des Normalen gestreckt. Hinter dem Punkt und hinter dem '`!''`' sogar um das dreifache. Zwischen zusammengehörigen Zeichen ist kein Leim, so daß die einzelnen Worte ihr Aussehen behalten. Wie siehts nun aus, wenn dieser Satz auf das Minimum zusammengestaucht wird?

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Der Zwischenraum nach dem Komma wird nur 80% dessen, der Zwischenraum nach dem Punkt, bzw. dem Ausrufezeichen sogar nur ein Drittel dessen kleiner, wie der normale Zwischenraum zwischen den Worten.

Für den Normalfall ist das auch so in Ordnung, was aber, wenn ein Punkt überhaupt nicht ein Wortende markiert? Dazu gibt es mehrere Möglichkeiten:

1. Wenn sie drei Punkte schreiben wollen, um die Fortsetzung eines Satzes anzuzeigen . . ., dann sollten sie statt der drei Punkte lieber die Sequenz '`$\ldots$`' schreiben. Damit schalten sie kurzfristig in den mathematischen Modus um und geben dort mit dem Befehl '`\ldots`' die gewünschten Punkte an. Übrigens sieht diese Version auch besser aus. Drei Punkte ergeben nämlich normalerweise, ohne die Umschaltung in den mathematischen Modus: '...'.

2. Punkte hinter Abkürzungen beenden üblicherweise auch keinen Satz. Hier gibt es zwei Möglichkeiten. Entweder, sie ersetzen das Leerzeichen hinter dem Punkt durch die Tilde: '`~`', oder sie setzen ein Kontrollleerzeichen: '`\␣`', je nachdem, ob sie eine Trennung an dieser Stelle zulassen wollen oder nicht. Die Tilde wird durch ein Leerzeichen ersetzt, an dem aber kein Zeilenumbruch stattfinden kann.

3. Sie können sich diese Überlegungen sparen, wenn der Punkt hinter einem Großbuchstaben steht. TEX nimmt dann sowieso an, daß es sich um eine Abkürzung handelt und behandelt den Punkt nicht wie einen Satz-Ende-Punkt.

Letztlich können sie alle Überlegungen zu Zwischenräumen vergessen, wenn sie zu Beginn ihres Textes den Befehl: '`\frenchspacing`' schreiben. Mit diesem Befehl bewirken sie, daß alle Zwischenräume gleich groß sind, egal, wie und wo sie stehen. Unser Beispiel sähe damit folgendermaßen aus:

"Oh, oh!" cried Baby Sally. Dick and Jane laughed.

Mit '`\nonfrenchspacing`' wird wieder zurückgeschaltet. Angenommen, wir hätten den Befehl gegeben, normal mit Zwischenraum umzugehen, dann könnten wir uns den Leim ansehen, indem wir den Satz in eine Box schreiben und diese dann ansehen:

```
.\tenrm \ (ligature ‘‘)
.\tenrm O
.\tenrm h
.\tenrm ,
.\glue 3.33333 plus 2.08331 minus 0.88889
.\tenrm o
.\tenrm h
.\tenrm !
.\tenrm \" (ligature ’’)
.\glue 4.44444 plus 4.99997 minus 0.37036
```

```
.\tenrm c
.\tenrm r
.\tenrm i
.\tenrm e
.\tenrm d
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm B
.\tenrm a
.\tenrm b
.\kern-0.27779
.\tenrm y
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm S
.\tenrm a
.\tenrm l
.\tenrm l
.\tenrm y
.\kern-0.83334
.\tenrm .
.\glue 4.44444 plus 4.99997 minus 0.37036
```

Normalerweise hat der Leim bei der Schriftart '`\tenrm`' die Werte: 3.33333pt plus 1.66666 pt minus 1.11111 pt, aber achten sie darauf, wie sich bei der Interpunktion der Leim ändert. Sie können an diesem Beispiel noch mehr sehen. Zum einen wandelt TEX die beiden '' ' ' '' in eine Ligatur (also *ein* Zeichen) und ausserdem werden noch einige '`\kern`' Befehle eingefügt. '`\kern`' verhät sich ähnlich wie Leim, nur das er weder streck- noch schrumpfbar ist und es wird hinter einem '`\kern`' Befehl *nie* ein Zeilenumbruch durchgeführt, es sei denn es folgt unmittelbar ein Leimbefehl.

5.2.3 Die genauen Regeln für den Leim

Ihnen dürfte eventuell schon aufgefallen sein, daß die oben beschriebenen Regeln nicht *hunderprozentig* richtig sind. So würden die obigen Regeln nicht erklären, wieso das Ausrufezeichen richtig behandelt wird, obwohl noch die beiden Anführungsstriche folgen.

Es gibt einen weiteren wichtigen Wert, den Zwischenraumfaktor (f). Dieser ist üblicherweise 1000. Wenn der Faktor größer als 2000 wird, wird der Zwischenraum folgendermaßen berechnet: Addiere zum normalen Zwischenraum den sog. Extrazwischenraum[7] und multipliziere den Streckwert mit $f/1000$, den Schrumpfwert mit $1000/f$.

Es gibt nun für sie einfache Möglichkeiten den normalen Zwischenraum zu überschreiben. Mit den Befehlen: '`\spaceskip`' und '`\xspaceskip`' wird folgendermaßen verfahren: Ist der Zwischenraumfaktor (f) größer als 2000, dann wird entweder der Leimwert von '`\xspaceskip`' genommen, wenn dieser ungleich Null ist, oder der von '`\spaceskip`', wobei die Streck- und Schrumpfkomponenten mit $f/1000$ bzw. $1000/f$ multipliziert werden. Das '`\raggedright`' Makro z.B. benutzt die beiden Befehle, um jedes Strecken oder Schrumpfen zu verhindern.

Wenn eine horizontale Liste begonnen wird, wird der Zwischenraumfaktor auf 1000 gesetzt. Ebenso nach jedem Nichtbuchstaben oder einer mathematischen Formel. Jedes andere Zeichen, was an die Liste angefügt wird, hat einen eigenen Zwischenraumfaktor, der dann den alten Wert ersetzt. Dabei gibt es allerdings noch zwei Regeln:

1. Ist der Faktor des Zeichens 0, dann wird der alte Wert beibehalten,

[7] Zu jedem Zeichensatz sind die folgenden Angaben vorgegeben: Normaler Zwischenraum, normaler Streckwert, normaler Schrumpfwert und eben dieser Extrazwischenraum. Für `cmr10` sind die Werte: 3.33333pt, 1.66666pt, 1.11111pt und 1.11111pt

2. Ist der Wert von 1000 verschieden, und es gilt z.B.: $f < 1000 < g$ (g der Faktor des Zeichens), dann erhält f den Wert 1000. Mit andern Worten, f *überspringt* nicht die 1000er Grenze.

Der größtmögliche Zwischenraumfaktor ist 32767, und damit größer, als jemals nötig.

Wenn `INITEX` seine Arbeit aufnimmt, haben alle Zeichen den Zwischenraumfaktor 1000, außer den Großbuchstaben, die den Wert 999 haben.[8] *Plain* TEX ändert nun einige Werte mit dem Primitiv: '`\sfcode`', der ähnlich dem '`\catcode`' Befehl arbeitet.

```
\sfcode`)=0   \sfcode`.=3000
```

ändert den Faktor für die schließende Klammer so, daß er ignoriert wird, für den Punkt so, daß zusätzlicher Zwischenraum eingefügt wird.

Bei Ligaturen berechnet sich der Zwischenraumfaktor aus den Einzelkomponenten, aus denen die Ligatur besteht.

Nun zu einem Beispiel, wie TEX *wirklich* eine Zeile zusammensetzt: Die natürliche Breite einer Box setzt sich zusammen aus allen Zeichenboxbreiten plus den natürlichen Breiten der Leimstücke. Ist die natürliche Breite gleich der gewünschten Breite, passiert nichts. Ist hingegen die gewünschte Breite kleiner oder größerr, als die natürliche Breite, dann wird ein Faktor berechnet, mit dem jeder Schrumpf- bzw. Streckfaktor der Leimstücke multipliziert wird. Das Ergebnis ist eine Box, die genau die gewünschte Breite hat.

Um einer `hbox` eine feste Breite zu geben benutzt man den Befehl:

```
\hbox to <dimen>{ ... }
```

Das Makro '`\centerline`' beginnt z.B.: '`\hbox to\hsize`'. Ähnlich arbeitet die Angabe von:

```
\hbox spread <dimen>{ ... }
```

Die angegebene Dimension besagt, um wieviel größer, als die natürliche Breite (<dimen>> 0), bzw. um wieviel kleiner als die natürliche Größe (<dimen>< 0) die `hbox` werden soll.

5.3 Die Feinheiten einer vertikalen Box

Die Grundlinie einer horizontalen Box ist gleich der gemeinsamen Grundlinie aller Elemente, soweit diese nicht angehoben oder abgesenkt wurden. Die Höhe einer derartigen Box ist gleich der Höhe des höchsten Elements, die Tiefe gleich der tiefsten Einzeltiefe. Somit kann eine horizontale Box nur positive Höhe und Tiefe haben, wohl aber eine negative Breite.

Wenn nun horizontale Boxen zu einer vertikalen Box zusammengefügt werden sollen, dann funktioniert das ähnlich, wie bei den horizontalen Boxen, mit einem Unterschied. Die horizontalen Boxen sollen sich nicht berühren. Dazu gibt es drei Werte, die vorher festgelegt werden müssen:

```
\baselineskip=<Leim>
\lineskip=<Leim>
\lineskiplimit=<dimen>
```

Wird nun eine neue horizonatale Box an eine vertikale Liste angefügt, dann wird der Zeilenabstand so berechnet, daß der Abstand der Basislinie der neuen Box, von der Basislinie der letzten Box genau den Wert von '`\baselineskip`' hat. Ist das Ergebnis so, daß die Zeilen mit dieser Lösung zu nahe aneinanderliegen, genauer, näher als der Wert in '`\lineskiplimit`', dann wird statt dessen der Wert '`\lineskip`' benutzt.

Als Ausnahme von dieser Regel dienen nur die Befehle: '`\vskip`' und '`\kern`', und alle daraus zusammengesetzten Befehle. Hiermit ist es möglich einen anderen, als den normalen Zeilenabstand zu erzeugen. Z.B. der Befehl '`\smallskip`' arbeitet so. Eine weitere Ausnahme besteht darin, daß

[8] Erraten sie, warum Punkte hinter Großbuchstaben anders behandelt werden?

hinter '`\xrule`'-Befehlen kein Zeilenzwischenraum eingefügt wird. Sie können den Zwischenraum aber auch mit dem Befehl: '`\nointerlineskip`' selber verhindern.

Als Regel sollte sie sich folgendes angewöhnen: Wenn ihr Text über mehrere Zeilen geht, dann sollte '`\baselineskip`' *keine* Schrumpf- oder Streckangaben enthalten, um so das Aussehen der Seiten gleichmäßiger zu gestalten. Bei Texten, die nur über eine Seite gehen, ist dagegen die Angabe dieser Werte sinvoll, um eine bessere Ausnutzung der Seite zu erhalten.

Zur Berechnung des Zeilenzwischenraums greift TEX übrigens auf ein Primitiv mit Namen: '`\prevdepth`' zurück. Normalerweise die Tiefe der letzten horizontalen Box, die angefügt wurde. Der Wert wird allerdings auf -1000 gesetzt, am Anfang einer neuen vertikalen Liste, oder nach einem '`\rule`'-Befehl. Sie können den Wert aber auch *von Hand* ändern, wenn sie es wollen.

Schließlich noch die exakten Regeln, nach denen eine neue horizontale Box an eine vertikale Box angehängt wird: Angenommen die neu Box hat eine Höhe von h, und der Wert von '`\prevdepth`' ist p. Weiterhin sollen sein:

```
\baselineskip=b plus y minus z
\lineskiplimit=l
```

Wenn $p \leq -1000$pt wird kein Zeilenzwischenraum eingefügt. Ist andererseits: $b - p - h \geq l$ dann wird der Zwischenraum $(b - p - h)$ plus x minus y) eingefügt, ansonsten `\lineskip`.

Für die vertikalen Boxen gelten dieselben Befehle, wie für die horizontalen Boxen, mit dem Unterschied, daß der Befehl: '`\vbox`' verwendet wird. Es sind natürlich auch die Befehle: '`\vbox to ...`' und '`\vbox spread ...`' erlaubt.

Der tiefste vorkommende Referenzpunkt innerhalb einer vertikalen Box wird als Referenzpunkt für die ganze vertikale Box genommen. Normalerweise werden vertikale Boxen so untereinander gesetzt, daß ihre Referenzpunkte senkrecht untereinander liegen, mit den Befehlen:

```
\moveright<dimen><box>
\moveleft<dimen><box>
```

können sie aber auch nach rechts oder links verschoben werden. Dies entspricht den Befehlen '`\raise`' und '`\lower`' in horizontalen Boxen.

5.4 Sonderfälle

Neben dem Befehl für eine vertikale `vbox` gibt es auch den Befehl: '`\vtop`'. Dieser erzeugt auch eine vertikale Box, und die zugehörigen Befehle arbeiten komplett analog zu denen für '`\vbox`'. Der Unterschied besteht darin, daß die Basislinie der *obersten* enthaltenen horizontalen Box als Referenzpunkt genommen wird. Damit sind Konstruktionen wie die folgende möglich:

```
\hbox{Hier sind \vtop{\hbox{zwei Zeilen}\hbox{mit Text}}}
```

Das ergibt:

Hier sind zwei Zeilen
mit Text

Ein weiteres Element bei der Arbeit mit vertikalen Boxen stellt der Befehl '`\strut`' dar. Mit ihm wird eine Box der Höhe 8.5pt, der Tiefe 3.5pt und der Breite Null bereitgestellt. Sie können diesen Befehl benutzen, um immer korrekte Abstände zu erzwingen. Schauen sie sich z.B. im Anhang das '`\footnote`' Makro an, das dafür sorgt, daß mehrere Fußnoten auch voneinander abgesetzt werden.

Letztlich sind noch die Befehle '`\rlap`' und '`\llap`' interessant. Sie sorgen dafür, daß der Text, auf den sich die Befehle beziehen folgendermaßen verhält. Das Makro '`\rlap`' gibt den zugehörigen Text aus, nimmt aber keinen Platz weg. Genauer, es wird eine Box der Breite Null erzeugt, an den sich der zugehörige Text anschließt. Man kann das auch so verstehen, als würde der Text ausgegeben, und dann um die Breite des Textes zurückgegangen. Somit ist das Zeichen: '$\neq$' auch produzierbar mit dem Befehl: '`\rlap=/`'. Analog arbeitet der Befehl '`\llap`' nur, daß hier der Text sozusagen *nach links* ausgegeben wird. Das obige Zeichen ließe sich also auch mit. '`/\llap=`' erzeugen.

Kapitel 6

Zusammensetzung der Seiten

6.1 Modi

TEX kennt sechs verschiedene Bearbeitungsmodi:

- Den *vertikalen Modus*, in dem die äusserste vertikale Liste bearbeitet wird, die zur Ausgabe der Seiten führt.
- Den *internen vertikalen Modus* zum Aufbau von '`\vbox`'en.
- Den *horizontalen Modus* um aus Zeilen Paragraphen aufzubauen.
- Den *eingeschränkten horizontalen Modus* zum Aufbau von '`\hbox`'en.
- Den *mathematischen Modus* mit dem mathematische Formeln in eine horizontale Liste eingefügt werden.
- Den *abgesetzten mathematischen Modus*, mit dem Formeln in eine eigene Zeile gesetzt werden, die den momentanen Paragraphen unterbricht.

Normalerweise braucht sie der Modus, in dem TEX sich gerade befindet nicht zu interessieren, höchstens bei der Fehlersuche kann es hilfreich sein die verschiedenen Modi zu kennen. Grundsätzlich reicht es zu wissen, daß es die drei Grundtypen: vertikaler, horizontaler und mathematischer Modus gibt. Manche Befehle haben, je nach momentanem Modus unterschiedliche Wirkung. Der Befehl '`\kern`' bewirkt z.B. einen vertikalen resp. horizontalen Zwischenraum, je nach Modus, in dem er angetroffen wird.

Betrachten wir mal wieder unser Lieblingsbeispiel: die kleine Story. Zu Bearbeitungsbeginn befindet sich TEX immer im vertikalen Modus. Die Abstandsanweisung, sowie der horizontale Strich, werden einfach an die vertikale Liste angefügt. Auch die Box, die von '`\centerline`' erzeugt wird, wird einfach an die Liste angehängt, sie erfordert aber etwas mehr Vorarbeit. Die Box, die '`A SHORT STORY`' enthält wird im eingeschränkten horizontalen Modus erstellt. Die beiden folgenden Paragraphen werden dann im normalen horizontalen Modus erstellt.

Wenn sich TEX im vertikalen Modus befindet, sorgt jeder auftretende Buchstabe für ein Umschalten in den horizontalen Modus. Das gilt aber auch für die Befehle: '`\char`', '`\accent`', '`\hskip`', '`\␣`' und '`\vrule`' oder die Umschaltung in den mathematischen Modus ($). Ausserdem läßt sich mit den Befehlen: '`\indent`' oder '`\noindent`' in den horizontalen Modus umschalten. Diese Befehle werden zu Beginn eines Paragraphen gebraucht um eine Einrückung der Größe '`\parindent`' zu bewirken, oder eben nicht.

Der horiontale Modus wird beendet durch:

1. Zwei Leerzeilen,
2. den Befehl: '`\par`'

3. einen inkompatibelen Befehl wie z.B.: '`\vskip`', der im horizontalen Modus keinen Sinn ergäbe.

Der mathematische Modus wird durch ein Dollarzeichen ($) eingeleitet und solange beibehalten, bis wieder ein schließendes Dollarzeichen gefunden wird. Die Dollarzeichen wirken also wie eine Klammer.

Mit einem doppelten Dollarzeichen wird in den abgesetzten mathematischen Modus geschaltet ($$). Der laufende Paragraph wird unterbrochen, die mathematische Formel in einer eigenen Zeile gesetzt und nach dem schließenden doppelten Dollarzeichen wird der Paragraph wieder fortgesetzt. So ergibt:

```
Die Nummer $$\pi \approx 3.1415926536$$ ist wichtig.
```

Die Nummer

$$\pi \approx 3.1415926536$$

ist wichtig.

Im vertikalen Modus werden Leerzeichen und -zeilen ignoriert. Eine Ausnahme bildet nur das kontrollierte Leerzeichen ('`\␣`'), das einen neuen Paragraphen einleitet, nachdem zuvor die Einrückung vorgenommen wurde.

Um einen Text zu beenden sollte man am besten den Befehl: '`\bye`' verwenden, der eine Abkürzung für die Befehlsfolge: '`\vfill`' (auffüllen der letzten Seite mit Leerplatz), '`\eject`' (Ausgabe der Seite) und '`\end`' (Beendigung der Arbeit von TEX) darstellt.

Der interne vertikale Modus unterscheidet sich nur geringfügig vom normalen vertikalen Modus und der eingeschränkte horizontale Modus noch weniger vom normalen horizontalen Modus. Dennoch gibt es kleine Unterschiede, da die Zielsetzung ein wenig anders ist.

TEX schaltet häufig zwischen den Modi hin und her. Dabei ist der äusserste Modus immer der vertikale Modus. Wird er unterbrochen, dann wird nach der Unterbrechung der vertikale Modus wiederaufgenommen. Alle Modi können getestet werden, wenn man folgenden File eingibt:

```
\tracingcommands=1
\hbox{
$
\vbox{
\noindent$$
x\showlists
$$}$}\bye
```

Der erste Befehl in diesem File bewirkt, daß alle weiteren Aktionen genauer mitprotokolliert werden. Der Logfile zu dieser Datei sieht damit folgendermaßen aus:

```
{vertical mode: \hbox}
{restricted horizontal mode: blank space}
{math shift character $}
{math mode: blank space}
{\vbox}
{internal vertical mode: blank space}
{\noindent}
{horizontal mode: math shift character $}
{display math mode: blank space}
{the letter x}
```

Versuchen sie diese Ausgabe selber zu verstehen, so schwierig, wie es auf den ersten Blick aussieht ist es gar nicht. Denken sie nur daran, daß TEX die Zeilenendemarkierung in ein Leerzeichen umwandelt. Der Befehl '`\showlists`' soll hier zunächst ignoriert werden.

6.2 Wie macht TEX aus Zeilen Paragraphen?

6.2.1 Das Vorgehen und Sonderfälle

Grundsätzlich werden Zeilenumbrüche innerhalb eines Paragraphen immer für den ganzen Paragraphen berechnet. Die Vorgehensweise ist dabei so, daß die Umbrüche gesucht werden, bei denen die *badness* für den ganzen Paragraphen am geringsten ist. TEX kann natürlich nicht für *psychologisch* schlechte Umbrüche verantwortlich gemacht werden. In ganz schlimmen Fällen[1] muß man wohl per Hand nachhelfen. Es gibt aber für viele Fälle eine einfache Möglichkeit Zeilenumbrüche zu verhindern: Die Tilde ('~').

Die Tilde ersetzt ein Leerzeichen, TEX setzt an die Stelle auch ein Leerzeichen, bricht aber an dieser Stelle die Zeile nicht um. Man sollte die Tilde z.B. in folgenden Fällen benutzen:

1. Referenzen in Texten:

```
Kapitel~1          Satz~12
Anhang~A           Figur~13
```

2. Zwischen Vornamen und zwischen den Teilen von Nachnamen:

```
Donald~E. Knuth
Vincent van~Gogh
Fritz~der~Gro\ss{}e
```

3. Zwischen mathematischen Symbolen und ihren Bezeichnern:

```
Breite~$b$         H\"ohe~$h$
```

4. Zwischen Symbolen in Serie:

```
1,~2 oder~3
1,~2 $\ldots$~$n$
```

Auch '`\hboxen`' werden von TEX nicht gebrochen. Wenn sie z.B. einen Seitenbereich angeben wollen, dann schreiben sie am besten: '`Seiten \hbox{1--3}`'. Seien sie aber bitte mit dieser Variante vorsichtig. Es ist z.B. besser '`Kapitel~12`', als '`\hbox{Kapitel 12}`' zu schreiben, da im ersten Falle das Leerzeichen noch gestreckt oder geschrumpft werden kann und außerdem das Wort '`Kapitel`' noch getrennt werden kann.

Einen Zeilenumbruch an einer bestimmten Stelle erzwingen sie mit dem Befehl '`\break`'. Erkann allerdings dazu führen,
daß ihre Zeile etwas in die Länge gezogen wird. Wenn sie wollen, daß TEX zunächst die Zeile mit leerem Platz auffüllt,
dann benutzen sie den Befehl '`\hfil\break`' um genau das zu erreichen.

Mitunter, besonders in Gedichten wollen sie vielleicht, daß die Zeilen immer da umgebrochen werden, wo sie auch in der Eingabe umgebrochen sind. Sie können das erreichen, indem sie hinter jede Zeile ein '`\par`' schreiben, einfacher ist es allerdings, wenn sie innerhalb einer Gruppe den Befehl '`\obeylines`' benutzen. Auch hier wieder ein Beispiel aus dem Original:

```
{\obeylines\smallskip
Roses are red,
\quad Violets are blue;
Rhymes can be typeset
\quad With boxes and glue.
\smallskip}
```

[1] Ich erinnere hier immer wieder gerne an das Beispiel aus der *Context* Dokumentation: Man sollte Urinstinkt nicht zu: Urin – stinkt trennen.

6.2.2 Wie gehts genau?

Die horizontale Liste

Wie macht TEX es denn nun, wenn es einen Paragraphen in Zeilen bricht? Zunächst ist ein Paragraph für TEX nur eine lange Reihe von Wörtern. Genauer gesagt nicht aus Wörtern, sondern aus einer Reihe von Elementen, die folgendes sein können:

1. Ein Box (Also ein Zeichen, eine Ligatur, eine Linie, eine hbox oder eine vbox).
2. Eine mögliche Trennung (wird gleich erklärt).
3. Ein “WasDenn” (kommt später).
4. Vertikales Material (von ‘`\mark`’, ‘`\vadjust`’ oder ‘`\insert`’).
5. Etwas Leim (oder ‘`\leaders`’, wie auch später noch erklärt wird).
6. Ein Kern (so was wie Leim, der aber nicht gestreckt oder geschrumpft werden kann).
7. Strafpunkte (die anzeigen, ob hier ein Zeilenumbruch günstig oder ungünstig ist).
8. Ein Zeichen, daß den Beginn oder das Ende des mathematischen Modus anzeigt.

Die letzten vier Möglichkeiten heißen übrigens *entfernbar*, da sie bei einem Zeilenumbruch verändert werden können, oder sogar ganz wegfallen.

Die mögliche Trennung

Immer, wenn ein Wort auf eine besondere Art getrennt werden soll, dann gibt es in TEX die Möglichkeit diese Trennung gesondert anzugeben. Besonders in der deutschen Sprache ist diese Möglichkeit wichtig, da im Deutschen höchst seltsame Trennregeln existieren. Aber auch bei ungewöhnlichen Trennungen (z.B. Trennung von ‘st’), kann dieser Befehl benutzt werden. Die allgemeine Form lautet dabei folgendermaßen:

```
\discretionary{<Vor-Bruch Text>}{<Nach-Bruch Text>}{<Nicht-Bruch Text>}
```

Im Deutschen wird z.B. jedes ‘ck’ in ‘k-k’ getrennt. Sollte es einmal wichtig werden, dann schreiben sie doch einfach:

```
dr\"u\discretionary{k-}{k}{ck}en
```

wenn sie das Wort ‘drücken’ richtig trennen wollen. So kompliziert braucht es aber nur in solchen Sonderfällen zu sein. Für die mögliche Trennung:

```
\discretionary{-}{}{}
```

gibt es auch die Abkürzung: ‘`\-`’ TEX macht übrigens eigentlich genau das Gleiche, wenn es Worte trennen will. Es werden einfach in ein Wort derartige mögliche Trennungen eingefügt. Aus dem Wort ‘`Trennung`’ wird z.B. ‘`Tren\-nung`’.

Die Berechnung der Trennungen

Zunächst versucht TEX einen Paragraphen *ohne* Trennungen zu setzen. Die dabei auftretenden Streckungen und Stauchungen müssen dabei unter dem Wert von '`\pretolerance`' bleiben. Erst im zweiten Durchlauf, sollte er denn nötig sein, werden Wörter getrennt und dann müssen die Werte von badness unter '`\tolerance`' bleiben. Voreingestellt sind übrigens:

```
\pretolerance=100
\tolerance=200
```

Zunächst einmal: Wo darf überhaupt getrennt werden? Es darf an den folgenden Stellen im Text getrennt werden:

1. Bei Leim, sofern dieser Leim nicht unmittelbar hinter einem entfernbaren Element steht und er nicht in einer mathematischen Formel auftritt. Gebrochen wird am linken Rand des Leimstücks.

2. Bei Kern, vorausgesetzt, daß diesem Kern etwas Leim folgt und er nicht in einer mathematischen Formel steht.

3. An einem Mathe-Ende-Zeichen, dem sofort Leim folgt.

4. Bei Strafpunkten, die eventuell automatisch eingefügt wurden.

5. Bei einer möglichen Trennung.

Zu jeder möglichen Trennung gehört eine *ästhetische Strafe*. In den Fällen: 1, 2 und 3 ist die Strafe 0, im vierten Fall ist die Strafe exakt angegeben, im fünften Fall ist es etwas komplizierter. Ist der Text vor der Trennung nicht-leer, dann wird der Wert von '`\hyphenpenalty`' (voreingestellt mit 50), ist er leer der Wert von '`\exhyphenpenalty`' (voreingestellt mit 50) verwendet.

Wenn sie explizit an einer Stelle in ihrem Text angeben: '`\penalty 100`', dann kann dort zwar noch gebrochen werden, es tritt aber eine Strafe von 100 auf[2]. Wenn sie hingegen '`\penalty-100`' schreiben, dann geben sie damit an, daß an dieser Stelle besonders gut gebrochen werden kann, da *negative* Strafpunkte die Trennung fördern. Eine Strafe von 10000 ist z.B. so hoch, daß TEX hier nie eine Zeile umbrechen wird, eine Strafe von -10000 so niedrig, daß TEX hier immer die Zeile umbrechen wird. Der Befehl '`\nobreak`' ist nur eine Abkürzung für '`\penalty 10000`' und die Tilde wird einfach als: '`\nobreak\␣`' realsiert.

Nach dem Zeilenumbruch werden alle entfernbaren Elemente entfernt und zwar bis zum ersten nicht-entfernbaren Element, oder bis zum nächsten Zeilenumbruch.

Die genaue Berechnung der badness einer Zeile braucht hier nicht zu interessieren, sie ist ungefähr das Hundertfache der dritten Potenz der Streckung oder Schrumpfung der Zeile. Wenn eine Zeile z.B. 10pt zu Streckung zur Verfügung hat, die Zeile tatsächlich um 9pt gestreckt werden muß, um auf das notwendige Maß zu kommen, dann ist die badness: $100 \times (9/10)^3 \approx 73$.

Zeilen werden nun entsprechend ihrer badness klassifiziert. Ist die badness kleiner als 13, dann gilt die Zeile als *anständig*, liegt sie zwischen 13 und 99, dann heißt sie *dicht*, wenn sie geschrumpft wurde, *lose*, wenn sie gestreckt wurde. Ist die badness größer als 100, dann heißt die Zeile *sehr dicht* oder *sehr lose*. Zwei Zeilen sind optisch inkompatibel, wenn mehr als ein Bereich übersprungen wird, wenn z.B. eine anständige Zeile vor einer sehr losen kommt.

TEX berechnet nun für eine Folge von Zeilenumbrüchen die Minuspunkte des Paragraphen, indem es die Minuspunkte aller Zeilen aufaddiert. Die Minuspunkte einer Zeile ergeben sich dabei nach folgender Formel (dabei ist: d die Minuspunkte, p die Strafpunkte, b die badness, l die sogenannte Zeilenstrafe, die von TEX üblicherweise mit 10 voreingestellt ist ('`\linepenalty`'). Sie kann erhöht

---

[2] Penalty=Strafpunkte

werden, wenn man erreichen will, daß die Paragraphen mit möglichst geringer Zeilenzahl gesetzt werden sollen):

$$d = \begin{cases} (l+b)^2 + p^2 & \text{wenn} \quad 0 \leq p < 10000 \\ (l+b)^2 - p^2 & \text{wenn} \quad -10000 < p < 0 \\ (l+b)^2 & \text{wenn} \quad p \leq -10000 \end{cases}$$

Sind zwei aufeinanderfolgende Zeilen optisch inkompatibel, dann wird außerdem noch der Wert von '`\adjdemerits`' hinzugefügt, enden zwei aufeinanderfolgende Zeilen mit einer Trennung, dann wird der Wert von '`\doublehyphendemerits`' addiert und endet die vorletzte Zeile eines Paragraphen mit einer Trennung, dann wird noch '`\finalhyphendemerits`' addiert. Dabei sind die folgenden Werte voreingestellt:

```
\adjdemerits=10000
\doublehyphendemerits=10000
\finalhyphendemerits=5000
```

Die genaue Berechnung der einzelnen Durchläufe läßt sich verfolgen, wenn '`\tracingparagraphs=1`' gesetzt wird, dessen Erklärung aber hier zu weit führen würde.

Die letzte Zeile

Bisher dürfte noch nicht klar sein, wieso die letzte Zeile eines Paragraphen kürzer sein kann, als die letzten. Die Lösung ist aber ganz einfach: Bevor TEX anfängt die besten Zeilenumbrüche zu suchen unternimmt es noch zwei wichtige Dinge: (1) Steht am Ende eines Paragraphen ein Stück Leim, so wird dieser entfernt. (2) Werden drei Befehle an den Paragraphen angehängt: (a) '`\penalty10000`', was einen Zeilenumbruch vor den letzten Befehlen verhindert, (b) '`\hskip\parfillskip`', was den leeren Platz am Ende der Zeile bewirkt, '`\parfillskip`' ist dabei üblicherweise als `0pt plus 1fil` definiert und (c) der Befehl '`\penalty-10000`', was den letzten Zeilenumbruch bewirkt. Mit dem Wert für '`\parfillskip`' kann übrigens auch gespielt werden. Diese Absatz wurde mit dem Wert `0pt` gesetzt, so daß am Ende des Abschnitts kein Platz gelassen wird. Das ist allerdings nur bei längeren Abschnitten problemlos möglich.

Lange Paragraphen

Was tun, wenn ein Paragraph *wirklich* lang wird? TEX hält immer den ganzen Paragraphen im Speicher, und daher kann es vorkommen, daß lange Paragraphen den Speicherplatz, der zur Verfügung steht, sprengen. Man kann das umgehen, indem man in einen Paragraphen die Befehlsfolge:

```
{\parfillskip0pt\par\parskip0pt\noindent}
```

einfügt. Auf die Art wird der Paragraph in kleinere Paragraphen unterteilt, ohne daß der Leser es merkt.

6.2.3 Feinheiten und Tricks

Einrückungen und Flattersatz

> Bei dem Satz eines Paragraphen werden außer den üblichen Größen auch noch die Werte von '`\rightskip`' und '`\leftskip`' berücksichtigt. Es handelt sich dabei um Leimstücke, die vor, bzw. nach jeder Zeile eingefügt werden. Normalerweise sind die Werte einfach Null, aber z.B. das Makro '`\narrower`' erhöht die beiden Werte auf den Wert von '`\parindent`', so daß alle Zeilen rechts und links eingerückt werden. Dieser Absatz wurde mit dem Makro gesetzt.

Auch das Makro '`\raggedright`' wird mit '`\rightskip`' realisiert, aber nicht einfach mit:

```
\rightskip 0pt plus 1fil
```

wie man vielleicht annehmen könnte, da diese Lösung dazu führen würde, daß auch ganz kurze Zeilen noch für TEX akzeptabel wären. Statt dessen sollte man die Wortabstände konstant wählen und '`\rightskip`' zwar hoch, aber auch nicht *zu* hoch setzen.

Totale Kontrolle

Es gibt schließlich noch eine *sehr* eindrucksvolle Möglichkeit die Länge von Zeilen im Text zu beeinflußen. Den Befehl '`\parshape`'. Ihm folgen zunächst eine einfache Zahl, und dann Paare von Werte, die die folgende Bedeutung haben: Die erste Zahl gibt an, auf wieviele Zeilen sich der Befehl bezieht (in diesem Paragraphen sind es 16 Zeilen). Bei den folgenden Wertepaaren gibt der erste Wert an, wieviel die Zeile vom linken Rand her eingerückt werden soll, und der zweite Wert, wie lang die jeweilige Zeile sein soll. In diesem Paragraphen mußten also 16 Wertepaare angegeben werden. Ist der Paragraph kürzer, als die angegebene Anzahl von Zeilen, dann werden die überflüßigen Angaben einfach ignoriert, ist er länger, dann gilt die Angabe für die letzte Zeile. Haben sie einen Paragraphen geschrieben, und wissen nicht, ob ihre Angaben nicht eventuell mit in den nächsten Abschnitt hineinragen, dann können sie die letzte Angabe einfach dadurch unwirksam machen, daß sie '`\parshape 0`' eingeben. So schalten sie wieder auf die ursprünglichen Werte von Zeilenlänge. Eine wichtige Abkürzung für den Befehl '`\parshape`' stellt TEX mit den Befehlen '`\hangindent`' und '`\hangafter`' bereit. Diese werden viel öfters gebraucht, als der eigentliche Befehl '`\parshape`', der nur in wirklich seltenen Fällen zur Anwendung kommt.

Mit den Befehlen '`\hangindent`' und '`\hangafter`' können an den vier *Ecken* eines Absatzes rechteckige Flächen freigelassen werden. '`\hangafter`' dient dabei der Angabe, wieviele Zeilen von der Einrückung betroffen sind, und zwar sind die Zeilen $n+1$, $n+2\ldots$ betroffen, wenn n ein positiver Wert von '`\hangafter`' ist, ist n negativ, dann sind die Zeilen 1, 2, $|n|$ betroffen. Mit '`\hangindent`' wird nun angegeben, wie groß die Einrückung ist. Bei positivem Wert, wird links, andernfalls rechts eingerückt. Dieser Paragraph wurde mit:

```
\hangafter=-3
\hangindent=\parindent
```

gesetzt. Die beiden Befehle werden übrigens für die Makros '`\item`' und '`\itemitem`' benutzt, die für Aufzählungen zuständig sind. Wenn sie z.B. schreiben:

```
\item{1.} Dies ist die erste Aufz\"ahlungsebene
\itemitem{(a)} Hier nun die Unterebene,
\itemitem{(b)} und ein weiterer Eintrag in die Unterebene
\item{2.} Und zum Schlu\ss{} noch ein Eintrag in der h\"oheren Eintragebene,
um das Ganze etwas interessanter zu gestalten.
```

dann erhalten sie:

1. Dies ist die erste Aufzählungsebene
    (a) Hier nun die Unterebene,
    (b) und ein weiterer Eintrag in die Unterebene
2. Und zum Schluß noch ein Eintrag in der höheren Eintragebene, um das Ganze etwas interessanter zu gestalten.

Die Werte von '`\parshape`', '`\hangindent`' und '`\hangafter`' werden übrigens am Ende eines Paragraphen zurückgesetzt.

Die Höhe von Paragraphen

Eine abgesetzte Formel in einem Paragraphen nimmt den Platz von drei Zeilen ein. Wenn ihr Paragraph also vier Zeilen vor und zwei Zeilen nach der Formel hat, dann ist der gesamte Paragraph $4 + 3 + 2 = 9$ Zeilen hoch. Intern wird die Anzahl der Zeilen, die schon bearbeitet wurden in der Variablen '`\prefgraf`' festgehalten. Dies können sie dazu ausnutzen, daß sie den Wert bei höheren Formeln ändern. Ist ihre Formel z.B. sehr hoch, dann könnten sie in dem vorigen Beispiel vor den beiden letzten Zeilen den Wert von '`\prefgraf`' auf 8 setzen, so daß TEX denkt, es hätte insgesamt einen Paragraphen mit 10 Zeilen gesetzt.

Ein weiteres Mittel gibt es um die Höhe von Paragraphen zu beeinflußen. Dazu verwenden sie den Befehl '`\looseness`'. Setzen sie diesen Wert z.B. auf 1, dann macht TEX den Paragraphen um eine Zeile länger, als es ihn normalerweise machen würde. Mit negativen Werten kann ein Paragraph auch kürzer gemacht werden. Der Wert von '`\looseness`' wird auch am Ende des Paragraphen zurückgesetzt.

6.2.4 Der Satz von Paragraphen

Schließlich, wenn alle Berechnungen zum Zeilenumbruch beendet sind, müssen die Zeilen noch in die vertikale Liste der Seite geschrieben werden. Dabei geht TEX folgendermaßen vor: Unmittelbar vor dem Paragraphen wird ein spezielles Leimstück in die vertikale Liste eingefügt. Den Abstand zweier aufeinanderfolgender Paragraphen: '`\parskip`'. Plain TEX setzt diesen Abstand auf:

```
\parskip=0pt plus 1pt
```

Danach folgen die einzelnen Zeilen, die schon umgebrochen wurden. Zwischen die Zeilen werden ggf. noch besondere Strafpunkte geschrieben, die später für den Seitenumbruch gebraucht werden. Insbesondere wird zwischen die ersten beiden Zeilen und unmittelbar vor die letzte Zeile eine besondere Strafe geschrieben, um "Schusterjungen" und "Hurenkinder" zu vermeiden[3]. Normalerweise wird zwischen die Zeilen der Wert von '`\interlinepenalty`' geschrieben, handelt es sich um den Zwischenraum zwischen erster und zweiter Zeile, dann wird der Wert von '`\clubpenalty`' addiert. Handelt es sich um den Zwischenraum vor der letzten Zeile, dann wird '`\widowpenalty`' eingefügt, wenn keine abgesetzte Formel davor stand, sonst '`\displaywidowpenalty`'. Fand in der letzten Zeile eine Trennung statt, dann wird auch noch '`\brokenpenalty`' addiert. Die Werte werden in Plain TEX folgendermaßen gesetzt:

```
\interlinepenalty=0
\clubpenalty=150
\widowpenalty=150
\displaywidowpenalty=50
\brokenpenalty=100
```

Neben den horizontalen Linien kann man auch aus einem Paragraphen heraus andere Dinge in die vertikale Liste schreiben. Dazu dienen die Befehl '`\insert`', '`\mark`' und '`\vadjust`'. Die ersten beiden werden später beschrieben, der dritte fügt das Argument des Befehls unmittelbar nach der Zeile, in der der Befehl steht, in die vertikale Liste ein. Hier habe ich z.B. mit dem Befehl '`\vadjust{\kern2pt}`' einen zusätzlichen Zwischenraum in den Paragraphen eingefügt.

6.2.5 Die restlichen Befehle

Ausführungen für jeden Paragraphen

Der Befehl '`\everypar`' gestattet es einen Befehl zu Beginn *jedes* Paragraphen auszuführen. In der einfachsten Form sieht der Befehl folgendermaßen aus: '`\everypar{A}`'. Jedem Paragraphen

[3] Mit Schusterjungen und Hurenkindern bezeichnen die Drucker einzelne Zeilen eines Abschnittes, die durch Seitenumbruch vom Rest des Abschnittes getrennt wurde.

würde nun ein 'A' vorangestellt. Nicht sehr sinnvoll, aber denken sie einmal daran, wie in der LATEX Umgebung 'itemize' die Aufzählungsmarkierung realisiert wird.

Leere Zeilen

Einen leeren Paragraphen erhalten sie mit der Befehlsfolge '`\noindent\par`', wenn '`\everypar`' leer ist. Es wird dann nur '`\parskip`' in die vertikale Liste eingefügt.

Das letzte Hilfsmittel

Wenn überhaupt nichts mehr geht, und sie sich vor lauter Fehlermeldungen nicht mehr retten können, dann benutzen sie den Befehl '`\emergencystretch`'. Dieser Befehl läßt auch ungewöhnlich große Streckungen zu. Ihr Text wird dann allerdings nicht mehr so gut aussehen.

6.3 Wie macht TEX aus Paragraphen Seiten?

6.3.1 Der Normalfall

Sie können es TEX einfach machen die besten Stellen für den Seitenumbruch zu finden, wenn sie viele abgesetze Formeln in ihrem Text verwenden, oder wenn sie oft die Befehl '`\smallskip`', '`\medskip`' oder '`\bigskip`' verwenden. Haben die diese Möglichkeit nicht, und sie sind mit dem Umbruchalgorithmus von TEX nicht einverstanden, dann können sie auch jederzeit selbst einen Seitenumbruch bewirken, indem sie die Befehle '`\vfill\eject`' verwenden. Die Verwendung von '`\eject`' alleine ist nicht unbedingt zu empfehlen, dann dann die Zeilen der Seite auf das erforderliche Maß auseinandergezogen würden. Der Seitenumbruch geschieht sehr ähnlich, dem Zeilenumbruch, der im letzten Paragraphen besprochen wurde. Aus Speicherplatzgründen wird allerdings nur *lokal* optimiert. Es wird also nicht der gesamte Text im Speicher gelassen, bevor die besten Stellen für den Seitenumbruch gesucht werden.

6.3.2 Die gehts genau?

Die Inhalte einer Seite, werden zunächst auch einfach als Liste, hier aber als vertikale Liste repräsentiert. In dieser Liste können folgende Elemente auftreten:

1. Eine Box (hier eine hbox, eine vbox oder eine rulebox).
2. Ein "WasDenn" (kommt wieder später).
3. Eine Markierung (wird auch später erklärt).
4. Eine Einfügung (auch das erst später).
5. Etwas Leim (oder '`\leaders`').
6. Ein Kern.
7. Eine Strafe.

Die drei letzten Elemente sind dabei wieder *entfernbare* Elemente, entsprechend den Ausführungen im letzen Abschnitt. Es kann ebenfalls nicht an beliebiger Stelle eine Seitenumbruch erfolgen, das geht nur an:

1. Bei einem Leimstück, vorausgesetzt, daß dem Leim ein nicht-entfernbares Element unmittelbar vorangeht.
2. Bei einem Kern, vorausgesetzt, dem Kern folgt sofort etwas Leim.
3. Bei Strafpunkten.

Die Regeln für Strafpunkte entsprechen dabei wieder denen für den Zeilenumbruch. Im letzten Abschnitt wurde ja auch schon erläutert, an welchen Stellen einer vertikalen Liste Strafpunkte auftreten können.

Den Befehlen '`\small-`', '`\med-`' und '`\bigskip`' in der horizontalen Liste entsprechen jetzt die Befehle: '`\smallbreak`', '`\medbreak`' und '`\bigbreak`', mit denen -50, -100 oder -200 Strafpunkte vergeben werden können, um anzuzeigen, daß ein Seitenumbruch an einer bestimmten Stelle besonders gut oder schlecht zu machen ist. Außerdem gibt es noch den Befehl '`\goodbreak`', der eine Abkürzung für die Befehle '`\par\penalty-200`' darstellt. Dieser Befehl sollte am Ende eines Paragraphen verwendet werden, wenn dort ein Seitenumbruch erwünscht ist, ohne daß sofort mit dem drastischen Befehl '`\eject`' gearbeitet werden soll. Schließlich gibt es noch den Befehl '`\filbreak`'. Dieser Befehl schließt die Seite ab und füllt sie mit Leerraum auf, wenn das vertikale Material bis zum nächsten '`\filbreak`' nicht mehr auf die Seite paßt.

Das Analog zum Befehl '`\raggedright`' heißt '`\raggedbottom`' und dieser Befehl bewirkt, daß die Seiten nicht mehr gleichmäßig aufgefüllt werden, sondern der Zeilenabstand immer konstant bleibt.

Zur Berechnung des Seitenumbruchs kommt es nun folgendermaßen. Zu Beginn kennt TEX die Größen '`\vsize`', sowie '`\maxdepth`'. Aus diesen Werten berechnet sich TEX das *Ziel* der Berechnung: '`\pagegoal`'.[4] Die aktuelle Höhe der Seite wird in '`\pagetotal`' gespeichert. Nun wird Zeile an Zeile in die Seite geschrieben, bis die Seite voll ist, also '`\pagegoal`' gleich '`\pagetotal`' ist. Das passiert natürlich nur im Idealfall. Würde die Seite mit der letzten vertikalen Box, die eingefügt werden soll zu voll, dann wird sie sozusagen in eine Warteposition gestellt, dann wird versucht noch Einfügungen auf der Seite unterzubringen[5] bevor sie dann ausgegeben wird; genauer an die Outputroutine übergeben wird. Seitenzahlen und ähnliches wird erst *nach* Fertigstellung der Seite hinzugefügt.

6.3.3 Einfügungen

Illustrationen

Fußnoten und Illustrationen werden in TEX als Einfügungen realisiert. Der Rest des Kapitels beschäftigt sich mit der Frage, wie diese Einfügungen mit dem Seitenumbruchalgorithmus kooperieren. Zunächst beschäftigen wir uns dabei mit den komplizierteren Befehlen, die Plain TEX zur Verfügung stellt, später sehen wir uns dann an, wie die eigentliche Organisation vonstatten geht.

Normale Einfügungen

Die einfachste Art etwas in einen Text einzufügen besteht in dem Anweisungspaar:

```
\topinsert <vertikales Material> \endinsert
```

Das 'vertikale Material' ist dabei das, was eingefügt werden soll. TEX versucht dieses Material am Anfang der laufenden Seite unterzubringen. Ist dort kein Platz, dann wird es an den Anfang der nächsten Seite geschrieben. Man kann z.B. folgendes schreiben:

```
\topinsert\vskip2in\hsize=3in\noindent
{\bf Figur 3} Dies ist die Unterschrift unter der dritten Figur meines
Textes. Ich habe vor dem Text 2 Inch Platz gelassen, damit ich dort
sp\"ater meine Illustration einkleben kann \endinsert
```

TEX fügt hinter dem eingefügten Material automatisch einen '`\bigskip`' ein.

Analog funktioniert der Befehl '`\pageinsert`', nur daß das vertikale Material, das dem Befehl folgt auf eine eigene Seite geschrieben wird, üblicherweise auf die nächste Seite. und schließlich gibt es

[4] Zusätzlich wird noch der Wert von `\topskip` ausgewertet, der angibt, wieviel am oberen Rand der Seite freigelassen werden soll.

[5] Kommt gleich.

noch den Befehl '`\midinsert`', der versucht das Material an der Stelle des Auftretens unterzubringen. Dann ist der Effekt wie:

```
\bigskip\vbox{<vertikales Material>}\bigbreak
```

ist nicht genug Platz vorhanden, dann wird das Material wie bei '`\topinsert`' behandelt.

Benutzen sie die Befehl nicht innerhalb von horizontalen Boxen, sondern nur an Stellen, an denen sich TEX im vertikalen Modus befindet.

Wenn sie viele der obigen Befehl hintereinander benutzen, kann es passieren, daß TEX die Inhalte der Befehl auf viele folgende Seiten verteilen muß. Die Einfügungen behalten dabei aber ihre vorgegebene Reihenfolge. Wenn sie verhindern wollen, daß Einfügungen in ein späteres Kapitel übernommen werden, dann benutzen sie die Befehlsfolge '`\vfill\supereject`'. Damit werden alle noch vorhandenen Einfügungen, nach Leerraum ausgegeben, bevor neuer Text gesetzt wird.

Fußnoten

Neben der Möglichkeit mit '`\topinsert`' etwas an den Anfang einer Seite zu schreiben, wird mit dem Befehl '`\footnote`' etwas ans Ende einer Seite geschrieben. Außerdem kann dieser Befehl auch innerhalb eines Paragraphen benutzt werden. Fußnoten werden mit zwei Argumenten aufgerufen, das erste Argument stellt dabei die Fußnotenmarkierung, das zweite den Fußnotentext dar.[6]

Fußnoten können umgebrochen werden und auf der folgenden Seite weitergeführt werden. Dies wird allerdings soweit möglich vermieden. Einfügungen können nicht innerhalb von Einfügungen verwendet werden. Es ist also nicht möglich innerhalb eines '`\topinsert`'s eine Fußnote zu benutzen. Sollten sie es doch einmal brauchen, dann können sie den Befehl '`\vfootnote`' verwenden. Er ist dazu da im vertikalen Modus eine Fußnote zu schreiben, und sie müssen dann die Fußnotenmarkierung in der Einfügung per Hand anbringen, und den '`\vfootnote`' Befehl irgendwo auf die Seite schreiben, auf der die Einfügung später erscheinen wird. Sie merken schon, nicht ganz elegant, aber das sind Fußnoten sowieso nicht.

Bevor wir uns nun mit den Feinheiten von Einfügungen beschäftigen können müssen wir lernen, was TEX unter einem *Register* versteht.

6.3.4 Register

Einfache Register

TEX kennt ein Konstrukt, daß sich Register nennt und das in anderen Programmiersprachen wohl am ehesten den Variablen gleichkommt. Es gibt verschiedene Registertypen, und von jedem Registertyp 256 Register. Die einfachen Registertypen sind das Zahlenregister, das Dimensionsregister, das Skipregister und das Muskipregister. Die Zahlregister heißen '`\count0`' ... '`\count255`', die Dimensionsregister '`\dimen0`' ... '`\dimen255`'. Der Zahlbereich der Zahlregister reicht von -2^{31} bis 2^{31}. Die Dimensionsregister können jeden gültigen Dimensionswert annehmen, die Skip- und Muskipregister jeden gültigen Wert für Leim. Die Zuweisung erfolgt als:

```
\count<number> = <number>
\dimen<number> = <dimen>
\skip<number> = <glue>
\muskip<number> = <muglue>
```

Mit diesen Register können sie auch rechnen. Addition und Subtraktion erfolgen mit dem gleichen Befehl:

---

[6] In diesem Text wird das Fußnotenmakro von LATEX benutzt. In *Plain* TEX sieht die Sache etwas anders aus. Zwischen den Fußnoten wird kein zusätzlicher Zwischenraum eingefügt, und die Fußnoten werden auch in normaler Schriftgröße gesetzt. Die Fußnotenmarkierung wird in einen Zwischenraum der Größe `\textindent` gesetzt. Das Makro `\textindent` arbeitet ähnlich, wie `\item`, allerdings ohne eine hängende Einrückung.

```
\advance\count<number> by <number>
\advance\dimen<number> by <dimen>
\advance\skip<number> by <glue>
\advance\muskip<number> by <muglue>
```

Die Anweisung: ‘`\dimen8=\hsize \advance\dimen8 by 1in`’ bewirkt z.B., daß der Inhalt des 9ten Dimensionsregister eine Länge enthält, die um ein Inch größer ist, als die augenblickliche Textbreite.

Unendliche Maße verdrängen endliche Maße. Nach den Anweisungen:

```
\skip2 = 0pt plus 2fill minus 3fill
\advance\skip2 by 4pt plus 1fil minus 2filll
```

Hat das Register den Wert:

```
4pt plus 2fill minus 2filll
```

Auch die Division und Multiplikation von Registern ist möglich, allerdings nur ganzzahligen Werten. Bei den Zahlregistern ist die Sache einfach. Bei den anderen Registern werden alle Angaben mit der entsprechenden Zahl multipliziert, bzw. dividiert. ‘`\multiply\count2 by 2`’ verdoppelt beispielsweise den Inhalt des 3ten Zahlregisters, ‘`\divide\skip2 by 2`’ halbiert alle Werte im 3ten Skipregister. Da es sich nur um ganzzahlige Rechenoperationen handelt, wird bei der Division etwaiger Rest vernachlässigt. Das Vorzeichen des Ergebnisses bei der Division hängt von den Vorzeichen der Operanden ab. Bei ungleichen Vorzeichen ist das Ergebnis negativ, sonst positiv.[7]

Sie können Register überall da verwenden, wo sie auch explizite Angaben verwenden dürfen. Die Angabe von ‘`\hskip\dimen2`’ ist ebenso erlaubt, wie die Angabe von ‘`\advance\count10 by \count10`’.

Dimensionsregister können auch als Zahlregister, und Skipregister können ihrerseits sowohl als Dimensions- oder auch als Zahlregister verwendet werden. Bei der Verwendung eines Skipregisters als Dimensionsregister, wird etwaige Angaben für Schrumpfung oder Streckung ignoriert, bei der Verwendung eines Dimensionsregisters wird als Einheit der *sp* (Scaled Point) verwendet.

Normalerweise haben Register ihren Wert nur innerhalb der Gruppe, in der sie benutzt werden,[8] es ist aber möglich Register auch für alle Gruppen zu ändern, indem man den Befehl ‘`\global`’ voranstellt. Die Anweisungen:

```
\count1=10 \count2=20 \count3=30
{\count1=20 \global\advance\count1 by 12 \global\count2=2 \count2=4}
```

haben zur Folge, daß die Register folgende Werte haben:

```
\count1 : 32
\count2 :  2
\count3 : 30
```

Die ersten zehn Zahlregister (‘`\count0`’ ... ‘`\count9`’) sind reserviert und sollten nicht benutzt werden. Die Inhalte dieser Register werden bei der Ausgabe auf dem Bildschirm ausgegeben und durch Dezimalpunkte getrennt. Führende leere Register werden dabei nicht ausgegeben. *Plain* TeX benutzt nur das erste Zahlregister für die Seitenzahl, weshalb sie auch meist nur die Ausgabe: ‘`[0] [1] [2] ...`’ sehen. Wäre z.B. ‘`\count5=3`’ und ‘`\count7=4`’ dann wäre die Ausgabe: ‘`[3.0.4]`’.

Es ist in TeX einfach möglich symbolische Namen für Register zu vergeben. Dazu gibt es z.B. den Befehl ‘`\countdef`’. Nach der Anweisung:

```
\countdef\chapno=28
```

kann der Befehl ‘`\chapno`’ als Abkürzung für ‘`\count28`’ verwendet werden. Analog arbeiten die Befehle: ‘`\dimendef`’, ‘`\skipdef`’ und ‘`\muskipdef`’

[7] Vermeiden sie die Division durch Null und Multiplikationen, die die Kapazität des Registers überschreiten.

[8] Daraus ergibt sich, daß TeX wesentlich mehr Register hat, als oben angegeben.

Boxregister

Neben den numerischen Registern kennt TEX auch noch die sogenannten Boxregister (`\box0` ... `\box255`). In diesen Boxregistern kann eine beliebige hbox, vbox oder Rulebox gespeichert werden. Die Anweisung:

```
\setbox0=\hbox{A}
```

hat zur Folge, daß das erste Boxregister (`\box0`) den Buchstaben 'A' zum Inhalt hat. Die Abmessungen des Inhalts eines derartigen Registers können abgerufen werden. Mit den Befehlen:

```
\wd<nummer>
\ht<nummer>
\dp<nummer>
```

werden die Breite, Höhe und Tiefe der entsprechenden Box angesprochen.

Einen wichtigen Unterschied zwischen Boxregistern und numerischen Registern gibt es: Boxregister werden beim Gebrauch geleert. Die Anweisung `\raise\2pt\box3` in einer horizontalen Liste bewirkt erstens, daß der Inhalt der 4ten Box, um 2pt noch oben verschoben, in die Liste aufgenommen wird, zweitens ist aber auch die 4te Box nach dieser Anweisung leer. Wenn sie den Inhalt dieser Box nocheinmal verwenden wollen, benutzen sie den Befehl `\copy`, satt des Befehls `\box`. Im obigen Beispiel wäre dann die Anweisung `\raise2pt\copy3` richtig gewesen.

Eine weitere Anweisung, die mit Boxregistern durchgeführt werden kann ist das '*Unboxen*'. Damit können sie eine Ebene von Boxen aufheben. Die Anweisungen:

```
\setbox3=\hbox{A} \setbox3=\hbox{\box3 B}
\setbox4=\hbox{A} \setbox4=\hbox{\unhbox4 B}
```

bewirken, daß in `\box3` `\hbox{\hbox{A} B}` steht, und in `\box4` `\hbox{AB}`. Analog arbeitet der Befehl `\unvbox` für vertikale Boxen. Der Vorteil dieses Befehls liegt auf der Hand: Weniger Speicherbenutzung und schnellere Bearbeitung. Ebenso gibt es die Befehle `\unhcopy` und `\unvcopy`, deren Verwendung eigentlich sofort klar sein dürfte. Interessant ist nur noch das Verhalten von Leim beim Unboxen. Nach

```
\setbox5=\hbox{A \hbox{B C}}
\setbox6=\hbox to 1.05\wd5{\unhcopy5}
```

ist `\box6` um 5 Prozent breiter, als `\box5`, es wird aber nur zwischen dem 'A' und dem 'B' gestreckt, die *innere* Box bleibt unberührt.

Es gibt einen Unterschied zwischen einem leeren Boxregister und einem Boxregister, das eine leere Box enthält. Enthält ein Register eine leere hbox (`\hbox{}`), dann können auf dieses Register nur die Befehl `\unhbox` und `\unhcopy` angewendet werden. die vertikalen Äquivalente sind nicht mehr zuläßig. Bei einer leeren Box durfen alle Befehle verwendet werden.

Allgemeine Registerbefehle

Wenn sie sich unschlüssig sind, wie TEX mit Registern arbeitet, dann können sie verschiedene `\show`-Befehle benutzen.

```
\showthe\count1 \showthe\dimen2 \showthe\skip3
```

gibt z.B. die Inhalte des 2ten Zahl-, 3ten Dimensions- und 4ten Skipregisters aus. Mit dem Befehl `\showbox4` können sie sich den Inhalt des 5ten Boxregisters ansehen.[9]

Wenn viele Leute an einem Makropaket für TEX arbeiten, wäre es fatal, wenn verschiedene Menschen dasselbe Register für unterschiedliche Zwecke benutzen wollen. TEX stellt deshalb den Befehl `\newcount` bereit, um ein noch unbenutztes Register ansprechen zu können. Sie können einfach mit

[9] Die Ausgabe erfolgt übrigens nur in den Logfile, es sei denn, sie setzen `\tracingonline=1`.

```
\newcount\meinzaehler
```

ein neues Zahlregister bereitstellen lassen, daß dann mit '`\meinzaehler`' angesprochen werden kann. Analog gibt es auch die Befehle:

```
\newdimen
\newskip
\newmuskip
\newbox
```

Außerdem stellt *Plain* TEX auch noch die Befehle:

```
\newtoks
\newread
\newwrite
\newfam
\newinsert
```

bereit, deren Bedeutung später erklärt wird.

Wenn sie den Inhalt eines numerischen Registers in den Text übernehmen wollen, benutzen sie den Befehl '`\the`'. Mit '`\the\meinzaehler`' wird der augenblickliche Inhalt des Zahlregisters '`\meinzaehler`' in den Text übernommen.

Für lokalen Gebrauch hat es sich durchgesetzt die Register '`\count255`', '`\dimen255`' usw. zu benutzen. Dabei sollten sie allerdings nicht das Register '`\box255`' benutzen, da es eine besondere Bedeutung hat. Die Register `0...9` haben auch spezielle Bedeutungen und werden auch nicht von den '`\new...`' Befehlen zurückgegeben.

6.3.5 Die Details der Einfügung

Es gibt 255 Klassen von Einfügungen. '`\insert0`' ... '`\insert255`'. Jede Einfügung korrespondiert mit einigen der entsprechenden bekannten Registern. '`\insert100`' korrespondiert mit '`\count100`', '`\dimen100`', '`\skip100`' und '`\box100`'. *Plain* TEX stellt Befehle zur Verfügung, diese Register sofort zu reservieren. Mit dem Befehl

```
\newinsert\footins
```

Wird z.B. die Einfügungsnummer für Fußnoten festegelegt. Mit '`\count\footins`' kann jetzt das entsprechende Zahlregister angesprochen werden. Tatsächlich sind die Klassennummern festgelegt,[10] aber im weiteren braucht uns das zunächst nicht zu stören. Nehmen wir einfach an es ginge um die Einfügung n. Dann gilt für diese Klasse von Einfügung:

`\box`n	Enthält das Material, das bei der Ausgabe eingefügt werden soll.
`\count`n	Enthält den Vergrößerungsfaktor, der für den Seitenumbruch gebraucht wird.
`\dimen`n	Enthält die maximale Einfügungsgröße pro Seite.
`\skip`n	Enthält den Platz, der auf einer Seite zusätzlich bereitgestellt werden soll.

Wenn die Einfügung einfach nur ihre normale Höhe hat, kann der Vergrößerungsfaktor auf 1000 gesetzt werden, auf dem er normalerweise auch steht. Eine 10pt Fußnote braucht auch 10pt Platz. Es kann nun aber sein, daß diese Fußnote in zwei Spalten gesetzt werden soll, dann braucht sie in der Höhe nur den halben Platz. In diesem Falle wäre '`\count`n' auf 500 zu setzen. Der Vergrößerungsfaktor wird also gebraucht um auch in schwierigen Fällen die Seitenberechnung richtig durchführen zu können.

[10] Fußnoten sind Klasse 254, Topeinfügungen die Klasse 253

Die erste Fußnote auf einer Seite braucht etwas mehr Platz, als sie selber einnimmt, da wir ja die Fußnoten etwas vom übrigen Text absetzen wollen und auch noch eine horizontale Linie einfügen wollen. Dieser Extraplatz steht in dem entsprechenden Skipregister. *Plain* TEX setzt z.B.

```
\skip\footins=\bigskipamount
```

Damit wird der notwendige Platz bereitgestellt.

Mitunter ist es wünschenswert den maximalen Platz für Einfügungen zu begrenzen. Z.B. will niemand eine ganze Seite voller Fußnoten lesen. Der Maximale Paltz, den alle Einfügungen einer Klasse einnehmen dürfen steht im entsprechenden Dimensionsregister. Für Fußnoten etwa, definiert *Plain* TEX:

```
\dimen\footins=8 in
```

Somit können maximal 8 Inch einer Seite mit Fußnoten aufgefüllt werden, der Rest wird auf weitere Seiten übernommen.

Zur Zerlegung einer Einfügung gibt es einen Befehl: '`\vsplit`'. Mit der Anweisung

```
\setbox200=\vsplit100 to 50pt
```

geht TEX folgendermaßen um: Zunächst wird der Anfang des Inhaltes von Box100 nach Box200 kopiert, bis die Box200 die geforderte Höhe von 50pt hat.[11] Danach wird aus der Box100 alles entfernt, was jetzt in der Box200 steht. So wird auch verfahren, wenn eine Einfügung über mehr als eine Seite verteilt werden soll.

[11] Hierbei werden dieselben Regeln wirksam, wie beim Seitenumbruch.

Kapitel 7

Mathematische Formeln

7.1 Eingabe einfacher Formeln

7.1.1 Zeichen im mathematischen Modus

Mathematische Formeln werden in TEX mit zwei Dollarzeichen ($) eingeklammert. Alles was zwischen diesen Klammern steht wird als mathematische Formel verstanden. TEX verfährt dabei so, daß komplexere Formeln aus einfacheren Formeln modular zusammengesetzt werden.

Innerhalb des mathematischen Modus von TEX verhält sich TEX anders, als im Textmodus. Die normalen Zeichen ('A'...'Z' und 'a'...'z') werden in Italicschrift gesetzt. Die Punkte. Kommata u.ä. werden normal in Roman gesetzt, aber z.B. das Minuszeichen sieht anders aus, als der einfache Bindestrich. Leerzeichen werden im mathematischen Modus gänzlich ignoriert, statt dessen verwendet TEX eigene Abstände für die Formeln. Die Funktion des Leerzeichens als Beendigung einer Kontrollsequenz bleibt allerdings erhalten. Die in der Mathematik oft gebrauchten griechischen Buchstaben stehen sowohl als Klein, wie auch als Großbuchstaben zur Verfügung. Die Eingabe von '`$$\alpha, \beta, \gamma, \delta$$`' ergibt:

$$\alpha, \beta, \gamma, \delta$$

Und durch eine kleine Änderung wird mit '`$\Gamma$`': Γ. Einige griechische Buchstaben stehen sogar in verschiedenen Ausführungen bereit.

'`$\phi,\theta,\epsilon,\rho$`' ergibt: $\phi, \theta, \epsilon, \rho$

'`$\varphi,\vartheta,\varepsilon,\varrho$`' ergibt: $\varphi, \vartheta, \varepsilon, \varrho$

Darüber hinaus gibt es noch viele Zeichen, die meist nur in der Mathematik verwendet werden, wie '$\approx$' ('`\approx`') oder '$\mapsto$' ('`\mapsto`').

7.1.2 Potenzen und Indizes

Nun soll etwas $^{\text{hochgestellt}}$ oder $_{\text{tiefgestellt}}$ werden. In TEX verwendet man dazu die Zeichen '`^`' und '`_`'. Im einfachsten Fall, wenn nur ein Zeichen hoch- bzw. tiefgestellt werden soll, sieht das folgendermaßen aus:

Eingabe	*Ausgabe*
`$x^2$`	x^2
`$x_2$`	x_2
`$2^x$`	2^x
`$x^2y ^2$`	x^2y^2
`$x ^ 2y ^ 2$`	x^2y^2
`$_2F^3$`	$_2F_3$

Wenn mehrere Zeichen hoch- bzw. tiefgestellt werden sollen, dann müssen diese Zeichen zu einer Gruppe zusammengefaßt werden:

`$x^{2y}æ`	x^{2y}
`$2^{2^x}$`	2^{2^x}
`$y_{x^2}$`	y_{x^2}

Wie es auch bei *guten* Mathematikern der Fall ist, sind Kombinationen, wie '`x^y^z`' oder '`x_y_z`' verboten.[1] Verwenden sie *immer* Gruppenklammern, um ihre Intention deutlich zu machen, also z.B. '`{x_y}_z`' oder '`x^{y^z}`'. Aus ähnlichem Grunde ist es auch schlechter Stil '`_2F_3`' zu schreiben. Setzen sie den ersten Index besser an eine leere Gruppe: '`{}_2F_3`'.

Die Reihenfolge der Indizierung und Potenzierung eines Zeichens ist egal, wie sie bei den folgenden Beispielen sehen können:

`$x^2_3$`	x^2_3
`$x_3^2$`	x_3^2
`$x^{31415}_{92}+\pi$`	$x^{314151}_{92}+\pi$
`$x_{y^a_b}^{z_c^d}$`	$x_{y^a_b}^{z_c^d}$

Das die normalen Zeichen in Italic ausgegeben werden, werden Indizes etwas an das Zeichen herangerückt, der Index steht also nicht genau unter der Potenz. Will man diesen Effekt vermeiden, muß man wieder eine leere Gruppe verwenden:

`$P_2^2$`	P_2^2
`$P{}_2^2$`	$P{}_2^2$

Sehr oft wird als hochgestelltes Zeichen in der Mathematik ein einfacher Strich benutzt. TEX stellt hierfür den Befehl '`\prime`' zur Verfügung.

`$y_1^\prime$`	$y_1^\prime$
`$y_2^{\prime\prime}$`	$y_2^{\prime\prime}$
`$y_2^{\prime\prime\prime}$`	$y_2^{\prime\prime\prime}$

7.1.3 Wurzeln und Ähnliches

In TEX gibt es einige Zeichen, die sich in ihrer Größe automatisch den Erfordernissen anpassen. Dazu gehören das Wurzelzeichen ('`\sqrt`'), die Unterstreichung ('`\underline`') und die Überstreichung ('`\overline`'). Die Eingaben von

```
$\sqrt2$
$\sqrt{x+2}$
$\underline4$
$\overline{x+y}$
$\overline x+\overline y$
$x^{\underline n}$
$x^{\overline{n+m}}$
$\sqrt{x^3+\sqrt\alpha}$
```

ergeben der Reihe nach:

$\sqrt2$, $\sqrt{x+2}$, $\underline4$, $\overline{x+y}$, $\overline x+\overline y$, $x^{\underline n}$, $x^{\overline{n+m}}$, $\sqrt{x^3+\sqrt\alpha}$

Auch andere, als die zweite Wurzeln, lassen sich in TEX realisieren:

`$\root 3 \of 2$`	$\sqrt[3]{2}$
`$\root n \of {x^n+y^n}$`	$\sqrt[n]{x^n+y^n}$
`$\root n+1 \of a$`	$\sqrt[n+1]{a}$

Wurzeln, Über- und Unterstreichungen passen sich nicht nur der Länge des Arguments an, sondern auch der Höhe. Deutlich wird das am Unterschied zwischen '`\overline l`' ($\overline l$) und '`\overline m`'

[1] Der Grund liegt in der Nichteindeutigkeit der Schreibweise. 9^{9^9} kann sowohl als $9^{(9^9)}$ verstanden werden, als auch als $(9^9)^9 = 9^{81}$

($\overline{m}$). Oder auch der Folge: $\sqrt{a}+\sqrt{d}+\sqrt{y}$. Wenn sie diese unterschiedlichen Höhen vermeiden wollen, dann gibt es den Befehl '`\mathstrut`', der dem '`\strut`' Befehl weitgehend entspricht, nur für den mathematischen Modus. Die Eingabe von

```
$\sqrt{\mathstrut a}+\sqrt{\mathstrut d}+\sqrt{\mathstrut y}$
```

ergibt: $\sqrt{\mathstrut a}+\sqrt{\mathstrut d}+\sqrt{\mathstrut y}$

7.1.4 Die Zeichen im mathematischen Modus

Ich habe oben schon angedeutet, daß im mathematischen Modus manche Zeichen anders ausgegeben werden, als im Textmodus. Hier nun ein etwas systematischerer Überblick, welche Zeichen sich wie ändern.

Die normalen Buchstaben ('A' bis 'Z' und 'a' bis 'z') werden in Italic gesetzt, da damit üblicherweise Variablen bezeichnet werden.

$$A \dots Z \quad \text{und} \quad a \dots z$$

Für den Buchstaben 'l' gibt es sogar zwei Zeichen, da in handschriftlichen Texten das 'l' oft mit der '1' verwechselt wird. Der zugehörige Befehl heißt '`\ell`' und er ergibt ℓ.

Die 18 Zeichen

```
0 1 2 3 4 5 6 7 8 9 ! ? . | / ‘ @ \"
```

sind einfache Symbole, für sie wird kein extra Platz bereitgestellt. Man achte allerdings auf den Unterschied zwischen der Null (0) und dem kleinen oder großen 'O'.

Die drei Zeichen '`-`', '`+`' und '`*`' heißen binäre Operatoren, da sie sich üblicherweise auf zwei Teile einer Formel beziehen. Sie werden mit etwas zusätzlichem Zwischenraum gesetzt:

Eingabe	*Ausgabe*
`$x+y-z#`	$x+y-z$
`$x+y*z$`	$x+y*z$
`$x*y/z$`	$x*y/z$

Beachten sie auch, daß der Bindestrich, im Textmodus '-', zum Minuszeichen ($-$) wird, und das Sternchen, im Textmodus '*' tiefer gesetzt wird ($*$). Beim Schrägstrich wird *kein* zusätzlicher Platz gelassen, auch wenn meist zwei Teile durcheinander geteilt werden.

TEX stellt noch mehr zweiseitige Operatoren bereit:

`$c\times y\cdot z$`	$x \times y \cdot z$
`$x\circ y\bullet z$`	$x \circ y \bullet z$
`$x\cup y\cap z$`	$x \cup y \cap z$
`$x\sqcup y\sqcap z$`	$x \sqcup y \sqcap z$
`$x\vee y\wedge z$`	$x \vee y \wedge z$
`$x\mp y\pm z$`	$x \mp y \pm z$

In manchen Fällen wird auch bei einem zweiseitigen Operator *kein* zusätzlicher Zwischenraum eingefügt:

1. Wenn nur auf einer Seite des Operators etwas steht:

`$x=+1$`	$x=+1$
`$3.142-$`	$3.142-$
`$(D*)$`	$(D*)$

2. Bei Indizes und Potenzen

`$K_n^+,K_n^-$`	K_n^+, K_n^-
`$z^*_{ij}$`	z^*_{ij}
`$g^\circ \mapsto g^\bullet$`	$g^\circ \mapsto g^\bullet$
`$f^*(x) \cap f_*(y)$`	$f^*(x) \cap f_*(y)$

Neben den einfachen zweiseitigen Operatoren stellt TEX auch noch Relationsoperatoren bereit, die *etwas* anders gesetzt werden. Die einfachsten Relationsoperatoren sind '`=`', '`<`', '`>`' und '`:`', wie die folgenden Beispiele aber zeigen, nicht die einzigen.

`$x=y>z$`	$x = y > z$
`$x:=y$`	$x := y$
`$x\le y\ne z$`	$x \le y \ne z$
`$x\sim y\simeq z$`	$x \sim y \simeq z$
`$x\equiv y\not\equiv z$`	$x \equiv y \not\equiv z$
`$x\subset y\subseteq z$`	$x \subset y \subseteq z$

Die Interpunktionszeichen Komma (,) und Semikolon (;) werden mit etwas zusätzlichem Platz *hinter* dem Zeichen, aber nicht *vor* dem Zeichen gesetzt

`$f(x,y;z)$` $f(x, y; z)$

Der Punkt wird als normales Zeichen behandelt, anders, als der Doppelpunkt. Soll dieser als normales Zeichen gesetzt werden, dann kann das mit dem Befehl '`\colon`' geschehen.

`$f:A\to B$`	$f : A \to B$
`$f\colon A\to B$`	$f\colon A \to B$

Wenn das Komma als einfaches Symbol gesetzt werden soll, schreiben sie es in geschweifte Klammern. Alles, was in geschweiften Klammern steht wird als einfaches Symbol betrachtet.

`$12,345x$`	$12, 345x$	(falsch)
`$12{,}345x$`	$12{,}345x$	(richtig)

Nun gibt es nur noch einige Tasten zu besprechen. Die Zeichen '`(`' und '`[`' werden 'Öffner' genannt, die Zeichen '`)`' und '`]`' 'Schließer'. Das Zeichen '`'`' kennen wir ja schon, und es bleiben nur noch die reservierten Zeichen:

```
\ $ % # & ~ { } _ ^
```

Die besonderen Zeichen für Hoch- und Tiefstellung, sowie einige der anderen mathematischen Befehle, wie '`\approx`' oder '`\alpha`' können nicht in normalem Text verwendet werden. TEX benutzt diese Zeichen, um einen möglichen Fehler ihrerseits finden zu können. Wird ein derartiges Zeichen oder ein solcher Befehl außerhalb des mathematischen Modus angetroffen, geht TEX davon aus, daß die ein '`$`' vergessen haben, oder eins zuviel geschrieben haben.

7.1.5 Die mathematischen Akzente

TEX stellt neben den oben beschriebenen Zeichen auch noch eine Reihe von Akzenten bereit, die auch nur im mathematischen Modus ansprechbar sind. Hier eine vollständige Liste:

`$\hat a$`	$\hat{a}$
`$\check a$`	$\check{a}$
`$\tilde a$`	$\tilde{a}$
`$\acute a$`	$\acute{a}$
`$\grave a$`	$\grave{a}$
`$\dot a$`	$\dot{a}$
`$\ddot$`	$\ddot{a}$
`$\breve$`	$\breve{a}$
`$\bar$`	$\bar{a}$
`$\vec$`	$\vec{a}$

Die Akzente lassen sich natürlich auch auf andere Buchstaben als das 'a' setzen. Die ersten neun Akzente sind auch im Textmodus verwendbar, haben dort allerdings eine andere Kontrollsequenz. Wenn sie einen der Buchstaben mit einem bestimmten Akzent öfters brauchen, emfiehlt sich eine Definition zu Beginn des Textes

```
\def\Ahat{{\hat A}}
\def\chat{{\hat c}}
```

Die Buchstaben 'i' und 'j' stehen im mathematischen Modus auch ohne Punkt,mit den Befehlen '`\imath`' und '`\jmath`', zur Verfügung. Diese eignen sich besser zur Akzentuierung.

Auch doppelte Akzente, jeder Mathematiker hätte seine helle Freude, lassen sich in TEX realsieren, es bedarf allerdings etwas Mühe. Das Symbol '$\hat{\hat{A}}$' läßt sich z.B. mit dem Befehl '`\skew6\hat\Ahat`' erreichen. Die '6' wurde durch probieren gefunden, mit dieser Angabe wird der Akzent nach rechts oder links verschoben. '5' hätte den Akzent etwas zu weit nach links gesetzt, '7' etwas zu weit nach rechts. Der entscheidende Befehl heißt '`\skew`'.

Abschließende kennt TEX noch zwei Akzente, die mit dem Argument mitwachsen (zumindest in gewissem Maße). Es sind dies die Akzente '`\widehat`' und '`\widetilde`'

`$\widehat x$`	$\widehat{x}$
`$\widetilde x$`	$\widetilde{x}$
`$\widehat{xy}$`	$\widehat{xy}$
`$\widetilde{xy}$`	$\widetilde{xy}$
`$\widehat{xyz}$`	$\widehat{xyz}$
`$\widetilde{xyz}$`	$\widetilde{xyz}$

7.2 Weitere Konstruktionselemente

7.2.1 Gestapelte Formeln

Die beliebtesten Formeln aller Mathematiker sind die Brüche und verwandte Formeln. Die Konstruktion von

$$\frac{1}{2} \quad \text{oder} \quad \frac{n+1}{3} \quad \text{oder} \quad \binom{n+1}{3} \quad \text{oder} \quad \sum_{n=1}^{3} Z_n^2$$

sind in TEX möglich durch die Befehle

```
$$1\over2$$
$$n+1\over3$$
$$n+1\choose3$$
$$\sum_{n=1}^3 Z_n^2$$
```

Brüche

Brüche werden in TEX durch den Befehl '`\over`' realisiert. Er bezieht sich auf *alles* was vor resp. nach ihm steht, es sei denn sie gruppieren die gewünschten Zeichen.

`$$x+y^2\over k+1$$`	$\frac{x+y^2}{k+1}$
`$${x+y^2\over k}+1$$`	$\frac{x+y^2}{k}+1$
`$$x+{y^2\over k}+1$$`	$x+\frac{y^2}{k}+1$
`$$x+{y^2\over k+1}$$`	$x+\frac{y^2}{k+1}$
`$$x+y^{2\over k+1}$$`	$x+y^{\frac{2}{k+1}}$

Bei Mehrfachbrüchen müssen sie sogar geschweifte Klammern setzen

`$${a\over b}\over 2$$`	$\frac{\frac{a}{b}}{2}$
`$$a\over{b\over 2}$$`	$\frac{a}{\frac{b}{2}}$

Bei derartigen Mehrfachbrüchen empfiehlt sich aber sowieso die Verwendung des Schrägstriches. Die obigen Brüche sähen dann wie $\frac{a/b}{2}$ und $\frac{a}{b/2}$ aus. Denken sie nur daran, daß bei Brüchen mit Schrägstrichen etwas andere Regeln gelten. Sie müssen eventuell ein paar Klammern mehr schreiben.

7.2.2 Die verschiedenen Stile

Bei den Beispielen dürften ihnen schon aufgefallen sein, daß TEX die Zeichen im mathematischen Modus in unterschiedlichen Größen und Stilen setzt. Es gibt acht verschiedene Stile in TEX eine Formel zu setzen:

Display Stil	Für abgesetzte Formeln
Text Stil	Für Formeln im Text
Skript Stil	Für Hoch- bzw. Tiefstellungen
Skriptskript Stil	Für die zweite Ebene von Hoch- bzw. Tiefstellungen

Außerdem gibt es noch zu jedem Stil einen "geklammerten" Stil, der sich nicht wesentlich unterscheidet von dem Originalstil, nur daß die Exponenten nicht so hoch gesetzt werden. Im weiteren werden die Stile durch:

$$D, D', T, T', S, S', SS, SS'$$

abgekürzt. Wird eine Formel in einfachen Dollarzeichen eingeschlossen, dann erscheint sie in Textstil, ist sie in doppelten Dollarzeichen eingeschlossen, erscheint sie im Displaystil. Aus der Stilart folgt die Größe, in der die Formel, oder die Unterformel, gesetzt wird.

Es gibt keinen SSS-Stil. Dieser wäre nicht mehr lesbar. Die Zusammenhänge der Stilarten ergeben sich wie folgt:

Die Formel	*Hochstellung*	*Tiefstellung*
D, T	S	S'
D', T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Wenn z.B. die Formel '`x^{a_b}`' im D-Stil gesetzt werden soll, dann wird das '`a`' in S und das '`b`' in SS' gesetzt. Das Ergebnis ist: x^{a_b}.

Der Hauptunterschied zwischen Displaystil und Textstil zeigt sich bei Brüchen. Die Zusammenhänge zeigt die folgende Tabelle:

	Bruch	
Formel	*Nenner*	*Zähler*
D	T	T'
D'	T'	T'
T	S	S'
T'	S'	S'
S, SS	SS	SS'
S', SS'	SS',	SS'

Am deutlichsten immer noch im Beispiel. Der Bruch 'einhalb' erscheint im Text als: $\frac{1}{2}$ und in der abgesetzten Formel als:

$$\frac{1}{2}$$

Übrigens, auch die Befehle '`\overline`' und '`\sqrt`' ändern den D-Stil in den D'-Stil. Die Stilarten sind natürlich nicht gottgegeben. Sie können sie mit den Befehlen

```
\displaystyle
\textstyle
\scriptstyle
\scriptscriptstyle
```

jederzeit selber frei wählen. Dazu zunächst ein etwas *dummes* Beispiel: Die Eingabe von

```
$$n+\scriptstyle n+\scriptscriptstyle n$$
```

erzeugt

$$n+\scriptstyle n+\scriptscriptstyle n$$

Man sieht aber schon, daß auch das Pluszeichen mit verkleinert wird. Etwas besser geeignet um die Unterschiede zu zeigen ist das folgende Beispiel.[2] Die Eingabe von

```
$$a_0+{1\over\displaystyle a_1+
          {\strut 1\over\displaystyle a_2+
            {\strut 1\over\displaystyle a_3+
              {\strut 1\over a_4}}}}$$
```

erzeugt

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{1}{a_4}}}}$$

Ohne die Befehle '`\displaystyle`' und '`\strut`' saähe das Ergebnis

$$a_0+\frac{1}{a_1+\frac{1}{a_2+\frac{1}{a_3+\frac{1}{a_4}}}}$$

aus. Die Nenner werden automatisch zentriert. Will man das verhindern, dann sollte man den Befehl '`\hfill`' benutzen. Damit sieht unser Serienbruch so

$$a_0+\cfrac[l]{1}{a_1+\cfrac[l]{1}{a_2+\cfrac[l]{1}{a_3+\cfrac[l]{1}{a_4}}}}$$

aus.

[2] Der Befehl `\strut` wird benutzt, um die Zähler etwas größer zu machen. In einem späteren Kapitel beschäftigen wir uns noch mehr mit dem *schönen* Satz von Formeln.

7.2.3 Weitere Befehle zur Stapelung

Neben dem Befehl '`\over`' zur Bildung von Brüchen existiert in TEX auch noch der Befehl '`\atop`', der einen 'Bruch' ohne Bruchstrich ausgibt.

`$$x\atop y+2$$` $${x \atop y+2}$$

Weiterhin existiert der Befehl '`\choose`'[3] der einen Binomealkoeffizienten ausgibt.

`$$n\choose k$$` $${n \choose k}$$

Auch Mischungen dieser Befehle sind erlaubt, vorausgesetzt, sie halten sich an die Regeln für Gruppierungen innerhalb von Formeln.

$${{n \choose k} \over 2}$$

$${n \choose {k \over 2}}$$

Es gibt noch einen Befehl, der zum Satz von Brüchen herangezogen werden kann, bei dem allerdings die Stärke des Bruchstrichs extra angegeben werden muß. Der Befehl heißt '`\above`' und wird z.B. folgendermaßen angewendet

```
$$\displaystyle{a\over b}\above1pt\displaystyle{c\over d}$$
```

Das Ergebnis sieht dann so aus:

$$\displaystyle{a\over b}\above1pt\displaystyle{c\over d}$$

7.2.4 Summen und Integrale

Summen und Integrale werden im Textstil und im Displaystil in unterschiedlichen Größen gesetzt, sie heißen auch *große* Symbole.[4]

`$\sum x_n$`	ergibt	$\textstyle\sum x_n$	(T-Stil)
`$$\sum x_n$$`	ergibt	$\displaystyle\sum x_n$	(D-Stil)

Bei den Summationszeichen werden meist Grenzen angegeben

$$\sum_{n=1}^{m} x_n$$

Die Grenzen werden in abgesetzten Formeln über und unter das Summensymbol gesetzt. in Textformeln werden sie neben das Symbol gesetzt: $\sum_{n=1}^{m} x_n$. Bei Integralen werden die Integrationsgrenzen immer neben das Integrationszeichen gesetzt.

$\textstyle\int_{-\infty}^{+\infty}$ Im T-Stil

$\displaystyle\int_{-\infty}^{+\infty}$ Im D-Stil

Mit `\limits` und `\nolimits` lßt sich das Setzen der Grenzen aber auch ganz individuell gestalten. Die Eingabe von:

```
$\sum\limits_{n=1}^m x_n
```

[3] Die Binomealkoeffizienten werden meist in der Statistik gebraucht, um die Anzahl der Möglichkeiten einer Wahl auszudrücken.

[4] Es gibt außer diesen beiden noch einige andere, die aber hier nicht weiter besprochen werden.

erzeugt in Textformeln: $\sum\limits_{n=1}^{m} x_n$ und die Eingabe von:

```
$$\int\limits_{-\infty}^{+\infty}$$
```

erzeugt:

$$\int\limits_{-\infty}^{+\infty}$$

Andererseits ergibt

```
$$\sum\nolimits_{n=1}^3 x_n$$
```

$$\sum\nolimits_{n=1}^3 x_n$$

Mit dem Befehl ‘`\displaylimits`’ werden die Grenzen *jedes* großen Symbols nur noch im D-Stil ober- resp. unterhalb des Symbols gesetzt, sonst immer dahinter.

Mehrzeilige Grenzen

Bei Mathematikern kommt es auch vor, daß die Angabe der Grenzen sich über mehrere Zeilen erstreckt. Auch das ist in TEX möglich. Die Formel:

$$\sum_{\scriptstyle 0\le i\le m\atop\scriptstyle 0<j<n} P(i,j)$$

wurde z.B. mit der Befehlsfolge

```
$$\sum_{\scriptstyle0\le i\le m\atop\scriptstyle0<j<n}P(i,j)$$
```

erzeugt.

7.2.5 Größenanpassung von Symbolen

Manche mathematischen Formeln sind ecklig lang. TEX bietet daher die Möglichkeit manche Symbole in ihrer Größe anzupassen. Die Eingabe von

```
$$\sqrt{1+\sqrt{1+\sqrt{1+
            \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$
```

erzeugt z.B.:

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$

Die drei größten Wurzelzeichen sind dabei untereinander identisch, bis auf die Tatsache, daß der Aufstrich des Wurzelzeichens durch unterschiedlich lange ‘|’ verlängert wurde. Die anderen Wurzeln sind eigene Zeichen. Analog ist eine Größenanpassung von Klammern möglich.

$$\left(\left(\left(\left(\Biggl(\biggl(\Bigl(\bigl((\;)\bigr)\Bigr)\biggr)\Biggr)\right)\right)\right)\right)\left\{\left\{\left\{\left\{\Biggl\{\biggl\{\Bigl\{\bigl\{\{\;\}\bigr\}\Bigr\}\biggr\}\Biggr\}\right\}\right\}\right\}\right\}$$

Klammern sind in Formeln sehr wichtig, deshalb bietet TEX eine ganze Reihe von ihnen an:

Eingabe	*Bedeutung*	*Ausgabe*
`(`	linke Klammer	$($
`)`	rechte Klammer	$)$
`[` oder `\lbrack`	linke eckige Klammer	$[$
`]` oder `\rbrack`	rechte eckige Klammer	$]$
`\{` oder `\lbrace`	linke geschweifte Klammer	$\{$
`\}` oder `\rbrace`	rechte geschweifte Klammer	$\}$
`\lfloor`	linke Bodenklammer	$\lfloor$
`\rfloor`	rechte Bodenklammer	$\rfloor$
`\lceil`	linke Deckenklammer	$\lceil$
`\rceil`	rechte Deckenklammer	$\rceil$
`\langle`	linke einfach eckige Klammer	$\langle$
`\rangle`	rechte einfach eckige Klammer	$\rangle$
`/`	Schrägstrich	$/$
`\backslash`	Gegenschrägstrich	$\backslash$
`|` oder `\vert`	Gerader Strich	$\vert$
`\\|` oder `\Vert`	Doppelter gerader Strich	$\Vert$
`\uparrow`	Pfeil nach oben	$\uparrow$
`\downarrow`	Pfeil nach unten	$\downarrow$
`\Uparrow`	Doppelpfeil nach oben	$\Uparrow$
`\Downarrow`	Doppelpfeil nach unten	$\Downarrow$
`\updownarrow`	Pfeil nach oben und unten	$\updownarrow$
`\Updownarrow`	Doppelpfeil nach oben und unten	$\Updownarrow$

Um eine größere Fassung der gezeigten Symbole zu erreichen muß ihnen nur der Befehl '`\bigl`' (für linke Klammersymbole) bzw. '`\bigr`' (für rechte) vorangestellt werden. Die Eingabe von

```
$\bigl(x-s(x)\bigr)\bigl(y-s(y)\bigr)$
```

erzeugt: $\bigl(x-s(x)\bigr)\bigl(y-s(y)\bigr)$, die Eingabe von

```
$\bigl\lfloor\sqrt A\bigr\rfloor$
```

erzeugt: $\bigl\lfloor\sqrt A\bigr\rfloor$.

Sehen wir uns doch einmal die verschiedenen Symbol im Vergleich an: Zunächst die normalen mit den ge'`\big`'ten:

$$( ) [ ] \{ \} \lfloor \rfloor \lceil \rceil \langle \rangle / \backslash | \| \uparrow \Uparrow \downarrow \Downarrow \updownarrow \Updownarrow$$
$$\big(\big) \big[\big] \big\{ \big\} \big\lfloor \big\rfloor \big\lceil \big\rceil \big\langle \big\rangle \big/ \big\backslash \big| \big\| \big\uparrow \big\Uparrow \big\downarrow \big\Downarrow \big\updownarrow \big\Updownarrow$$

Jetzt können sie auch noch '`\Big`x sagen (das 'x' steht für 'l' oder 'r'):

$$\Big(\Big) \Big[\Big] \Big\{ \Big\} \Big\lfloor \Big\rfloor \Big\lceil \Big\rceil \Big\langle \Big\rangle \Big/ \Big\backslash \Big| \Big\| \Big\uparrow \Big\Uparrow \Big\downarrow \Big\Downarrow \Big\updownarrow \Big\Updownarrow$$

Wenn ihnen das noch nicht genügt, gibt es die Befehle '`\bigg`'x

$$\bigg(\bigg) \bigg[\bigg] \bigg\{ \bigg\} \bigg\lfloor \bigg\rfloor \bigg\lceil \bigg\rceil \bigg\langle \bigg\rangle \bigg/ \bigg\backslash \bigg| \bigg\| \bigg\uparrow \bigg\Uparrow \bigg\downarrow \bigg\Downarrow \bigg\updownarrow \bigg\Updownarrow$$

Und zu guter Letzt noch '`\Bigg`'x

$$\Bigg(\Bigg) \Bigg[\Bigg] \Bigg\{ \Bigg\} \Bigg\lfloor \Bigg\rfloor \Bigg\lceil \Bigg\rceil \Bigg\langle \Bigg\rangle \Bigg/ \Bigg\backslash \Bigg| \Bigg\| \Bigg\uparrow \Bigg\Uparrow \Bigg\downarrow \Bigg\Downarrow \Bigg\updownarrow \Bigg\Updownarrow$$

Neben den 'l' und 'r' Formen dieser Befehle bietet TEX auch noch eine 'm' Variante an, also z.B. '`\bigm`' oder '`\Biggm`'. Bei dieser Befehlsform wird das entsprechend folgende Symbol nicht wie eine Klammer, öffnend oder schließend, gesetzt, sondern wie eine Relation. Welches Ergebnis am besten aussieht hängt vom Einzelfall ab. Als Beispiel diene hier die Eingabe von

```
$$\bigl(x\in A(n)\bigm|\||x\in B(n)\bigr)$$
```

was

$$(x \in A(n) \mid \||x \in B(n))$$

erzeugt. Außerdem gibt es noch die Möglichkeit die Befehl *ohne* Links-, oder Rechtsangabe zu benutzen. Dann wird das Symbol wie eine normale Variable gesetzt. Dies empfiehlt sich meist nur beim Schräg- oder Gegenschrägstrich.

```
$${a+1\over b}\bigg/{c+1\over d}$$
```

ergibt

$${a+1\over b}\bigg/{c+1\over d}$$

Die Rechts-Links-Befehle

TEX bietet noch eine einfache Möglichkeit die Klammersymbole der Größe der Formel anzupassen. Die Befehl '`\left`' und '`\right`' gefolgt von einem Klammersymbol sorgen selbständig für die richtige Größe des Klammersymbols. Die Eingabe von

```
$$1+\left(1\over1-x^2\right)^3$$
```

ergibt in der Ausgabe

$$1+\left(\frac{1}{1-x^2}\right)^3$$

In einer einfachen Formel, wie '`$\left(x\right)$`', werden nur die einfachen Klammersymbole gewählt und die Ausgabe ist (x). Die Befehle '`\left`' und '`\right`' müssen immer in einer Gruppe stehen. Konstruktionen wie

```
$$\left( ... { ... \right) ... }$$
```

sind nicht erlaubt. Die Befehle stellen allerdings *ihrerseits* selber eine Gruppierung dar, wie das obige Beispiel zeigt. Das '1+' wird nicht in den Bruch einbezogen. Daraus folgt auch, daß immer zu jedem '`\left`'-Befehl ein entsprechender '`\right`'-Befehl geschrieben werden muß. Die Klammersymbole müssen allerdings nicht zueinander passen, wie das folgende Beispiel zeigt.

```
$$\left(1+n\atop 1-n\right[$$
```

ist erlaubt und ergibt

$$\left({1+n \atop 1-n}\right[$$

Jetzt fragt man sich, wieso man überhaupt noch die Einzelbefehle für die Größe der Klammersymbole benutzen soll, wenn doch TEX das so schön selber kann. Nun, es können drei Situationen auftreten, wo die Links-Rechts-Befehle eben doch nicht so hundertprozentig passen.

1. Manchmal wählt TEX eine Klammer, die für diese Situation zu klein ist. Wenn man z.B.

   ```
   $\left|\left|x\right|+\left|y\right|\right|$
   ```

   eingibt, dann ist das Ergebnis $||x|+|y||$, obwohl man durch explizite Eingabe der Größen, das wesentlich bessere Ergebnis $\bigl||x|+|y|\bigr|$ bevorzugen würde.

2. Andererseits kann es aber auch vorkommen, daß TEX zu große Symbole aussucht. Dies passiert z.B. bei großen Symbolen. Vergleichen sie einmal

   ```
   $$\left(\sum_{k=1}^n A_k\right)$$
   ```

mit

```
$$\biggl(\sum_{k=1}^n A_k\biggr)$$
```

Und nun die Ausgabe

$$\left(\sum_{k=1}^n A_k\right) \qquad \biggl(\sum_{k=1}^n A_k\biggr)$$

3. Schließlich kann es vorkommen, daß sie eine Formel über mehr als eine Zeile schreiben müssen. Sie werden dann Probleme bekommen die zusammengehörigen Befehle innerhalb einer Gruppe zu realisieren. Auch in diesem Fall müssen sie die Größenanpassung selber vornehmen.

Es gibt übrigens für die Links-Rechts-Befehle auch eine *leere* Klammer, die bei Anwendungen, wie

$$|x| = \left\{ \begin{array}{ll} x & \text{für} \quad x \geq 0 \\ -x & \text{für} \quad x < 0 \end{array} \right.$$

nötig ist. Zu jedem '`\left`'-Befehl muß ja auch der passende '`\right`'-Befehl geschrieben werden. Man gibt hier bei dem '`\right`'-Befehl statt eines Klammersymbols einen Punkt an. Im obigen Beispiel stand am Ende einfach

```
...x<0\right.$$
```

Manchen sie sich keine Sorgen über die zweizeilige Formel, wie so etwas geht wird auch noch besprochen. Die *leere* Klammer ist übrigens nicht leer, statt dessen wird von TEX noch der Zwischenraum, der in der Variablen '`\nulldelimiterspace`' steht, eingefügt (Voreingestellt ist 1.2pt).

Teilklammern

Weiter oben wurden einige Symbole gezeigt, die aus Teilzeichen zusammengesetzt waren. Diese Teilzeichen kann man natürlich auch einzeln verwenden. Mit den Befehlen

```
\arrowvert
\Arrowvert
\bracevert
```

werden einfache, bzw. doppelte senkrechte Striche gezogen, mit den Befehlen

```
\lgroup
\rgroup
\lmoustache
\rmoustache
```

werden Teile der geschweiften Klammer benutzt. Die Klammern sehen folgendermaßen aus

$$\Big\arrowvert \cdots \Big\Arrowvert \cdots \Big\bracevert \cdots \Big\lgroup \cdots \Big\rgroup \cdots \Big\lmoustache \cdots \Big\rmoustache$$

Diese Klammern sind nur als ziemlich große Symbole (größer als Big) verfügbar.

Mitunter möchten sie selber irgendwelche Symbole definieren, und in unterschiedlichen Situationen benutzen. Dafür bietet TEX den Befehl

```
\mathchoice{<Teil>}{<Teil>}{<Teil>}{<Teil>}
```

Die '`<Teil>`'e sind dabei Teilformeln, die folgendermaßen benutzt werden: Die erste Teilformel wird benutzt wenn der Stil D oder D' aktiv ist, die zweite, wenn der Stil T oder T' aktiv ist, wie es weitergeht können sie sich denken. Beachten sie im Anhang auch den Befehle '`\mathpalette`', der in diesem Zusammenhang interessant ist.

7.2.6 Die Achse einer Formel

Wenn sie sich einmal eine Formel mit Klammern genau ansehen, dann werden sie feststellen, daß die Klammern bezüglich einer unsichtbaren Achse zentriert sind. Diese Achse heißt 'Achse der Formel' und liegt bei Textformeln so: ——. Z.B. jeder Bruchstrich liegt auf dieser Achse. Mitunter ist es notwendig eine vertikale Box auf dieser Achse zu zentrieren.[5] TEX bietet natürlich auch hierfür einfache Befehle. Geben sie einfach

```
\vcenter{<vertikales Material>}
```

ein. Das vertikale Material schreiben sie so, als handele es sich um eine `\vbox`. Den Rest erledigt TEX. Auch solche Konstruktionen wie '`\vcenter to...`' oder '`\vcenter spread...`' sind möglich. Übrigens, auch alle anderen Boxkonstruktionen, inklusive '`\raise`' oder '`\lower`' sind im mathematischen Modus möglich und erlaubt.

7.3 Die Interna

Dieser Abschnitt ist zum eigentlichen Verständnis von TEX im mathematischen Modus nicht notwendig, er beschäftigt sich mit den Interna der Zeichenauswahl in mathematischen Formeln.

7.3.1 Familien

Jedes Zeichen, das im mathematischen Modus ausgegeben wird, gehört zu einer von sechzehn Familien, die ihrerseits aus drei Zeichensätzen bestehen. Mit den Befehlen '`\textfont`', '`\scriptfont`' und '`scriptscriptfont`' wählen dabei jeweils eine dieser Zeichensätze. Die Familie 0 wird in *Plain* TEX für die Roman Zeichensätze verwendet und die Zuweisungen lauten

```
\textfont0=\tenrm
\scriptfont0=\sevenrm
\scriptscriptfont0=\fiverm
```

Wird kein Zeichensatz angegeben, verwendet TEX '`\nullfont`'.[6] Die Schriftfamilie wird erst *nach* Abarbeitung der Formel ausgewertet. Angenommen, sie haben auf ihrem Rechner eine Helevetica-schrift zur Verfügung, dann erhalten sie nach der Anweisung

```
$\textfont0=\tenrm 9 \textfont0=\helevetica 9$
```

zwei Neuen in der Schriftart Helevetica, da diese Anweisung am Ende der Formel gültig ist. Anders sieht es aus, wenn sie

```
$\textfont0=\tenrm 9\hbox{$9\textfont0=\helevetica$}$
```

eingeben, dann erhalten sie die erste Neun in 'tenrm' und die zweite in 'helevetica'.

7.3.2 Zeichenklassen

Zu jedem mathematischen Zeichen gibt es einen Code zwischen 0 und 4096, der sich aus der Positionsnummer innerhalb des Zeichensatzes plus dem 256fachen der Familiennummer ergibt. Am einfachsten wird das hexadezimal notiert. Die Kodierung '`24A`' bedeutet, daß es sich um das `2A`te Zeichen in der Familie 2 handelt. Außer der Familienzugehörigkeit und der Position innerhalb eines Zeichensatzes gibt es aber auch noch eine Klassifizierung der Zeichen.

[5] Bei dem $|x| = \{\ldots$ Beispiel von eben war das z.B. nötig

[6] Haben sie sich nicht schon gefragt, wofür dieser Zeichensatz benötigt wird?

Klasse	*Bedeutung*	*Beispiel*
0	Normal	/
1	Großes Symbol	$\sum$
2	Binärer Operator	$+$
3	Relation	$=$
4	'Öffner'	(
5	'Schließer'	)
6	Interpunktion	,
7	Variable Familie	x

Die Klassen 0 bis 6 besagen, welchem Sprachteil das Zeichen zugehört, die Klasse 7 wird sofort besprochen. Die Klassenkodierung wird mit 4096 multipliziert der Zeichenkodierung zuaddiert, so daß sich eine vierstellige hexadezimale Zahl ergibt. `1350` bedeutet: Klasse `1` (Großes Symbol), in der Familie `3`, und dort das Zeichen `50` (Alle Zahlangeben hexadezimal).

Die Klasse 7 erlaubt es Zeichen in verschiedenen Familien aufzutauchen. Normalerweise verhalten sich Zeichen der Klasse 7 wie die der Klasse 0, es sein denn die Integervariable '`\fam`' hat einen gültigen Familienwert $(0 \ldots 15)$. Normalerweise wird der Wert auf -1 gesetzt, kann aber geändert werden. *Plain* TeX benutzt diese Möglichkeit für den Befehl '`\rm`'. Er ist eine Abkürzung für die Befehle '`\fam=0`' und '`\tenrm`'. Die Familie aller Zeichen (haben normalerweise die Klasse 7) wird also 0 und die Schrift `tenrm` wird als voreingestellte Schrift verwendet. Mit diesem Befehl ist es dann möglich innerhalb mathematischer Formeln normalen Text zu setzen.

7.3.3 Mathcode

Von einer Tabelle mit 256 "mathcode"-Werten hängt die Interpretation der Zeichen im mathematischen Modus ab. Der zu einem Zeichen gehörige Wert kann mit dem Befehl '`\mathcode`' geändert werden.[7] Der Befehl

```
\mathcode‘<=\"313C
```

bedeutet, daß dem Zeichen '`<`' der Code '`313C`' zukommt, also 3te Klasse in der 1ten Familie und dort Zeichen Nummer '`3C`'. Der Buchstabe 'b' wird z.B. mit '`7162`' kodiert. Es gibt noch den 'mathcode' '`8000`', der zur Folge hat, daß das Zeichen wie ein aktives Zeichen (catcode=13) behandelt wird.

Analog zu dem Befehl '`\char`' gibt es auch einen Befehl um jedes mathematische Zeichen ansprechen zu können, und der heißt, wie nicht anders zu erwarten war, '`\mathchar`'. Die Definition des Summenzeichens wäre also möglich mit der Befehlsfolge

```
\def\sum=\mathchar\"1350
```

(Klasse 1, großes Symbol; Familie 3, Zeichen 50) Aber es gibt auch noch eine bessere Möglichkeit, denn genau, wie es den Befehl '`\chardef`' gibt, gibt es auch den Befehl '`\mathchardef`' und damit reduziert sich die Aufgabe zu

```
\mathchardef\sum=\"1350
```

TeX kann auch einer ganzen Teilformel eine Klasse zukommen lassen. Dazu dienen die Befehle

```
\mathord
\mathop
\mathbin
\mathrel
\mathopen
\mathclose
\mathpunkt
```

[7] Entspricht soweit dem Befehl `\catcode`, der schon behandelt wurde.

Sie beziehen sich immer auf das Zeichen, oder die gruppierte Teilformel, die dahinter steht. Die Anweisung

```
$G\mathbin:H$
```

sorgt dafür, daß der Doppelpunkt wie ein Zeichen eines binären Operators verwendet wird. Der Befehl '`\mathord`' ist eigentlich überflüßig, da alles in Klammern wie ein normales Symbol verwendet wird. Die beiden folgenden Anweisungen sind also gleichwertig

```
$1\mathord,2345$
$1{,}2345$
```

Es gibt noch eine achte Klassifizierung: '`\mathinner`', die für spezielle Zwecke gebraucht wird. Brüche und Rechts-Links-Konstruktionen werden als *inner* bezeichnet, und mit etwas zusätzlichem Leerraum umgeben. Normalerweise werden Teilformeln als normales Symbol angesehen.

7.3.4 Delcode

Klammern werden noch weiter spezifiziert, sie haben neben dem `catcode` und dem `mathcode` auch noch einen sechstelligen, hexadezimalen `delcode`.[8] Die ersten drei Ziffern geben dabei das *kleine*, die zweiten drei Ziffern das *große* Symbol an. Die Anweisung

```
\delcode‘(=\"028300 \delcode‘.=0
```

besagt, daß für die öffnende Klammer als kleines Symbol das `28`te Zeichen aus der Familie `0` in Frage kommt, und die große Variante in Familie `3` an der Position `0` zu finden ist. Im Falle des Punktes wird keine große und keine kleine Variante angegeben, so daß es auch zu keiner Ausgabe kommt.[9] Eine Klammer kann auch direkt mit dem Befehle '`\delimiter`' angegeben werden. Der zugehörige Zahlwert muß nur kleiner als '`\"7FFFFFF`' sein. Die Angabe enthält sofort auch die Klassenangabe.

```
\def\langle{\delimiter\"426830A}
```

Bedeutet, daß '`\langle`' ein öffnendes Symbol ist, dessen kleine Variante bei '`268`' zu finden ist, und die große Variante bei '`30A`'. Tritt der '`\delimiter`' Befehl nach einem '`\left`' oder einem '`\right`' Befehl auf, dann wird die Klassenangabe ignoriert, in anderem Kontext werden die letzten drei Stellen gestrichen und die ersten vier Ziffern wie bei einer '`\mathchar`' Angabe verwendet. Die folgenden Angaben sind also äquivalent

```
$\langle x$
$\mathchar\"4268 x$
```

Andere ähnliche Befehle

Wie auch der '`\delimiter`' Befehl arbeiten die Befehle '`\radical`' und '`\mathaccent`', die dazu dienen Wurzelzeichen, bzw. mathematische Akzente zu definieren. Die Kodierung ist auch hier wieder dieselbe, wie schon in den vorherigen Beispielen.

7.3.5 Familienzugehörigkeit nach *Plain* TEX

Plain TEX setzt die Familien normalerweise wie folgt: Familie 1 für die Italic-Schriften, Familie 2 für normale mathematische Symbole, Familie 3 für große Symbole. Die Familien 2 und 3 sind meist mit den Zeichensätzen '`cmsy`' und '`cmex`' verbunden, die noch weitere Informationen zum Satz beinhalten.

[8] Vom englischen *Delimiter*.

[9] Erinnern sie sich an die Bedeutung des Punktes in Zusammenhang mit Klammern?

7.3.6 Elemente einer mathematischen Liste

Sie können sich vermutlich schon denken, daß auch mathematische Formeln von TEX zunächst wie eine lange Liste von Elementen angesehen wird, wie es auch schon bei der horizontalen und der vertikalen Liste der Fall war. In einer mathematischen Liste können nun folgende Elemente auftreten.

1. Ein Atom (Siehe unten)
2. Horizontales Material (Ein Strich, eine Trennung, eine Strafe oder ein "WasDenn")
3. Vertikales Material
4. Etwas Leim
5. Ein Kern
6. Ein Stilwechsel (von den Befehlen `\displaystyle` o.ä.)
7. Ein Bruch o.ä.
8. Ein Bündel (meist eine Teilformel, die z.B. in Klammern gesetzt ist)
9. Eine Mathe-Auswahl von `\mathchoise`

7.3.7 Atome

Das wichtigste Element in einer mathematischen Liste ist das sog. Atom. Es besteht aus einem *Körper*, einer *Hochstellung* und einer *Tiefstellung*. Wenn sie z.B. eingeben

```
(x_i+y)^{\overline{n+1}}
```

dann erhalten sie eine Folge von fünf Atomen: (, x_i, +, y und $)^{\overline{n+1}}$. Die Körper dieser Atome sind: (, x, +, y und), ihre Hochstellungen sind leer, bis auf die Hochstellung des letzten Atoms, die aus $\overline{n+1}$ besteht, ihre Tiefstellungen sind auch leer, bis auf die des zweiten Atoms, welche aus i besteht. Es gibt verschiedene Atomtypen.

Ord	Ein einfaches Atom wie x
Op	Ein großes Atom wie $\sum$
Bin	Ein binäres Atom wie $+$
Rel	Eine Relation wie $=$
Open	Ein öffnendes Atom wie (
Close	Ein schließendes Atom wie)
Punct	Ein Interpunktionsatom wie ,
Inner	Ein inneres Atom wie $\frac{1}{2}$
Over	Eine überstrichenes Atom wie $\overline{x}$
Under	Ein unterstrichenes Atom wie $\underline{x}$
Acc	Ein Akzentatom wie $\hat{a}$
Rad	Ein Wurzelatom wie $\sqrt{2}$
Vcent	Eine vertikale Box, von `\vcenter`

Der Körper, die Hochstellung und die Tiefstellung eines Atoms heißen *Felder*. Für diese Felder gibt es vier Möglichkeiten, ein Feld enthält

1. nichts
2. ein mathematisches Symbol
3. eine Box

4. eine mathematische Liste

Den genauen Aufbau dieser Atome können sie sich mit '`\showlists`' ansehen. Manche Atome haben übrigens weitere Informationen, so sind z.B. alle 'Op' Atome mit der zusätzlichen Information über '`\limits`' oder '`\nolimits`' versehen.

7.4 Schöne Aussichten

Es sind nun schon fast alle Möglichkeiten von TEX besprochen, eine Formel zu setzen, wenn man allerdings mit der Zeit etwas Übung darin bekommen hat, zu erkennen, wie eine Formel aussieht, wenn man sie schreibt, dann kommt der Wunsch auf Formeln nicht nur einfach zu setzen, sondern ihr Aussehen noch positiv zu beeinflußen. Hier nun einige Regeln dafür.

7.4.1 Interpunktion

Punkt und Kommata sollten in Textformeln nicht zwischen die '`$`'s geschrieben werden, wenn sie nicht zur Formel gehören. Eine Textformel

```
f\"ur $x=a, b$, oder $c$
```

schreibt man besser als

```
f\"ur $x=a$, $b$, oder~$c$
```

Der Grund hierfür liegt darin, daß *in* einer Formel die Abstände hinter Interpunktionszeichen anders gesetzt werden, als im Text, man erhielte also unterschiedlich breite Leerräume, was höchst unschön aussieht. Außerdem bricht TEX eine Formel hinter einem Interpunktionszeichen nur *sehr* ungerne um, was aber in diesem Fall gerade wünschenswert wäre.[10]

Innerhalb einer abgesetzten Formel sieht die Sache etwas anders aus. Hier sollte das Interpunktionszeichen *in* die Formel geschrieben werden. Der Grund wird sofort klar, wenn man sich die beiden folgenden Beispiele ansieht.

```
$$f(a,b).$$
```

ergibt das gewünschte

$$f(a,b).$$

wohingegen

```
$$f(a,b)$$.
```

etwas seltsam aussieht:

$$f(a,b)$$

. Achten sie darauf, wo der Punkt hingekommen ist.

Mehr Platz

Man kann Textformeln auch dadurch hervorheben, daß man ihnen etwas mehr Platz verschafft. TEX bietet diese Möglichkeit mit dem Befehl '`\mathsurround`'. Die Textformel

```
$x=a$, $b$, oder $c$
```

sieht mit den Vorgaben für '`\mathsurround`' für ein, bzw. zwei pt folgendermaßen aus

$x = 1$, b, oder c

$x = 1$, b , oder c

Normalerweise wird der Wert auf 0pt gesetzt.

[10] Der Grund für das Verhalten von TEX liegt darin, daß man ja üblicherweise nicht eine Formel wie $f(a,b)$ in $f(a$, und $b)$ getrennt haben möchte.

7.4.2 Buchstaben in Nicht-Italic

Generelles

Normalerweise werden Buchstaben in Formeln immer in der Schriftart Italic gesetzt. Bei Funktionsnamen o.ä. ist dies aber störend. TEX bietet daher einige Befehle, die die entsprechenden Funktionsnamen in der Schriftart Roman ausgeben.

`\arccos`	`\cos`	`\csc`	`\exp`	`\ker`	`\limsup`	`\min`	`\sinh`
`\arcsin`	`\cosh`	`\deg`	`\gcd`	`\lg`	`\ln`	`\Pr`	`\sup`
`\arctan`	`\cot`	`\det`	`\hom`	`\lim`	`\log`	`\sec`	`\tan`
`\arg`	`\coth`	`\dim`	`\inf`	`\liminf`	`\max`	`\sin`	`\tanh`

Hier einige Beispiele, die auch zeigen, daß immer der richtige Leerraum um die Funktionsnamen gesetzt wird.

Eingabe	*Ausgabe*
`$\sin2\theta=2\sin\theta\cos\theta$`	$\sin 2\theta = 2\sin\theta\cos\theta$
`$O(n\log n\log \log n)$`	$O(n\log n\log\log n)$
`$\Pr(X>x)=\exp(-x/\mu)$`	$\Pr(X>x)=\exp(-x/\mu)$
`$$\max_{1\le n\le m}\log_2P_n$$`	$\max\limits_{1\le n\le m}\log_2 P_n$
`$$\lim_{x\to0}{\sin x\over x}=1$$`	$\lim\limits_{x\to 0}\frac{\sin x}{x}=1$

Wie man am Beispiel des '`\max`'-Operators sieht werden manche dieser Kontrollsequenzen als große Symbole behandelt.

Mitunter reichen diese Befehle nicht aus. Sollten sie nur ab und zu einen anderen Befehle benutzen müssen, dann sollten sie es per Hand machen, tritt die Situation allerdings häufiger auf, dann sollten sie sich einen entsprechenden Befehl schaffen. Hier zunächst die Möglichkeit als solche Text in Formeln in einer anderen Schriftart zu setzen.

Am einfachsten erreichen sie das gewünschte Ziel, wenn sie den Befehl '`\rm`' verwenden. Die Eingaben von

```
$$\sqrt{{\rm Var}(X)}$$
$$x_{\rm max}-x_{\rm min}$$
```

führen zu dem Ergebnis

$$\sqrt{\mathrm{Var}(X)}$$

und

$$x_{\mathrm{max}}-x_{\mathrm{min}}$$

Denken sie allerdings daran, daß sie Leerzeichen in derartigen Texten mit der Kontrollsequenz '`\␣`' schreiben, da sie sonst ignoriert würden. Es geht auch mit dem Befehle '`\hbox`', aber Achtung, es können Fehler auftreten.

1. Wenn sie den `\hbox` Befehl verwenden, wird die Schriftgröße nicht geändert. Die Eingabe von `$x_{\hbox{min}}$` ergibt die Ausgabe: $x_{\hbox{min}}$.

2. Es ist nicht unbedingt sichergestellt, daß sie auch wirklich die Schriftart erreichen, die sie haben wollen. Der Text in der `\hbox` wird in der Schrift gesetzt, die außerhalb der Formel gültig ist. Wenn sie also z.B. die Schriftart Boldface benutzen, dann würde auch ihr Text in Boldface gesetzt.

Moduln

Für Modulnangaben existieren zwei Kontrollsequenzen. Einmal der Befehl '`\bmod`', der wie ein binärer Operator gesetzt wird und innerhalb von Formeln verwendet werden sollte, und andererseits der Befehl '`\pmod`', der am Ende von Formeln verwendet werden sollte, der wie eine Klammer gesetzt wird. Der Befehl '`\pmod`' setzt übrigens eigene Klammern, die Eingabe von

```
$$x\equiv y+1\pmod{m^2}$$
```

ergibt

$$x \equiv y+1 \pmod{m^2}$$

Ändern der Schriftart

Änderungen der Schriftart durch die normalen Befehle wirkt sich nur auf die normalen Buchstaben, die griechischen Großbuchstaben und die Akzente aus. Die Dollarzeichen, die eine Formel umschließen wirken dabei wie eine Gruppenklammer, so daß die Schriftartänderung lokal bleibt.

`$\bf a+b=\Phi_m$` ergibt $\mathbf{a+b=\Phi_m}$

Neben den üblichen Umschaltbefehlen existiert im mathematischen Modus auch Befehl '`\cal`', der kalligraphische Buchstaben erzeugt, allerdings *nur* Großbuchstaben. Die Angabe von '`$\cal A$`' ergibt $\mathcal{A}$.

Ein weiterer Befehl ist '`\mit`', der für "mathematisches Italic" steht. Mit diesem Befehle werden auch die griechischen Großbuchstaben in Italic ausgegeben. Auf die normalen Buchstaben hat dieser Befehl keinen Einfluß, genau, wie der Befehl '`\rm`' auf die griechischen Großbuchstaben keinen Einfluß hat. Worin besteht nun der Unterschied zwischen mathematischem Italic und normalem Italic, am besten zeigt das ein Beispiel

`$Dies\ ist\ mathematisches\ Italic$`	$Dies\ ist\ mathematisches\ Italic$
`{\it Das ist Text Italic}`	*Das ist Text Italic*

Die mathematischen Buchstaben sind etwas breiter und sie werden mit anderem Leerraum gesetzt.

7.4.3 Platz zwischen Formeln

Mitunter sollen, vor allem in abgesetzten Formeln, mehr als eine Formel gesetzt werden, die voneinander durch etwas Platz getrennt werden sollen. Der normale Abstand hilft hier nicht weiter. Der Formelsatz

$$F_n = F_{n-1} + F_{n-2}, n \geq 2$$

ist nicht gerade das, was man erwarten würde. Grundsätzlich ist es empfehlenswert derartige Formeln durch Text zu trennen, also

$$F_n = F_{n-1} + F_{n-2}, \text{ wenn } n \geq 2$$

es geht aber auch mit Zwischenraum, wenn man sie einer der beiden Befehle '`\quad`' oder '`\qquad`' bedient. Der Befehl '`\quad`' schafft etwas den Platz eines em, der andere Befehl etwa das doppelte. Die Formel

```
$$F_n=F_{n-1}+F_{n-2},\qquad n\ge2$$
```

sieht dann so aus

$$F_n = F_{n-1} + F_{n-2}, \qquad n \geq 2$$

Derartigen Platz müssen sie immer explizit mit solchen Befehlen angeben.

7.4.4 Platz *in* Formeln

Meist ist die Positionierung von Formelteilen in einer Formel hinreichend gut. Es kann aber vorkommen, daß die Vorgabe von TEX doch nicht ganz den Erwartungen entspricht. Es wäre unsinnig mit den Befehlen '`\quad`', '`\qquad`' oder '`\␣`' Platz zu schaffen, da diese Befehle viel zu große Zwischenräume schaffen. In Formeln werden von TEX drei verschiedene Abstände benutzt. Ein großer Abstand wird rechts und links von Gleichheitszeichen verwendet ($2 = 3 - 1$), mittleren Abstand wählt TEX vor und hinter binären Zeichen, wie dem Minuszeichen, der kleinste Abstand ist noch kleiner, er macht gerade den Unterschied zwischen 'loglog' und '$\log\log$' aus.

Sie können diese Zwischenräume auch selber in Formeln schreiben, wenn sie die Befehle

`\,`	Kleiner Zwischenraum (1/6 eines quad)
`\>`	Mittlerer Zwischenraum (2/9 eines quad)
`\;`	Großer Zwischenraum (5/18 eines quad)
`\!`	negativer kleiner Zwischenraum

Die Größenangaben sind nur im Normalfall richtig, da die Abstände in *muglue* definiert sind. Die Bezeichnung *glue*=Leim, kennen wir bereits, die Vorsilbe *mu* bedeutet 'mathematic units'. 18 mu machen ein em aus, je nach aktueller Schriftart ändern sich also auch die Abstände. Die Definition der Abstände lautet

```
\thinmuskip = 3mu
\medmuskip = 4mu plus 2mu minus 4mu
\thickmuskip = 5mu plus 5mu
```

Sie können den Platz, den sie in eine Formel zusätzlich hineinbringen wollen auch sofort in muglue angeben, dazu dient der Befehl '`\muskip`', der wie der `hskip` Befehl arbeitet, allerdings verlangt er als Argument *immer* mu's.

Beispiele der Anwendung

Als erstes Beispiel dient der dx Ausdruck am Ende eines Integrals. Normalerweise würde er zu nahe an den Inhalt des Integrals gesetzt.

`æ\int_0^\infty f(x)\,dx$`	$\int_0^\infty f(x)\,dx$
`$x\,dy/dx$`	$x\,dx/dy$

Im letzten Beispiel ist der Befehl vor dem dy natürlich überflüßig. Auch Einheiten sollten derartig abgesetzt werden

`$55\rm\,km/h$`	$55\,\mathrm{km/h}$
`$g=9,8\rm\,m/sec^2$`	$g = 9,8\,\mathrm{m/sec^2}$

Auch hinter Fakultätszeichen, hinter denen noch etwas folgt, sollten sie etwas zusätzlichen Zwischenraum setzen

`$3!\,4!\,5!$`	$3!\,4!\,5!$

Weitere Beispiele sehen sie nun

`$\sqrt2\,x$`	$\sqrt2\,x$
`$\sqrt{\,\log x}$`	$\sqrt{\,\log x}$
`$O\bigl(1/\sqrt n\,\bigr)$`	$O\bigl(1/\sqrt n\,\bigr)$
`$[\,0,1)$`	$[\,0,1)$
`$$\int\!\!\!\int_D dx\,dy$$`	$\int\!\!\!\int_D dx\,dy$

In der folgenden Tabelle werden alle Zwischenräume erklärt, die TEX automatisch zwischen Formelatomen setzt. Dabei stellen 0, 1, 2 und 3 keinen, kleinen, mittleren und großen Zwischenraum dar. Die Angaben stehen in Klammern, wenn der Zwischenraum nur im Display- und im Textstil gesetzt wird und sonst nicht. Wenn ein Stern in der Tabelle auftaucht, dann ist damit eine Situation gemeint, die nie auftreten kann.

		Rechtes Atom							
		Ord	Op	Bin	Rel	Open	Close	Punct	Inner
	Ord	0	1	(2)	(3)	0	0	0	(1)
	Op	1	1	*	(3)	0	0	0	(1)
	Bin	(2)	(2)	*	*	(2)	*	*	(2)
Linkes	Rel	(3)	(3)	*	0	(3)	0	0	(3)
Atom	Open	0	0	*	0	0	0	0	0
	Close	0	1	(2)	(3)	0	0	0	(1)
	Punct	(1)	(1)	*	(1)	(1)	(1)	(1)	(1)
	Inner	(1)	1	(2)	(3)	(1)	0	(1)	(1)

Die Regeln von TEX sind manchmal etwas fehlerhaft, besonders, wenn '`|`' und '`\|`' in einer Formel auftreten, da die Zeichen $|$ und $\|$ als normale Symbole und nicht als Klammern aufgefaßt werden.

<table>
<tr><td>Eingabe</td><td>Ausgabe</td></tr>
<tr><td><code>$|-x|=|+x|$</code></td><td>$| - x| = | + x|$</td></tr>
<tr><td><code>$\left|-x\right|=\left|+x\right|$</code></td><td>$\left|-x\right| = \left|+x\right|$</td></tr>
</table>

7.4.5 Punkte

Mathematiker brauchen öfters Punkte in ihren Formeln, um weitergehende Folgen von Elementen anzuzeigen. Die Eingabe von '`$x...y$`' führt zu dem unschönen Ergebnis $x...y$. TEX bietet statt dessen die Befehle '`\ldots`' und '`\cdots`' an, die drei Punkte auf der Basislinie, resp. etwas darüber setzen. Der erste Fall bietet sich bei Elementen an, die durch Kommata getrennt sind, z.B. $x_1, \ldots, x_n$, der zweite Befehl dient der Darstellung von Elementen, die per Operation verknüpft sind, wie $x_1 + \cdots + x_n$. Sie sollten allerdings daran denken, daß diese Befehle nicht genügend Leerraum lassen, speziell am Ende einer Formel, oder vor Klammern. Um das korrekte Ergebnis

$$(1-x)^{-1}=1+x+x^2+\cdots\,.$$

zu erhalten, muß man

```
$$(1-x)^{-1}=1+x+x^2+\cdots\,.$$
```

eingeben. Da dieser Fall oft vorkommt, bietet TEX für die Befehlsfolge '`\ldots\,`' die einfache Abkürzung '`\dots`' an.

7.4.6 Zeilenumbrüche

Auch mathematische Formeln können in Zeilen umgebrochen werden. TEX trennt dabei nur nach einer Relation oder einem binären Symbol, solange diese nicht in einer Gruppe oder einem Bruch stehen. Die Formel

```
$f(x,y) = x^2-y^2 = (x+y)(x-y)$
```

wird vorzugsweise bei den Gleichheitszeichen umgebrochen, wenn das nicht geht, eventuell auch noch nach dem Plus- bzw. Minuszeichen. Wenn man erreichen will, daß *nur* bei den Gleichheitszeichen umgebrochen werden kann, dann muß man die Formel als

```
$f(x,y) = {x^2-y^2} = {(x+y)(x-y)}$
```

schreiben. Man kann in Formeln einen ähnlichen Befehl wie den '`\-`' Befehl benutzen, er lautet '`\*`', allerdings schreibt TEX im Trennungsfall keinen Bindestrich, sondern ein Multiplikationssymbol ($\times$). Ebenso kann man, außerhalb einer Gruppierung, auch den Befehl '`\allowbreak`' verwenden, um TEX einen Hinweis zu geben, wo am besten umgebrochen wird.

Die Strafen für Zeilenumbrüche in Formeln sind

Befehl	*Strafe*	*Bedeutung*
`\relpenalty`	500	Umbruch nach einem Rel-Atom
`\binoppenalty`	700	Umbruch nach einem Bin-Atom

Sie können auch jederzeit in einer Formel die Strafe mit dem Befehl '`\penalty<Nummer>`' selber angeben. Auch der Befehl '`\nobreak`' ist erlaubt.

7.4.7 Geschweifte Klammern

Geschweifte Klammern werden in der Mathematik z.B. benötigt, um Mengen darzustellen. Bei einfachen Mengendarstellungen ist das auch problemlos möglich, denken sie nur daran, daß sie die geschweiften Klammern mit den Befehlen '`\{`' und '`\}`' schreiben. Somit ergibt '`$\{a,b,c\}$`' das gewünschte Resultat $\{a,b,c\}$. Etwas anders sieht es aus, wenn die Mengendarstellungen komplizierter werden. Um die korrekte Abstände in der Angabe

$$\{\,x\mid x>5\,\}$$

zu erreichen, müssen sie

```
$$\{\,x\mid x>5\,\}$$
```

eingeben. Noch etwas schwieriger wird die Sache, wenn die Klammern größer werden. Die Ausgabe

$$\bigl\{\,\bigl(x,f(x)\bigr)\bigm|x\in D\,\bigr\}$$

wurde mit der Anweisung

```
$$\bigl\{\,\bigl(x,f(x)\bigr)\bigm|x\in D\,\bigr\}$$
```

erreicht.

Ein anderes Beispiel, in dem geschweifte Klammern benötigt werden ist die Auswahl aus verschiedenen Alternativen. Mit

```
$$|x|=\cases{x,&wenn$x\ge0$;\cr -x,&anderenfalls.\cr}$$
```

erreichen sie das Ergebnis

$$|x|=\begin{cases}x, & \text{wenn}x\ge 0;\\ -x, & \text{anderenfalls.}\end{cases}$$

Sie sehen hier erstmals die Verwendung des '`&`'-Zeichens.[11] Mit dem Zeichen werden zwei Spalten hinter der öffnenden geschweiften Klammer, die von dem Befehl automatisch eingesetzt wird, getrennt. Alles *vor* diesem Zeichen wird automatisch in '`$`' gesetzt, steht also im mathematischen Modus. Alles hinter diesem Zeichen wird als normaler Text behandelt. Die Zeilen, die schließlich die Auswahl erst ausmachen, werden durch den Befehl '`\cr`' getrennt. Es sind übrigens beliebig viele Zeilen möglich.

Schließlich gibt es noch geschweifte Klammern über, bzw. unter Formeln. Diese werden mit den Befehlen '`\overbrace`' und '`\underbrace`' erreicht. Da diese Befehle wie große Symbole behandelt werden, ist es möglich noch *Grenzen* anzugeben.

```
$$\overbrace{x+\cdots+x}^{k\rm\;mal}$$
$$\underbrace{x+y+z}_{>\,0}$$
```

ergeben

$$\overbrace{x+\cdots+x}^{k\rm\;mal}$$

$$\underbrace{x+y+z}_{>\,0}$$

[11] Dieses Zeichen wird in Tabellen verwendet. Da es sich hier um etwas ähnliches handelt, ist die Verwendung dieses Zeichens hier notwendig. Auch die andern, in einer Tabelle erlaubten Befehle sind hier erlaubt.

7.4.8 Matrizen

Eine weitere Anwendung, die einer Tabelle sehr ähnlich ist, sind die Matrizen. Da auch sie öfter vorkommen, bietet TEX einige Befehle zur Konstruktion dieser Matrizen. Die Matrix

$$A=\left(\begin{matrix}x-\lambda&1&0\\ 0&x-\lambda&1\\ 0&0&x-\lambda\end{matrix}\right)$$

wurde mit

```
$$A=\left(\matrix{x-\lambda&1&0\cr
                  0&x-\lambda&1\cr
                  0&0&x-\lambda\cr}\right)$$
```

gesetzt. Es werden wieder die '`&`' Zeichen benötigt, um die Spalten, und die '`\cr`' Befehle benötigt, um die Zeilen zu trennen. TEX setzt die Matrize nicht automatisch in Klammern, da aber Matrizen meist in runde Klammern gesetzt werden, gibt es den Befehl '`\pmatrix`', der dann die Klammern automatisch setzt.

Die einzelnen Einträge der Matrix werden normalerweise zentriert, und durch ein quad getrennt, man kann die Ausrichtung aber durch '`\hfill`' Befehle ändern. Die Einträge werden einzeln gesetzt, man sollte also nicht versuchen zwei Einträge in eine Gruppenklammer zu setzen.

Vielfach werden Matrizen nicht ganz ausgeschrieben. Neben den Punktbefehlen, die weiter oben beschrieben wurden, gibt es noch die Befehle '`\vdots`' für vertikale und '`\ddots`' für diagonale Punkte.

$$A=\begin{pmatrix}a_{11}&a_{12}&\ldots&a_{1n}\\ a_{21}&a_{22}&\ldots&a_{2n}\\ \vdots&\vdots&\ddots&\vdots\\ a_{m1}&a_{m2}&\ldots&a_{mn}\end{pmatrix}$$

erreicht man durch

```
$$A=\pmatrix{a_{11}&a_{12}&\ldots&a_{1n}\cr
             a_{21}&a_{22}&\ldots&a_{2n}\cr
             \vdots&\vdots&\ddots&\vdots\cr
             a_{m1}&a_{m2}&\ldots&a_{mn}\cr}$$
```

Neben den normalen Matrizen kennt TEX noch eine Sonderform, bei der Zeilen und Spalten der Matrix beschriftet werden. Der entsprechende Befehl heißt '`\bordermatrix`' und sieht in der Anwendung folgendermaßen aus.

$$M=\begin{array}{c c}
 & \begin{array}{ccc} C & I & C' \end{array} \\
\begin{array}{c} C \\ I \\ C' \end{array} & \begin{pmatrix}1&0&0\\ b&1-b&0\\ 0&a&1-a\end{pmatrix}
\end{array}$$

wird realisiert durch

```
$$M=\bordermatrix{&C&I&C’\cr C&1&0&0\cr I&b&1-b&0\cr C’&0&a&1-a\cr}$$
```

Jeweils die erste Zeile und die erste Spalte der Matrix werden dann nicht geklammert. Beachten sie, daß der erste Spalteneintrag der ersten Zeile leer ist, was durchaus erlaubt ist, und das die Klammern wieder automatisch gesetzt wurden.

Wenn sie in einer Textformel Matrizen schreiben wollen, dann sollten sie sie anders konstruieren, da die Matrixbefehle zu große Matrizen erzeugen. ${1\,0\choose0\,1}$ erzeugt man z.B. mit

```
$1\,0\choose0\,1$|
```

und eine Matrix wie $\bigl({a\atop l}{b\atop m}{c\atop n}\bigr)$ erzeugt man durch die Befehle

```
$\bigl({a\atop l}{b\atop m}{c\atop n}\bigr)$
```

7.4.9 Vertikaler Leerraum

Wir haben schon einige Befehle kennengelernt, die es erlauben die Höhe einer Formel so zu verändern, daß sie anders als normal gesetzt wird. Dazu gehören die Befehle '`\strut`' und '`\mathstrut`'. Es gibt aber noch andere Möglichkeiten. Als erstes den Befehl

```
\phantom{<Teilformel>}
```

TeX belegt bei Benutzung dieses Befehls zwar den Platz, den die Teilformel einnehmen würde, gibt die Formel selber aber nicht aus. Die Angabe von '`\phantom{0}2`' würde als Ergebnis nur die Zwei liefern, aber den Platz von der Null und der Zwei beanspruchen. Wenn sie also z.B. ein Symbol in der Größe eines Summenzeichens selber später per Hand einfügen wollen, dann schreiben sie

```
\mathop{\phantom\sum}
```

Der '`\mathop`' Befehl sorgt dafür, das der Platz wie für ein großes Symbol gesetzt wird.

Weiterhin gibt es die beiden Subbefehle '`\hphantom`' und '`\vphantom`'. Bei der ersten Variante wird die Breite der nachfolgenden Teilformel freigehalten, Höhe und Tiefe sind aber Null, bei der zweiten Alternative, wird die Höhe und die Tiefe der Teilformel freigehalten, die Breite aber ist Null. Der '`\mathstrut`' Befehl wird einfach mit

```
\vphantom(
```

erzeugt.

Plain TeX kennt noch den Befehl '`\smash{<Teilformel>}`', der zwar die Teilformel ganz normal ausgibt, aber die Höhe und die Tiefe der Ausgabe auf Null setzt. Somit ist es möglich jeder Ausgabe jede beliebige Höhe und Tiefe zu geben. Mit

```
\mathop{\smash\limsup\vphantom\liminf}
```

wird das große Symbol '$\limsup$' ausgegeben, allerdings mit den Höhenabmessungen von '$\liminf$', also mit der Tiefe Null.

Mit den Boxbefehlen '`\raise`' und '`\lower`' können Teilformeln auch noch höher oder tiefergestellt werden. Mit der Eingabe

```
$2^{\raise1pt\hbox{$\scriptstyle n$}}$
```

erreicht man, daß der Exponent um ein pt höher gesetzt wird, als normal, also $2^{\raise1pt\hbox{$\scriptstyle n$}}$ statt 2^n. Der Befehl '`\scriptstyle`' ist nötig, da normlerweise in einer hbox die Schriftart gewählt wird, die die mathematische Umgebung umgibt.

7.4.10 Spezielle Befehle für besondere Aufgaben

`\nonscript`

Folgt dem Befehl '`\nonscript`' unmittelbar etwas Leim, oder ein Kern, dann wird dieser nicht entsprechend dem Script- oder Scriptscriptstil ausgegeben, sondern in der für den Text- oder Displaystil vorgegebenen Größe. Der Befehl '`\nonscript\;`' erzeugt also genau den Zwischenraum, der in obiger Tabelle mit (3) angegeben ist.

`\every...`

Mit den beiden Befehlen '`\everymath`' und '`\everydisplay`' kann jeder mathematishcen Umgebung, Textformel, bzw. abgesetzte Formel eine Befehlsgruppe übergeben werden. Die Befehle entsprechen somit dem Befehl '`\everypar`' für Absätze.

7.5 Abgesetzte Formeln

7.5.1 Einzeilige Formeln

Wie abgesetzt Formeln aufgerufen werden, dürfte bis hier schon klar sein, auch, wie man normalen Text in eine derartige Formel schreibt. Wir haben auch schon gesehen, wie man mehr als eine Formel in eine derartige Zeile schreibt. Hier jetzt noch einmal ein zusammenfassendes Beispiel. Die Eingabe von

```
$$X_n=X_k\qquad\hbox{wenn, und nur wenn}\qquad
  Y_n=Y_k\quad\hbox{und}\quad Z_n=Z_k$$
```

erzeugt

$$X_n=X_k\qquad\hbox{wenn, und nur wenn}\qquad Y_n=Y_k\quad\hbox{und}\quad Z_n=Z_k$$

Achten sie auf die unterschiedlichen Zwischenräume, bei dem *wenn, und nur wenn* und dem *und*. Wenn sie hboxen in mathematischen Formeln verwenden, können sie sich auch noch für eine andere Art des Leerraums entscheiden. Beenden sie ihren Text in der hbox einfach mit einem Leerzeichen, das erscheint dann in der Formel. Mit der Eingabe von

```
$$... \hbox{f\"ur alle }n\ge0$$
```

bekommen sie

$$\ldots\hbox{für alle }n\ge0$$

Sie haben sich sicher schon gefragt, wie man in TeX denn derartige abgesetzte Formeln nummerieren kann. Die Lösung des Problems liegt in dem Befehl '`\eqno`'. Er wird

```
$$<Formel 1>\eqno<Formel 2>$$
```

angewendet, und alles hinter dem Befehl wird als Formelnummer verwendet. Mit der Eingabe von

```
$$x^2-y^2=(x+y)(x-y)\eqno(15)$$
```

erhalten sie

$$x^2-y^2=(x+y)(x-y) \tag{15}$$

Ähnlich arbeitet der Befehl '`\leqno`', nur daß dann die Formelnummer links von der Formel erscheint, sie muß trotzdem als *zweites* Argument angegeben werden.

```
$$x^2-y^2=(x+y)(x-y)\leqno(16)$$
```

erscheint also als

$$(16)\qquad x^2-y^2=(x+y)(x-y)$$

An den Beispielen können sie auch sehen, daß die Formel unabhängig von den Formelnummern zentriert wird. Nur wenn die Formel so lang wäre, daß sie bei der Zentrierung mit der Nummer in Berührung käme, würde sie etwas zur Seite gesetzt. Wenn sie dann immer noch nicht paßt, dann können sie noch einen der folgenden Tricks verwenden. Setzen sie die Formelnummer in einen '`\rlap`' oder einen '`\llap`' Befehl, sie erscheint dann in einer eigenen Zeile.[12]

```
$$x^2-y^2=(x+y)(x-y)\eqno\llap{(17)}$$
```

Erzeugt

$$x^2-y^2=(x+y)(x-y)$$

(17)

[12] Welchen der beiden Befehle sie verwenden müssen ergibt sich daraus, nach welcher Seite die Formelnummer überlappen muß.

7.5.2 Mehrzeilige Formeln

Für mehrzeilige Formeln stellt TEX einige nützliche Befehle bereit. Der erste Befehl dieser Art ist der Befehl '`\eqalign`', der es mit einer ähnlichen Konstruktion wie schon bei '`\matrix`' und '`\cases`' erlaubt, mehrere Formeln zu setzen. Der grundsätzliche Aufbau ist

```
\eqalign{<linke Seite 1> & <rechte Seite 1> \cr
         <linke Seite 2> & <rechte Seite 2> \cr
            .
            .
         <linke Seite n> & <rechte Seite n>\cr}
```

Dabei wird die linke Seite immer rechtsbündig, die rechte immer linksbündig gesetzt, so daß beide Teile eine gemeinsame Formel ergeben. Da häufig Gleichungen gesetzt werden müssen, bietet sich dieses Befehlsformat an. Man kann z.B. das Gleichheitszeichen immer als erstes Zeichen der rechten Seite wählen, und erreicht damit, daß die Gleichheitszeichen genau übereinander stehen.

```
$$\eqalign{X_1+\cdots+X_n&=m,\cr
           Y_1+\cdots+Y_m&=n.\cr}$$
```

Ergibt somit

$$\begin{aligned} X_1+\cdots+X_n &= m,\\ Y_1+\cdots+Y_m &= n. \end{aligned}$$

Der Befehl erzeugt eine Box, die vertikal zentriert wird, es können also auch mehrere Gleichungsketten nebeneinander dargestellt werden.

$$\left\{\begin{aligned}\alpha&=f(z)\\ \beta&=f(z^2)\\ \gamma&=f(z^3)\end{aligned}\right\}\qquad\left\{\begin{aligned}x&=\alpha^2-\beta\\ y&=2\gamma\end{aligned}\right\}.$$

wurde mit

```
$$\left\{\eqalign{
\alpha&=f(z)\cr \beta&=f(z^2)\cr \gamma&=f(z^3)\cr}
\right\}\qquad\left\{
\eqalign{x&=\alpha^2-\beta\cr y&=2\gamma\cr}\right\}.$$
```

erzeugt.

Bei derartigen Folgen von Gleichungen, aber auch bei anderen Konstruktionen, bei denen mehrere Formeln untereinander stehen, ist es mitunter wünschenswert die Zeilen zu gruppieren. Um etwas zusätzlichen Platz zwischen zwei Formeln zu erhalten, kann hinter einem '`\cr`' der Befehl

```
\noalign{\vskip3pt}
```

geschrieben werden, um 3pt mehr Platz zu erhalten. Statt 3pt kann natürlich jede Leimangabe stehen, die gerade gebraucht wird.[13]

Um Zeilen in einer mehrzeiligen Umgebung nummerieren zu können gibt es die zwei verwandten Befehle '`\eqalignno`' und '`\leqalignno`', wobei der zweite wieder nur die Formelnummer auf der linken Seite ausgibt. Wenn einzelne Zeilen nicht nummeriert werden sollen, dann braucht auch das '`&`' nicht geschrieben zu werden. Die Eingabe von

[13] Es kann mit diesem Befehl alles mögliche zwischen zwei Zeilen geschrieben werden, wie gleich in einem weiteren Beispiel zu sehen ist.

```
$$\eqalignno{(x+y)(x-y)&=x^2-xy+yx-y^2\cr
    &=x^2-y^2;&(7.1)\cr
  (x+y)^2&=x^2+2xy+y^2.&(7.2)\cr}$$
```

ergibt

$$\begin{aligned}(x+y)(x-y) &= x^2-xy+yx-y^2 \\ &= x^2-y^2; && (7.1) \\ (x+y)^2 &= x^2+2xy+y^2. && (7.2)\end{aligned}$$

was auch zeigt, daß durchaus auch einzelne Teile der Formel ganz leer bleiben können.

Es gibt noch einen wichtigen Unterschied zwischen den Befehlen, die eine Formelzeile ausgeben, und denen, die das nicht tun. Mit dem '`\eqalignno`' Befehl werden Zeilen ausgegeben, die über eine ganze Zeile gehen, mit dem '`\eqalign`' Befehl werden Boxen erzeugt, die nur ihre natürliche Breite haben, und die vertikal zentriert werden. Daraus ergeben sich einige Konsequenzen:

1. Der `\eqalignno` Befehl kann nur *einmal* in einer Formel verwendet werden, der `\eqalign` Befehl, wie oben gezeigt, mehrfach.

2. Die Zeilen, die aus einem `\eqalign` Befehl erzeugt wurden können nicht zwischen zwei Seiten umgebrochen werden, da dieser Befehl ja eine Box erzeugt. Der `\eqalignno` Befehl erzeugt dagegen einzelne Zeilen, die sehr wohl auf zwei Seiten aufgeteilt werden können.

3. Weiterhin können die Zeilen, die mit dem '`\eqalignno`' Befehl angeordnet werden, durch Text unterbrochen werden, *ohne* daß die Ausrichtung dabei verloren geht. Eine Anordnung, wie

$$x = y+z \qquad (7.3)$$

und

$$y^2+z^2 = x^2 \qquad (7.4)$$

wird mit einem '`\noalign{\hbox{und}}`' hinter dem ersten '`\cr`' erzeugt.

Eine weitere Möglichkeit mehrere Zeilen untereinander zu schreiben, besteht in dem Befehl '`\displaylines`'. Das generelle Format dieses Befehls lautet

```
$$\displaylines{<erste Zeile>\cr
               <zweite Zeile>\cr
                      .
                      .
               <letzte Zeile>\cr}$$
```

Die einzelnen Zeilen werden durch ein '`\hfil`' zentriert, was also durch ein '`\hfill`' überschrieben werden kann. Mit

```
$$\displaylines{\hfill x\equiv x;\hfill\llap{(1)}\cr
   \hfill\hbox{wenn}\quad x\equiv y\quad\hbox{dann}\quad
      y\equiv x;\hfill\llap{(2)}\cr
   \hfill\hbox{wenn}\quad x\equiv y\quad\hbox{und}\quad
      y\equiv z\quad\hbox{dann}\quad
      x\equiv z.\hfill\llap{(3)}\cr}$$
```

erhalten sie

$$x \equiv x; \qquad (1)$$

$$\text{wenn}\quad x\equiv y\quad\text{dann}\quad y\equiv x; \qquad (2)$$

$$\text{wenn}\quad x\equiv y\quad\text{und}\quad y\equiv z\quad\text{dann}\quad x\equiv z. \qquad (3)$$

Sie haben sicher schon bemerkt, daß der Abstand zwischen zwei und mehr Zeilen größer ist, als der Abstand der Zeilen im Text. Dies geschieht zum Zwecke der besseren Lesbarkeit. Der Befehl, der diese 'Öffnung' der Zeilen bewirkt, wird automatisch nach jedem der Mehrzeilenbefehle eingefügt, er lautet '`\openup1\jot`'. Wenn sie nicht mit TEX's Befehlen arbeiten wollen, sondern die Anordnung mit dem '`\halign`' Befehl selber bewerkstelligen wollen, dann sollten sie den Öffnungsbefehl selber einfügen. Aber beachten sie dabei folgendes. Die 'Öffnung' der Zeilen wirkt nur innerhalb der Gruppe, die durch die '`$$`' erzeugt wird, dadurch gibt es einen kleinen Nebeneffekt. Die letzte Formelzeile steht ja wieder vor einer Textzeile, und da ist der Abstand wieder der normale, sie müssen also am Ende der Formelzeilen *per Hand* noch etwas zusätzlichen Abstand einfügen.

7.5.3 Lange Formeln

Alle Regeln, die sie bisher über Formelsatz gelernt haben, nützen ihnen nichts, wenn sie eine Formel haben, die einfach länger ist, als eine Zeile. D. Knuth empfiehlt in solch einem Falle die Verwendung des '`eqalign`' Befehls. Man kann dabei eine lange Formel z.B. so setzen

$$\begin{aligned}\sigma(2^{34}-1,2^{35},1) &= -3+(2^{34}-1)/2^{35}+2^{35}\!/(2^{34}-1)\\ &\qquad+7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\end{aligned}$$

Diese Formel wurde mit

```
$$\eqalign{\sigma(2^{34}-1,2^{35},1)
  &=-3+(2^{34}-1)/2^{35}+2^{35}\!/(2^{34}-1)\cr
  &\qquad+7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\cr}$$
```

erzeugt. TEX kennt selber *keine* Regeln, für den Umbruch einer derartigen Formel, dazu muß man den Inhalt der Formel nicht nur kennen, sondern sie auch verstehen, um den besten Platz für den Umbruch zu erkennen. Statt dessen stehen hier einige kleine Regeln, die ihnen die Entscheidung etwas erleichtern sollen.

1. Brechen sie eine lange Formel *vor* einem binären Symbol, oder einem Relationssymbol um, also anders, als in Textformeln.

2. Rücken sie den zweiten Zeil der Formel etwas ein. Sie können auch den ersten Teil der Formel mit einem `\hfill` ganz nach links, den zweiten mit einem weiteren `\hfill` ganz nach rechts schieben.

Achten sie zum Abschluß darauf, daß sie keine '`\left`', '`\right`' Konstruktionen über mehrere Zeilen verwenden. Bei einer langen Formel müssen sie die Größenwahl der Klammer schon selber vornehmen.

Kapitel 8

Definitionen oder Makros

8.1 Generelles

Wenn sie einen Text schreiben, der oft dieselbe Folge von Zeichen enthält, dann können sie sich mit TEX die Arbeit erleichtern. Angenommen, sie schreiben einen mathematischen Text, der oft den Vektor '$(x_1, \ldots, x_n)$' enthält. Dann können sie mit der Anweisung

```
\def\xvek{(x_1,\ldots,x_n)}
```

einen neuen Befehl '`\xvek`' bereitstellen, der ab sofort als Abkürzung für '`(x_1,\ldots,x_n)`' steht. Eine komplizierte Formel, wie

$$\sum_{(x_1,\ldots,x_n)\ne(0,\ldots,0)} \bigl(f(x_1,\ldots,x_n)+g(x_1,\ldots,x_n)\bigr)$$

kann dann mit der einfachen Anweisung

```
$$\sum_{\xvek\ne(0,\ldots,0)} \bigl(f\xvek+g\xvek\bigr)$$
```

geschrieben werden. Derartige Abkürzungen bieten sich natürlich nur an, wenn eine Zeichenfolge wirklich öfters vorkommt. Sie bietet dann aber auch eine gewisse Gewähr gegen Tippfehler, die auch nicht zu vernachläßigen ist.

TEX expandiert ihre neu geschaffene Kontrollsequenz, bis nichts mehr zu expandieren ist, sprich nur noch Primitive übrigbleiben. Der Befehl

```
\xvek
```

wird zunächst zu

```
(x_1,\ldots,x_n)
```

expandiert, und dieser Ausdruck dann zu

```
(x_1,\mathinner{ldotp\ldotp\ldotp},x_n)
```

da auch '`\ldots`' ein Makro[1] ist. Die jetzt noch verbliebenen Kontrollsequenzen sind keine Makros mehr, werden also auch nicht weiter expandiert.

Es empfiehlt sich eine eigene Bibliothek solcher Makrodefinitionen anzulegen, in denen alle Definitionen gespeichert werden, die sie öfters brauchen. Achten sie aber darauf, daß die Bibliothek nicht *zu* groß wird, da sie für jeden Text mit

```
\input makro
```

[1]Statt des Begriffs 'Definition' hat sich auch der Begriff 'Makro' durchgesetzt, da quasi eine mikroskopisch kleine Anweisung einen makroskopischen Effekt haben kann.

eingelesen werden muß,[2] was jedesmal etwas Zeit dauert.

Definitionen dieser Art sind natürlich auch außerhalb des mathematischen Modus erlaubt, und sie haben auch dort dieselbe Syntax. Eine Definition hat *immer* den folgenden Aufbau

```
\def\<name>{<Ersatztext>}
```

sie wird also immer mit der Kontrollsequenz '`\def`' eingeleitet, der dann der Name der neu zu definierenden Kontrollsequenz, inklusive Gegenschrägstrich, folgt. Zum Schluß dann in geschweiften Klammern der Text, der aus der Kontrollsequenz expandiert werden soll. *Achtung*, die geschweiften Klammern haben hier ausnahmsweise einmal *nicht* die Wirkung einer Gruppierungsklammer. Wenn sie erreichen wollen, daß eine Kontrollsequenz zu '`{\bf x}`' expandiert werden soll, dann muß die Definition

```
\def\bfx{{\bf x}}
```

lauten. Anderenfalls würde der gesamte nachfolgende Text fett gesetzt.

8.2 Parameterisierte Definitionen

Angenommen, sie wollten außer '$(x_1, \ldots, x_n)$' auch öfters ähnliche Konstruktionen, nur mit anderen Buchstaben, also z.B. '$(y_1, \ldots, y_n)$', schreiben. Sie müssen dafür nicht jeweils eine Extradefinition schreiben, TEX bietet die Möglichkeit Makros auch mit Parametern zu definieren und aufzurufen. Die Definition

```
\def\vektor#1{(#1_1,\ldots,#1_n)}
```

erwartet bei Aufruf ein Argument, daß dan an die entsprechende Stelle gesetzt wird. Der Aufruf '`\vektor x`' erzeugt '$(x_1, \ldots, x_n)$', der Aufruf '`\vektor y`' erzeugt '$(y_1, \ldots, y_n)$'. Auch mehrere Parameter sind möglich. Wenn z.B. der höchste Wert der Vektoren nicht immer n sein soll, dann empfiehlt sich die Definition

```
\def\vektorn#1{(#1_1,\ldots,#1_#2))}
```

Der Aufruf '`\vektorn am`' erzeugt dann '$(a_1, \ldots, a_m)$'. Sie können bis zu neun Argumente an ein Makro übergeben (`#1` bis `#9`). Die Reihenfolge des Auftretens im Ersatztext ist dabei beliebig, nicht aber hinter der Angabe des Namens. Sie können z.B. kein '`#5`' benutzen, wenn sie nicht vorher alle Parameterzeichen bis '`#4`' verwendet haben.[3]

Wenn sie eine Kontrollsequenz mehrfach definieren, dann ist immer nur die letzte Definition gültig. Die Anweisungen

```
\def\a{Hallo} \def\a{wie geht's}
```

bewirkt, daß bei Aufruf von '`\a`' 'wie geht's' ausgegeben wird. Allerdings lassen sich Definitionen lokal, in einer Gruppe, ändern, ohne, daß die ursprüngliche Bedeutung verloren geht.

Sie können sich entscheiden, ob sie die Definition mit Leerzeichen übersichtlicher machen wollen, oder nicht. Die Definitionen

```
\def\a #1 #2{#1 oder #2}
\def\a#1#2{#1 oder #2}
```

[2] Ich gehe hier davon aus, daß sie alle Makros in einer Datei `makro.tex` gespeichert haben, wählen sie einen anderen Dateinamen, dann lautet auch der Name hinter dem `\input` Befehl anders.

[3] Ab hier dürfte auch klar sein, wieso das Zeichen `#` normalerweise verboten ist.

sind gleichwertig. Auch im Aufruf ist es gleichgültig, ob sie '`\a A B`' oder '`\aAB`' verwenden, Das Ergebnis ist immer 'A oder B'.

Normalerweise folgt einem Parameterzeichen nur eine Ziffer zwischen 1 und 9, man kann aber durch weitere Zeichen erreichen, daß auch größere Argumente leicht übergeben werden können.[4] Die Definition

```
\def\cs #1. #2\par{...}
```

definiert einen neuen Befehl mit zwei Argumenten, wobei das erste Argument durch das Vorkommen eines Punktes, gefolgt von einem Leerzeichen, hier ist das Leerzeichen wichtig, das zweite Argument von dem Befehl '`\par`' beendet wird. Diese Beender, gelten auch beim Aufruf. Wenn man diesen Befehl mit

```
\cs Sie schulden \$5.00 DM. Zahlen Sie sie.\par
```

aufruft, dann wird '*Sie schulden $5.00 DM*' zum ersten Argument; der erste Punkt wird nicht als Begrenzer erkannt, da ihm keine Leerstelle folgt, das zweite Argument ist dann '*Zahlen Sie sie.*' Weiterhin wird auch dann ein Argument weiter angenommen, wenn bei Auftreten eines Argumentbegrenzers noch geschweifte Klammern geöffnet sind. Hätte man obigen Befehl mit

```
\def\cs #1.#2\par{...}
```

definiert, dann müßte der Aufruf

```
\cs Sie schulden {\$5.00} DM. Zahlen Sie sie.\par
```

lauten, um das gewünschte Ergebnis zu erhalten. Man kann sich diese Form der Parameterbegrenzung zunutze machen, um z.B. mathematische Theoreme *schön* zu setzen.

Theorem 1. *TEX hat große Makrofähigkeiten.*

Tatsächlich kennt *Plain* TEX einen Befehl '`\proclaim`', mit dem die Beispielzeile gesetzt wurde. Er ist folgendermaßen definiert

```
\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2}\par\medbreak}
```

Wenn sie diese Definition ändern, was sie ohne weiteres durch eine Neudefinition zu Beginn ihres Textes tun können, ändern sie damit das Aussehen *aller* Theoreme in ihrem gesamten Text.

8.2.1 Die Regeln der Parameterisierung

Jede Makrodefinition hat die folgende Form

```
\def<Kontrollsequenz><Parameterliste>{<Ersatztext>}
```

Wobei die Parameterliste keine geschweiften Klammern enthalten darf, außerdem das Spezialzeichen '`#`' von Ziffern 1 bis (maximal) 9 gefolgt wird, die der Reihe nach auftreten müssen; der Ersatztext enthält nur paarweise geschweifte Klammern und die entsprechenden Parameterzeichen '`#`' haben nur Ziffern als nächstes Zeichen, es sei denn, ihnen folgt ein weiteres '`#`', was das einfache Zeichen '`#`' darstellt. Zur weiteren Betrachtung dient ein Beispiel einer Definition, die zu nichts anderem gut ist, als uns die Regeln, nach denen TEX vorgeht zu verdeutlichen.

```
\def\cs AB#1#2C$#3\$ {#3{ab#1}#1 c##\x #2}
```

Diese Definition hat eine Parameterliste, bestehend aus neun Token

[4] Von dieser Möglichkeit sollte man nur sehr sparsam Gebrauch machen, und nur, wenn einem die Definition eines Makros *immer* klar ist.

```
A, B, #1, #2, C, $, #3, \$, Leerzeichen
```

und einen Ersatztext, bestehend aus zwölf Token

```
#3, {, a, b, #1, }, #1, Leerzeichen, c, # \x, #2
```

Beim Aufruf erwartet TEX zunächst die beiden Zeichen '`AB`'. Falls diese beim Aufruf nicht mit angegeben werden, erhalten sie eine Fehlermeldung, die besagt, daß der Aufruf des Makros nicht mit seiner Definition übereinstimmt. Dann folgen die Argumente 1 und 2, ...

Woran merkt TEX nun, daß ein Parameter beendet ist? Es gibt zwei Fälle, einmal kann ein Parameter noch von einem Parameterbegrenzer gefolgt werden, oder es folgt ein nächster Parameter. Schließlich wird das Ende der Parameterliste erreicht, wenn die geöffnete, geschweifte Klammer den Beginn des Ersatztextes anzeigt. Der Aufruf

```
\cs AB {\Look}C${And\$ }{look}\$ 5.
```

bewirkt nun folgendes. Das erste Argument ist '`\Look`', von dem zuvor die Klammern entfernt wurden. Das zweite Argument bleibt leer, da das '`$C`' sofort folgt. Das dritte Argument bekommt den *Inhalt* '`{And\$ }{look}`', da das erste Auftreten von '`\$`' in einer geschweiften Klammer steht. Auch die Klammern um das gesamte Argument werden *nicht* entfernt, da dann die beiden inneren Klammern alleine übrigblieben, was auch wieder zu einer nicht richtigen Verteilung der geschweiften Klammern führen würde. Die Ausgabe dieses Aufrufs wäre

```
{And\$␣}{look}{ab\Look}\Look␣c#\x5.
```

Das Leerzeichen wird nicht entfernt, auch wenn es hinter dem Kontrollwort '`\Look`' steht, da TEX Leerzeichen hinter Kontrollworten sozusagen nur auf den ersten Rutsch entfernt, aber nicht bei der Expansion von Makros.

Noch einen Spezialfall. Wenn sie als letztes Zeichen vor der öffnenden Klammer, die den Ersatztext einleitet ein '`#`' schreiben, dann verhält sich TEX so, als stünde die öffnende Klammer beim Aufruf vor dem Argument. Wenn sie also

```
\def\a#1#{\hbox to #1}
```

definieren, dann ergibt der Aufruf '`\a3pt{x}`' als Ergebnis

```
\hbox to 3pt{x}
```

und nicht nur

```
\hbox to 3
```

wie man zunächst erwarten würde. Was TEX genau macht, wenn es die Parameter einer Makrodefinition auswertet kann man sich natürlich auch anzeigen lassen, es geht, wie nicht anders zu erwarten mit dem Befehl '`\tracingmacros=1`'.

Jeder Mensch macht Fehler. Der häufigste, in Zusammenhang mit Makrodefinitionen und Makroaufrufen dürfte eine fehlende schließende Klammer bei der Argumentübergabe an eine Kontrollsequenz sein. TEX tritt dem entgegen, indem es bei dem ersten Auftreten des Befehls '`\par`' das weitere Einlesen eines Arguments beendet, es sei denn, sie haben TEX ausdrücklich angewiesen '`\par`' zu akzeptieren. Dies geschieht, indem sie den Befehl '`\long`' vor das '`\def`' schreiben. Mit der Anweisung

```
\long\def\bftext{{\bf #1}}
```

können sie auch mehrere Paragraphen fett ausgeben lassen.[5] Etwas schwieriger sieht die Sache aus, wenn sie die schließende Klammer des Ersatztextes vergessen haben. Hier wäre eine Begrenzung durch ‘`\par`’ nicht wünschenswert, da dieser Befehl in Ersatztexten oft gebraucht wird. Es gibt aber dennoch eine Möglichkeit sich wenigstens etwas abzusichern. Wenn sie der ‘`\def`’ Anweisung ein ‘`\outer`’ voranstellen, dann kann die neudefinierte Kontrollsequenz nur in der äussersten vertikalen Liste verwendet werden. Sie kann also nicht in einer Tabelle stehen, noch kann sie in einem Ersatztext auftauchen o.ä. TeX hat es in diesem Fall leichter einen Fehler zu finden.

Schließlich kann einer ‘`\def`’ Anweisung noch ein ‘`\global`’ vorangestellt werden. Der Effekt ist wie man ihn erwarten würde. Da diese Situation oft auftritt, gibt es aber auch die Abkürzung ‘`\gdef`’, die das gleiche bewirkt.

8.3 Ein weiterer Zuweisungsbefehl

Bisher haben wir schon einige Befehle kennengelernt, die einer Kontrollsequenz etwas zuweisen. Z.B. ‘`\font`’, ‘`\chardef`’, ‘`\countdef`’ oder ‘`\def`’. Darüber hinaus gibt es noch den Befehl ‘`\let`’, der einer Kontrollsequenz eine Reihe von Token zuweist. Mit der Anweisung

```
\let\a=\def
```

könnte man erreichen, daß Neudefinitionen nicht mehr mit ‘`\def`’ eingeleitet werden brauchen, sondern statt dessen mit dem Befehl ‘`\a`’. Ist das Argument der ‘`\let`’ Anweisung ein einzelnes Zeichen, dann verhält sich der neu geschaffene Befehl wie dieses Zeichen, allerdings mit ein paar Ausnahmen. Wenn man z.B. mit ‘`\let\zero=0`’ den Befehl ‘`\zero`’ schafft, dann kann dieser dennoch nicht in numerischen Konstanten verwendet werden, da dort nur Ziffern erlaubt sind, und ‘`\zero`’ *kein* Makro ist, das expandiert wird. Letztlich gibt es auch noch den Befehl

```
\futurelet\cs=<Token1><Token2>
```

was sich genauso verhält, wie die Anweisung

```
\let\cs=<Token2><Token1><Token2>
```

8.4 Die Entscheidungsbefehle

TeX bietet eine Reihe von Befehlen, mit denen das Verhalten von TeX gesteuert werden kann, d.h. daß je nach Ausgangssituation unterschiedliche Ergebnisse herauskommen. Diese Möglichkeit findet natürlich auch besonders in Makros eine Anwendung. Der generelle Aufbau dieser Bedingungsprüfungen ist

```
\if<Bedingung><Wahrtext>\else<Falschtext>\fi
```

Wenn bei Erreichen dieser Anweisung die Bedingung *wahr* ist, dann wird der `Wahrtext` ausgeführt, andernfalls der `Falschtext`. Der Befehl

```
\ifodd\count0 \rechteseite \else \linkeseite \fi
```

sorgt dafür, daß je nach Seitenzahl, die bei TeX üblicherweise im Zählregister 0 steht, entweder die Kontrollsequenz für rechte, oder linke Seiten ausgeführt wird. Achten sie bei numerischen Vergleichen in den Abfragen darauf, daß die numerischen Konstanten immer richtig abgeschlossen werden. Bei der Anweisung

```
\ifnum\zaehler=0...
```

[5] Allerdings verbrauchen sie mit dieser Methode *sehr* viel Speicher — es gibt bessere Lösungen.

sollten sie hinter der 0 ein Leerzeichen schreiben, um sicherzustellen, daß die Konstante richtig erkannt wird, auch wenn der nachfolgende Text auch wieder mit Ziffern anfängt.

Hier nun eine vollständige Liste der Vergleichsbefehle.

1. '\ifnum<Nummer1><Relation><Nummer2>'
   Vergleicht zwei Zahlen miteinander. Als Relation sind dabei nur die Vergleiche '<', '=' und '>' zugelassen.

2. '\ifdim<Dimension1><Relation><Dimension2>'
   Genau wie der letzte Befehl, nur daß zwei Dimensionen vergliechen werden.

3. '\ifodd<Zahl>'
   Testet, ob die übergebene Zahl ungerade ist.

4. '\ifvmode'
   Testet, ob sich TeX gerade im vertikalen, oder im internen vertikalen Modus befindet.

5. '\ifhmode'
   Testet, ob sich TeX gerade im horizontalen, bzw. im eingeschränkten horizontalen Modus befindet.

6. '\ifmmode'
   Testet, ob sich TeX im mathematischen Modus befindet.

7. '\if<Token1><Token2>'
   Die beiden übergebenen Token werden miteinander auf Gleichheit vergleichen. Allerdings nicht uneingeschränkt. Tatsächlich werden die übergebenen Token expandiert, falls das erforderlich ist, und beim ersten nichtexpandierbaren Zeichen wird der Zeichencode getestet.

8. '\ifcat<Token1><Token2>'
   Arbeitet wie der letzte Befehl, nur daß jetzt nicht die Zeichencodes vergleichen werden, sondern die Kategoriecodes.

9. '\ifx<Token1><Token2>'
   Jetzt wird der Vergleich ausgeführt, *ohne* daß die Token zuvor expandiert werden. Handelt es sich um Zeichen, dan wird Zeichen- und Kategoriecode verglichen, sind es beides Makros, dann wird ihr Status verglichen.

10. '\ifvoid<Nummer> \ifhbox<Nummer> \ifvbox<Nummer>'
    Die übergebene Nummer ist eine Zahl zwischen 0 und 255, und die Bedingung ist wahr, wenn das entsprechende Boxregister leer ist, bzw. eine hbox oder vbox enthält.

11. '\ifeof'
    Testet auf Fileende.

12. '\iftrue \iffalse'
    Sind immer wahr, bzw. immer falsch.

Schließlich gibt es noch einen Befehl, der den einfachen Vergleichsbefehlen verwandt ist.

```
\ifcase<Nummer><Text Fall 0>\or
               <Text Fall 1>\or
                     ...
               <Text Fall n>\else
               <Text in allen anderen Situationen>\fi
```

Wenn die übergebene Nummer den Wert 0 hat wird der erste Text ausgeführt, ist der Wert 1, dann der zweite, usf. Paßt keiner der angegebene Fälle zu der Zahl, dann wird der Text hinter dem ‘`\else`’ ausgeführt.

Das Auftreten von ‘`\if`’ und dem beendenden ‘`\fi`’ ist *immer* paarweise, allerdings ist es unabhängig von Gruppierungsklammern.[6]

Wenn man eine eigene Bedingung erst schaffen will, dann gibt es noch den Befehl ‘`\newif`’. Mit

```
\newif\ifabc
```

werden die Befehl ‘`\ifabc`’, ‘`\abctrue`’ und ‘`\abcfalse`’ geschaffen. Mit dem ersten kann die Bedingung abgefragt werden, mit den beiden anderen kann die Bedingung gesetzt werden.

Für die Übergabe von Token an eine Definition oder einen Vergleich kennt TeX noch 256 Tokenregister, die ‘`\toks0`’ bis ‘`\toks255`’ heißen. Die Zuweisung an ein Tokenregister erfolgt mit

```
\toks<Nummer>={Token} oder
\toks<Nummer>=\toks<Nummer}
```

Es gibt natürlich auch hier wieder einen ‘`\newtoks`’ Befehl, der entsprechend den anderen ‘`\new`’ Befehlen arbeitet.

8.5 Expansionen

Wenn TeX einen Text abarbeitet, dann wird Token nach Token eingelesen. Ist eines dieser Token ein Makro, dann wird dieses erst expandiert, bevor TeX seine Arbeit fortsetzt.[7] Es gibt unterschiedliche Kontrollsequenzen, die expandiert werden müssen.

1. *Makros* Wenn ein Makro expandiert wird, dann sucht TeX ggf. erst nach Argumenten und ersetzt dann die Kontrollsequenz durch den angegebenen Ersatztext.

2. *Vergleiche* TeX entscheidet, ob die Bedingung zutrifft, oder nicht und ersetzt dann die Bedingung durch den der Bedingung entsprechenden Text.

3. ‘`\number<Nummer>`’ Die Nummer, die hinter dem ‘`\number`’ Befehl steht wird in eine Ziffernfolge expandiert, je nach Wert der Nummer noch mit einem führenden Minuszeichen.

4. ‘`\romannumeral<Nummer>`’ Komplett analog zum vorherigen Befehl, nur daß jetzt die Zahl in römische Zahlzeichen expandiert wird.

5. ‘`\string<Token>`’ Steht hinter dem Befehl eine Kontrollsequenz, dann wird der Befehl zu der Zeichenkette expandiert, aus der die Kontrollsequenz besteht (inkl. dem führenden Zeichen, entsprechend ‘`\escapechar`’). Handelt es sich nur um ein einfaches Zeichen, dann wird dieses Zeichen als Ergebnis der Expansion ausgegeben.

6. ‘`\jobname`’ Dieser Befehl wird zu dem augenblicklichen Arbeitsnamen expandiert. Wenn TeX z.B. in die Ausgabefiles *Papier.dvi* und *Papier.log* schreibt, dann wird ‘`\jobname`’ zu *Papier* expandiert.

7. ‘`\fontname`’ wird zu dem angegebenen Fontnamen expandiert. Die Angabe von
   ‘`\fontname\tenrm`’
   würde z.B. im Normalfall die Expansion *cmr10* zur Folge haben.

[6] Achtung, hier gibt es die Möglichkeit einige echte Sauereien zu programmieren!

[7] Als Ausnahme gibt es natürlich Befehle, wie `\def` oder `\ifx`.

8. '\meaning<Token>' gibt das aus, was auf dem Bildschirm mit der Anweisung
   '\let\test=<Token>'
   und
   '\show\test'
   ausgegeben würde. Wenn es sich um den Buchstaben 'A' handelt, wäre das Ergebnis: '*The letter A*'

9. '\csname'...'\endcsname' Dieses Befehlspaar ist das Gegenteil des '\string' Befehls. Alle Token zwischen den beiden Befehlen werden solange expandiert, bis nur noch Buchstaben übrig sind. Diese werden dann als Kontrollsequenz interpretiert.

10. '\expandafter<Token>' TEX ließt bei diesem Befehl das Token, das unmittlebar hinter dem Befehl steht ein und expandiert es nicht. Das folgende Token wird expandiert und schließlich das nicht expandierte wieder davor geschrieben.

11. '\noexpand<Token>' Die Expansion ist das Token selber, das aber behandelt wird, wie der Befehl '\relax'.

12. '\topmark, \firstmark, \botmark, \splitfirstmark, \splitbotmark'
    Die Expansionen sind der Inhalt des entsprechenden Markregisters, wie noch besprochen wird.

13. '\input<Filename>' Die Expansion ist leer, aber TEX bereitet sich darauf vor aus den angegebenen File zu lesen.

14. '\endinput' Die Expansion ist wieder leer, aber TEX beendet die Eingabe vom augenblicklichen File, sobald die aktuelle Zeile bearbeitet wurde.

15. '\the<Interne Gr\"o\ss{}e>' Der '\the' Befehl gibt eine Reihe von Token aus, die dem Wert der internen Größe entsprechen. Die Anweisung '\the\skip5' könnte z.B. *5.0pt plus 2.0fil* sein. Der '\the' Befehl hat viele Unterarten, die jetzt zusammen vorgestellt werden sollen.

    (a) '\the<Parameter>' Wobei Parameter eines der internen Parameter von TEX ist. Dies können
        Ganzzahlparameter ('\the\widowpeanlty'),
        Dimensionsparameter ('\the\parindent')
        Leimparameter ('\the\leftskip') oder
        Muglueparameter ('\the\thinmuskip') sein.

    (b) '\the<register>' womit die Inhalte der Zahl-, Dimensions-, Leim- und Mugluregister ausgegeben werden können.

    (c) '\the<Codename><8-Bit Zahl>' Hierbei steht '<Codename>' für
        '\catcode',
        '\mathcode',
        '\lccode',
        '\uccode',
        '\sfcode', oder
        '\delcode'.
        Die Anweisung '\the\mathcode'/' gibt den augenblich gültigen mathematischen Codewert des Schrägstrichs aus.

    (d) '\the<Spezialregister>' Als Spezialregister kommen dabei die folgenden in Frage:
        '\prevgraf, \deadcycles, \insertpenalties',
        \inputlineno, \badness, \parshape'
        oder eine der Dimensionen
        '\pagetotal, \pagegoal, \pagestretch, \pagefilstretch'

'`\pagefillstretch, \pagefilllstretch, \pageshrink, \pagedepth`'
Im horizontalen Modus kann außerdem noch die Dimension '`\spacefactor`' und im vertikalen Modus die Dimension '`\prevdepth`' ausgegeben werden.

(e) '`\the\fontdim<Parameternummer><Font>`' Jeder Font (Zeichensatz) hat seine eigenen Parameter. Der sechste Parameter ist der Wert von 'em'. Mit
'`\the\fontdim6\tenrm`'
erhielte man z.B. *10.0pt*.

(f) '`\the\hyphenchar<Font> \the\skewchar<Font>`' übergibt das entsprechende Zeichen, das für diesen Font definiert wurde.

(g) '`\the\lastpenalty \the\lastkern \the\lastskip`' ergibt die letzte Strafe, den letzten Kern oder den letzten Zwischenraum, vorausgesetzt, daß das letzte Element der aktuellen Liste auch genau das entsprechende Element war.

(h) '`\the<Definiertes Zeichen>`' gibt den ganzzahligen Wert einer Kontrollsequenz zurück, die mit einem der '`\...def`' Befehle zuvor gesetzt wurde.

Normalerweise produziert der '`\the`' Befehl eine Zeichenkette aus ASCII-Zeichen. Nur in einigen Fällen gilt das nicht.

(a) '`\the<Font>`' wählt den angegebenen Font aus. '`\the\font`' wählt z.B. den augenblicklichen Font.

(b) '`\the<Tokenvariable>`' übergibt eine Kopie des Inhalts des Tokenregisters.

Neben dem '`\the`' Befehl existiert auch noch der '`\showthe`' Befehl, der absolut analog arbeitet, nur daß die Ausgabe auf dem Bildschirm erfolgt.

8.5.1 Wann wird nicht expandiert

Es gibt Situationen, in denen eine Kontrollsequenz oder ein Makro nicht expandiert wird. Hier nun eine vollständige Liste all dieser Situationen. Manche Primitive, die hier angesprochen werden sind noch unbekannt, sie werden später besprochen.

1. Wenn ein Token während des Versuchs einen Fehler zu beheben, gelöscht wird.

2. Wenn ein Token in einer Vergleichsanweisung steht, die aufgrund der Bedingung nicht ausgeführt wird.

3. Wenn TeX die Argumente eines Makros liest.

4. Wenn TeX eine Kontrollsequenz einliest, die mit einem der '`\let`' oder einem der '`\def`' Befehle definiert werden soll.

5. Wenn TeX Argumente für einen der Befehle
   '`\expandafter`',
   '`\noexpand`',
   '`\string`',
   '`\meaning`',
   '`\let`',
   '`\futurelet`',
   '`\ifx`',
   '`\show`',
   '`\afterassignment`',
   '`\aftergroup`'
   einliest.

6. Beim Einlesen einer Parameterliste der '`\def`' Befehle.
7. Beim Einlesen eines Ersatztextes, oder beim Einlesen einer Tokenliste, dies geschieht z.B. bei der Bearbeitung von '`\lowercase`'.
8. Beim Einlesen des Vorspanns einer Tabelle.
9. Wenn ein Dollarzeichen eingelesen wurde.
10. Wenn ein '`' einen numerischen Wert einleitet.

Wenn sie erreichen wollen, daß in einer Makrodefinition der Ersatztext nach den obigen Regeln expandiert werden soll, dann verwenden sie den Befehl '`\edef`', statt des normalen '`\def`' Befehls, oder den Befehl '`\xdef`', der wie der '`\edef`' Befehl arbeitet, nur global. Mit der Anweisung

```
\def\doppelt#1{#1#1}
\edef\a{\doppelt{xy}}
```

würden sie erreichen, daß bei Aufruf von '`\a`' *xyxy* ausgegeben würde. Mit '`\noexpand`' wird die Expandierung des folgenden Tokens unterbunden. So können auch in einer expandierenden Definition Teile vor der Expansion geschützt werden.

Bei den Befehlen

```
\mark{...}
\message{...}
\errmessage{...}
\special{...}
\write{...}
```

wird der übergebene Text expandiert, genau wie bei einer '`\edef`' Definition. Allerdings braucht das Parameterzeichen '`#`' *nicht* doppelt verwendet werden, um es einmal zu erhalten, es genügt, im Gegensatz zur Verwendung in Makros, die einmalige Verwendung.

8.5.2 Fileeingabe

Neben der Möglichkeit mit dem '`\input`' Befehl weitere Files einzulesen kann TEX gleichzeitig bis zu 16 Files zur Eingabe verwalten. Um einen derartigen File bereitzustellen, sollten sie ihn mit

```
\openin<Nummer>=<Filename>
```

öffnen. Die Nummern liegen hierbei wieder zwischen 0 und 15, wobei es auch wieder einen '`\newread`' Befehl gibt, der absolut analog zu den entsprechenden Befehlen arbeitet.[8] Wird der File nicht gefunden, oder kann aus einem anderen Grunde nicht gelesen werden, dann gibt TEX *keine* Fehlermeldung aus. Sie müssen mit '`\ifeof`' selber prüfen, ob die Eingabe durch den File möglich ist. Nach dem Einlesen, sollten sie den File mit

```
\closein<Nummer>
```

wieder schließen. Eine Eingabe aus dem File erhalten sie nun mit der Anweisung

```
\read<Nummer>to<Kontrollsequenz>
```

Dabei wird die nächste Zeile des Files eingelesen, oder mehr, falls sich sonst unpaarige geschweifte Klammern ergeben würden, und in der Kontrollsequenz gespeichert. Diese Definition der Kontrollsequenz ist lokal, solange dem '`\read`' kein '`\global`' vorangestellt wird. Mit Zahlen außerhalb den Zulässigen, kann ein Dialog mit dem Benutzer geführt werden.

[8] Normalerweise wird von den meisten Implementationen das `.tex` automatisch angehängt, wenn keine spezielle Extension angegeben ist.

```
\message{Tippen sie ihren Namen}
\read16 to \myname
\message{Hallo, \myname}
```

fordert den Benutzer auf seinen Namen einzugeben, und gibt diesen danach, hinter dem Wort 'Hallo' wieder aus.

8.5.3 Schleifen

TEX bietet auch die Möglichkeit Schleifen zu programmieren. Mit der Anweisung

```
\loop <Teil 1> \if... <Teil 2> \repeat
```

wird folgendes bewirkt. Zunächst wird der erste Teil abgearbeitet, und dann die Bedingung vom '`\if`' getestet.[9] Wenn die Bedingung wahr ist, wird der zweite Teil ausgeführt und dann die ganze Schleife wiederholt. Ist die Bedingung falsch, dann wird der zweite Teil nicht mehr ausgeführt und die Schleife sofort verlassen.[10]

[9] Dies ist die einzige Variante, bei der `\if` *ohne* passendes `\fi` auftritt.

[10] Das TEXbook liefert an dieser Stelle einige bemerkenswerte Beispiele, wie man TEX als Taschenrechner, oder zur Berechnung von Primzahlen einsetzen kann, ich habe mir die Übernahme erspart :-)

Kapitel 9

Weitere Boxbefehle

In diesem Kapitel sollen die letzten, noch unbesprochenen Boxbefehle erläutert werden. Es stellt insofern nur eine Ergänzung der schon bekannten Befehle dar.

9.1 Die Strichboxen

Um eine Strichbox, genauer eine rechteckige, geschwärzte Fläche zu erzeugen, benutzen sie im vertikalen Modus den Befehl ‘`\hbox`’ und im horizontalen Modus den Befehl ‘`\vbox`’, inklusive der Dimensionsangaben, also z.B.

```
\vrule height4pt width3pt depth2pt
```

in einem Paragraphen, und sie erhalten ‘’ als Ergebnis. Wenn eine der Dimensionen doppelt angegeben wird, gilt die zweite Angabe, wird einer der Dimensionen ausgelassen, dann gelten folgende Standardwerte:

	`\hrule`	`\vrule`
Breite	$*$	0.4pt
Höhe	0.4pt	$*$
Tiefe	0.0pt	$*$

Der ‘$*$’ bedeutet, daß sich die Rulebox an der Dimension der kleinsten umschließenden Box orientiert.

Wenn sie z.B. einfach ‘`\hrule`’ schreiben, dann erhalten sie obiges Ergebnis, da die kleinste umschließende Box die Seite selber ist. Mit

```
\hrule width5cm height1pt \vskip1pt \hrule width6cm
```

erhalten sie das Ergebnis *unter* diesem Abschnitt. Zwischen Ruleboxen wird *kein* zusätzlicher Zwischenraum eingefügt. Die Zeilen sind genau ein pt voneinander entfernt.

Sie müssen sich schon an die Vorgabe halten, im vertikalen Modus nur horizontale Boxen zu erzeugen (‘`\hrule`’) und im horizontalen Modus nur vertikale (‘`\vrule`’). Auch wenn die Dimensionsangaben übereinstimmen und deshalb sich die Boxen durch nichts unterscheiden, passiert folgendes. Wenn sie im vertikalen Modus den Befehl ‘`\vrule`’ verwenden, beginnt TeX einen neuen Paragraphen. Verwenden sie ‘`\hrule`’ im horizontalen Modus, dann beendet TeX den augenblicklichen Paragraphen und geht in den vertikalen Modus über.

Negative Angaben bei einer Rulebox sind zuläßig und sinnvoll. Die Box ‘’ hat beispielsweise eine Tiefe von -2pt. Allerdings muß die Breite positiv sein und auch die Summe von Höhe und Tiefe muß positiv sein, damit überhaupt eine Box gezeichnet wird.

9.2 Boxtypen

Es gibt verschiedene Typen von Boxen: (1) Jedes Zeichen stellt für sich eine Box dar. (2) Die Befehle '`\hrule`' und '`\vrule`' erzeugen Boxen und (3) lassen sich Boxen auch explizit angeben

```
\hbox<Box Spez.>{<Horizontales Material>}
\vbox<Box Spez.>{<Vertikales Material>}
\vtop<Box Spez.>{<Vertikales Material>}
\box<Registernummer>
\copy<Registernummer>
\vsplit<Registernummer> to <Dimension>
\lastbox
```

Die '<Box. Spez.>' sind dabei die Angaben 'to <Dimension>' oder 'spread <Dimension>', sie können natürlich auch wegfallen. Der letzte Befehl ist bisher noch unbekannt. Außer man befindet sich im normalen vertikalen Modus, enthält '`\lastbox`' immer dann eine Box, wenn das letzte Element der aktuellen Liste eine Box war, sonst, und auch im mathematischen Modus ist sie leer.

Der '`\unskip`' Befehl ähnelt dem '`\lastbox`' Befehl, außer daß er sich auf Leim und nicht auf Boxen bezieht. Im vertikalen Modus erbringt der Befehl '`\unskip`' nicht die gewünschte Wirkung, also die Entfernung des letzten Leimstückes, aber mit

```
\vskip-\lastskip
```

kann derselbe Effekt erreicht werden.

In den zusammenfassenden Kapiteln am Ende dieses Buches wird unter einer Box immer eine der explizit angegebenen verstanden, da Zeichen(boxen) und Ruleboxen so speziell sind, daß sie nicht wie die anderen Boxen behandelt werden.

9.3 Leaders

Was sie in dieser Zeile sehen . . . . wird Leader genannt.[1] Leader sind der übergeordnete Begriff für Leim. Leim füllt einen Zwischenraum mit Leerzeichen, bei Leadern kann der Füllbuchstabe angegeben werden. Der Befehl für Leader lautet

```
\leaders<Box oder Rule>\hskip<Leim>
```

Der Effekt ist genauso, als hätten sie nur '\hskip<Leim>' angegeben, mit der Ausnahme, daß der Zwischenraum mit Kopien der Box gefüllt wird. Die Eingabe von

```
\def\leaderfill{\leaders\hbox to 1em{\hss.\hss}\hfill}
\line{$\alpha$\leaderfill $\omega$}
\line{Anfang\leaderfill Ende}
```

ergibt

α . . . . ω
Anfang . . . . Ende

Die Angabe von '`\hbox to 1em{\hss .\hss}`' erzeugt eine Box mit einem em Breite, in dessen Mitte ein Punkt steht. Mit diesen Boxen wird dann die Zeile aufgefüllt. Der Grund dafür, daß die Punkte genau übereinander liegen ist darin zu suchen, daß der Leaderbefehl sozusagen nur einen kleinen Ausschnitt von einer ganzen Reihe von Kopien dieser Box zeigt. Dies ermöglicht die Justierung der einzelnen Boxen auch übereinander. Leider hat dieses Verfahren auch einen kleinen

[1] Vom englischen *lead* für leiten, da sie die Augen entlang einer Zeile leiten sollen.

Nachteil. Vor dem ersten und hinter dem letzten Punkt kann ein etwas zu großer Zwischenraum stehen. Insbesondere, wenn ein Leader alleine auftaucht, kann das etwas störend sein. Dazu gibt es die Befehle '`\cleaders`' und '`xleadres`'. Mit dem ersten Befehl werden die Boxen insgesamt in dem Freiraum zentriert, mit dem zweiten werden die Boxen einzeln auf die passende Länge gezogen.

Statt einer Box kann auch eine Rulebox angegeben werden, evtl. mit Angaben über die Höhe, Breite und Tiefe. Hier macht dann auch eine '`\hbox`' im horizontalen Modus Sinn, da so eine horizontale Linie ausgegeben werden kann. Hätten wir die vorige Definition als:

```
\def\leaderfill{\leaders\hbox to 1em{\hrule}\hfill}
```

geschrieben, dann wäre das Ergebnis

α ____ ω

Anfang ____ Ende

gewesen.

Leaders können natürlich genauso im vertikalen Modus verwendet werden, wobei, wie auch im horizontalen Modus kein Zwischenraum zwischen den einzelnen Boxen gelassen wird. Wenn sie Boxen angeben, die eine negative Breite, oder eine negative Gesamthöhe (Höhe plus Tiefe) haben, ignoriert TeX die Angabe und verwendet einfach Leim.

9.3.1 Anwendungen

Liegende geschweifte Klammern

Mit den Befehlen '`\upbracefill`' und '`\downbracefill`' können ganze Textteile durch eine geschweifte Klammer überzogen, oder unterstrichen werden. Die vier Symbole '$\braceld$', '$\bracerd$', '$\bracelu$' und '$\braceru$' werden dabei durch Ruleleader verbunden.

$$\overbrace{\hspace{7cm}}$$
$$\underbrace{\hspace{3.5cm}}$$

Pfeile

Auch Pfeile lassen sich natürlich auf diese Art verlängern. Würde man eine Verlängerung eines Pfeilsymbols mit Minuszeichen oder Gleichheitszeichen versuchen, dann würden die einzelnen Zeichen durch kleine Leerstellen getrennt. Statt dessen dient der Befehl

```
\hbox to 100pt{\rightarrowfill}
```

zu dem Ergebnis '$\xrightarrow{\hspace{3cm}}$'

Vertikale Anwendung

Um auch die Anwendung vertikaler Leader einmal zu zeigen dient diese Textur

TeXTeXTeXTeXTeXTeXTeXTeXTeX
TeXTeXTeXTeXTeXTeXTeXTeXTeX
TeXTeXTeXTeXTeXTeXTeXTeXTeX
TeXTeXTeXTeXTeXTeXTeXTeXTeX

Sie wurde mit

```
$$\hbox to 2.5in{\cleaders
     \vbox to .5in{\cleaders\hbox{\TeX}\vfil}\hfil}$$
```

erzeugt.

9.4 WasDenn

Schon oft wurden sie erwähnt, hier nun endlich soll geklärt werden, worum es sich dabei handelt. Es handelt sich dabei um eine Möglichkeit Eigenschaften von TEX zu ändern, ohne dabei den Sourcecode von TEX selber allzu sehr zu modifizieren. Von dieser Möglichkeit sollte allerdings nicht allzuviel Gebrauch gemacht werden, um die Kompatibilität der Texte nicht zu gefährden. In allen TEX Implementationen sind allerdings mindestens zwei 'WasDenn' mit eingebaut

1. Die Fileausgabe.
2. Der `\special` Befehl.

9.4.1 Fileausgabe

Die Möglichkeit der Fileausgabe erlaubt die Verwendung der Ausgabe durch andere Programme, inklusive TEX selber. Sie wird z.B. für Inhaltsverzeichnisse und ähnliche Anwendungen gebraucht. Analog zur Fileeingabe gibt es die Befehle

```
\openout<Nummer>=<Filename>
\closeout<Nummer>
\write<Nummer>{<Tokenliste>}
\newwrite
```

Ist die Nummer größer als 15 oder kleiner als Null, dann wird wieder in den Logfile, bzw. auf dem Bildschirmen ausgegeben. Es gibt aber einen wesentlichen Unterschied, zu den entsprechenden Eingabebefehlen. Die Ausgabe erfolgt nicht beim Auftreten des Befehls, sondern bei dem Auftreten des WasDenn bei der eigentlichen Ausgabe der Seite an den Dvi-File. Eine sofortige Ausgabe kann durch unmittlebares voranstellen des Befehls '`\immediate`' vor dem Schreibbefehl erreicht werden. Die Anweisung

```
\immediate\write16{Auf Wiedersehen!}
```

Bewirkt die Ausgabe von 'Auf Wiedersehen!' auf dem Bildschirm, ähnlich, wie auch der '`\message`' Befehl, allerdings wird mit dem '`\write`' Befehl immer eine eigene Zeile ausgegeben.

9.4.2 Der \special Befehl

Der Befehl '`\special`' erlaubt es eine (expandierte) Tokenliste in den Dvi-File zu schreiben. Außerdem wird im Dvi-File auch vermerkt, an welcher Stelle im Originalfile der Befehl stand. Diese expandierte Tokenliste kann dann z.B. durch ein Ausgabeprogramm ausgewertet werden. Bislang gibt es *keine* Standardisierung für die `\special` Befehle.

Kapitel 10

Tabellen und Ausrichtungen

Die so oft vorkommenden Tabellen können in TEX auf zwei verschiedene Arten gesetzt werden. Zum einen kennt TEX eine Tabelle, und dann das Werkzeug der horizontalen Ausrichtung. Zunächst die Tabellen.

10.1 Tabellen

10.1.1 Gleiche Spaltenbreite

Mit der Anweisung '`\settabs n \columns`' wird die laufende Zeile in n Spalten gebrochen. Jede der so geschaffenen Spalten ist gleich breit.[1] Die einzelnen Zeilen dieser Tabelle werden dann mit:

```
\+ <Text 1> & <Text 2> & .... \cr
```

eingegeben. Die Eingabe von

```
\settabs 4 \columns
\+&&Text in der dritten Spalte\cr
\+&Text in der zweiten Spalte\cr
\+\it Text in der ersten Spalte&&&und in der vierten Spalte\cr
```

hat folgende Wirkung:

		Text 3. Spalte	
	Text 2. Spalte		
Text in der 1. Spalte			und 4. Spalte

Dieses Beispiel zeigt einige wichtige Eigenschaften der Tabulatorbefehle von TEX.

1. Die `&` entsprechen den Tabulatorstopps einer normalen Schreibmaschine.

2. Der Unterschied besteht darin, daß es *immer* genausoviele Tabstopps in einer Zeile gibt, wie es `&` Zeichen gibt, besonders gut an der untersten Zeile zu sehen.

3. Jede Zeile wird mit einem '`\cr`' beendet. Dies kann auch geschehen, wenn noch nicht alle Spalten mit `&` getrennt sind. Es kann also weniger als die angegebene Anzahl von Spalten geben.

4. Leerzeichen hinter '`&`' und '`\+`' werden ignoriert.

5. Jeder Spalteneintrag stellt eine eigene Gruppe dar. Das ist besonders an der ersten Spalte der letzten Zeile zu sehen, bei der für die lokale Schriftumschaltung *keine* Gruppenklammern verwendet werden brauchten.

[1] Einen entsprechenden Befehl gibt es in LATEX nicht!

Normalerweise bleiben die Spalten, die sie mit '`\settabs`' eingerichtet haben bestehen, auch wenn sie danach wieder normalen Text schreiben, der in normale Paragraphen umgebrochen wird. Sie können die Tabulatoren natürlich auch innerhalb einer Gruppe definieren, dann gelten sie nur dort. Eine Globalisierung durch '`\global`' ist *nicht* zuläßig.

Normalerweise werden Tabellen zwischen Paragraphen gesetzt. Es ist aber auch sinnvoll sie in `vbox`en zu gebrauchen. Damit sind z.B. Tabellen in abgesetzten Formeln möglich. Mit

```
$$\vbox{\settabs 3 \columns
  \+Dies ist&ein seltsames&Beispiel\cr
  \+mit einem&drei Spalten&Format.\cr}$$
```

erhalten sie

Dies ist	ein seltsames	Beispiel
mit einem	drei Spalten	Format.

10.1.2 Unterschiedliche Spaltenbreite

Es ist natürlich nicht immer wünschenswert, daß die Spalten alle dieselbe Breite haben. Für diesen Fall muß der Tabulatorbefehl etwas abgeändert werden. Dazu dient eine *Musterzeile*, die unmittelbar hinter den '`\settabs`' Befehl geschrieben wird und auch mit '`\cr`' abgeschlossen wird.

```
\settabs\+\indent&Horizontale Liste\quad&\cr % sample line
\+&Horizontale Liste&Kapitel 14\cr
\+&Vertikale Liste&Kapitel 15\cr
\+&Math. Liste&Kapitel 17\cr
```

Ist ein Beispiel für eine derartige Liste. Die Musterzeile wird dabei *nicht* ausgegeben, so daß das Ergebnis so aussieht

Horizontale Liste	Kapitel 14
Vertikale Liste	Kapitel 15
Math. Liste	Kapitel 17

Bei der Konstruktion einer längeren Tabelle mit dieser Methode muß man darauf achten, daß man den jeweils längsten Spalteneintrag in die Musterzeile schreibt.

Mit Einfügung von '`\hfill`' an geeigneten Stellen kann der Text auch andere Ausrichtungen, als die Linksbündigkeit erhalten. In *Plain* TEX wird definiert, daß jeder Spalteneintrag in eine Hbox mit der Breite einer Spalte geschrieben wird, gefolgt von dem Befehl '`\hss`'. '`\hfill`' ist der mächtigere Befehl, so daß damit die Wirkung von '`\hss`' unterbunden werden kann. Eine einfache Zentrierung ist allerdings mit einem vorgestellten '`\hss`' möglich, das nachgestellte wird ja automatisch eingefügt. Dies geht allerdings nur in den ersten Spalten. In der letzten Spalte wird der Text in eine Hbox geschrieben, die die natürliche Breite des Textes hat.

Eine sehr gute Anwendungsmöglichkeit dieser variablen Tabulatorstops bietet die Dokumentation von Programmsourcen. Stellen sie sich vor, sie sollten folgendes Computerprogramm in der vorgestellten Weise setzen.

if $n < r$ **then** $n = n + 1$
　　　else begin $print$; $n := 0$;
　　　　　end;
while $p > 0$ **do**
　begin $q := link(p)$; $free(p)$; $p := q$;
　end;

Es wäre ziemlich mühsam für jeder der vorkommenden Tabulatorpositionen einen Tabulatorstop zu setzen. Es ist nun zwar so, daß man durch einfügen eines weiteren '`&`' jederzeit eine neue Position für eine Spalte setzen kann, aber auch diese Methode wäre noch ziemlich unübersichtlich. Es gibt einen noch einfacheren Weg das obige Ergebnis zu erreichen. Die gesetzten Tabulatorpositionen können mit dem Befehl '`\cleartabs`' auch alle wieder gelöscht werden, um dann neue zu setzen. Somit konnte das obige Programm einfach mit

```
$$\vbox{\+\bf if $n<r$ \cleartabs&\bf then $n=n+1$\cr
\+&\bf else & {\bf begin} ${\it print}$; $n:=0$;\cr
\+&&{\bf end};\cr
\+\cleartabs {\bf while} $p>0$ {\bf do}\cr
\+\quad&{\bf begin} $q:=link(p)$; $free(p)$; $p:=q$;\cr
\+&\{\bf end};\cr
```

gesetzt werden.

Es dürfen keine '`\+`' Zeilen geschachtelt werden. Sie können also sozusagen keine Tabellen innerhalb von Tabellen definieren. Da diese Tabellen in Boxen gleicher Breite aufgebaut werden, können sie mit der Befehlsfolge '`\showbox\tabs`' genau ausgeben lassen, was bei der Konstruktion von Tabellen geschieht.

10.2 Ausrichtungen

Es gibt noch eine weitere Möglichkeit in TeX Tabellen zu setzen. Sie basiert auf der Idee einer Schablone. Mit der Eingabe von

```
\halign{\indent#\hfil&\quad#\hfil\cr
Horizontale Liste&Kapitel 14\\
Vertikale Liste&Kapitel 15\\
Math. Liste&Kapitel 17\cr}
```

erhält man dasselbe Ergebnis, wie schon zuvor mit dem '`\+`' Befehl, nämlich:

Horizontale Liste	Kapitel 14
Vertikale Liste	Kapitel 15
Math. Liste	Kapitel 17

Dieses Beispiel zeigt aber doch im wesentlichen, wie der '`\halign`' Befehl funktioniert.[2] Die erste Zeile, nicht im physikalischen Sinne, sondern der Text bis zum ersten '`\cr`', stellt die Schablone für die weiteren Aktivitäten dar. Dieses Beispiel enthält zwei Mustereinträge in der Schablone, nämlich

`\indent#\hfil` und `\quad#\hfil`

Jeder dieser Mustereinträge enthält genau ein '`#`' Zeichen, an der später der Text der Spalte eingefügt wird. In der ersten Zeile wird so der Eintrag

```
\indent Horizontale Liste \hfil
```

eingetragen. Nun zur Frage, was denn das '`\hfil`' in dem Eintrag soll. Nun. Im '`\halign`' Befehl werden zunächst *alle* Spalten eingelesen, und TeX setzt dann die Breite der Spalten nach dem breitesten vorkommenden Eintrag. Da ein aktueller Eintrag ggf. schmaler ist, als der breiteste, muß mit einem '`\hfil`' der restliche Platz aufgefüllt werden, um keine Fehlermeldung zu erhalten.

Es gibt einige Unterschiede zwischen dem '`\halign`' Befehl und dem '`\+`' Befehl.

1. Beim dem Befehl zur horizontalen Ausrichtung wird die Breite der Spalten nach dem breitesten Eintrag automatisch berechnet.

[2] halign kommt von 'horizontal alignment', was soviel beeutet, wie 'horizontale Ausrichtung'.

2. Jeder '`\halign`' Befehl berechnet seine Spaltenbreiten für sich. Wenn sie in zwei verschiedenen Tabellen die selbe Ausrichtung haben wollen, dann müssen sie sich schon einiger Tricks bedienen.

3. Große Tabellen lassen sich schlecht mit '`\halign`' erstellen, da die ganze Tabelle im Speicher gehalten werden muß und es deshalb zu Problemen kommen kann.

4. Der '`\halign`' Befehl ist wesentlich schneller, als der '`\+`' Befehl, da es sich um eine eingebaute Funktion von TEX und nicht um ein Makro handelt.

5. Man kann sich die Eingabe vereinfachen. Statt der obigen Eingabe wäre z.B.

   ```
   \halign{\indent# Liste\hfil&\quad Kapitel #\cr
   Horizontale&14\cr Vertikale&15\cr Math.&17\cr}
   ```

   auch möglich gewesen. Man sollte sich allerdings nicht *zu* viel Mühe mit der Optimierung geben (man hätte die '1' der Kapitelnummer noch mit in die Schablone übernehmen können), da die Zeit, die man dafür aufwenden muß in keinem Verhältnis zum Erfolg steht.

6. Wenn die Spaltenbreiten häufig wechseln, so wie in dem Programmbeispiel, sollte man lieber mit dem '`\+`' Befehl arbeiten.

10.2.1 Beispiel zur Anwendung

Der wichtigste Punkt bei der Erstellung von Tabellen mit dem '`\halign`' Befehl besteht in der günstigen Konstruktion der Schablone. Hier sollte man ruhig etwas Zeit investieren, die sich später erst auszahlt. Wenn sie z.B. die folgende Tabelle setzen wollen

Name	*Geschlecht*	*Alter*	*Schul-bildung*
Karl	*männlich*	20	Hauptschule
Tina	*weiblich*	22	Abitur
Otto	*männlich*	21	Mittl. Reife
Jupp	*männlich*	20	Abitur

Dann wünscht man sich natürlich eine einfache Eingabe. Die zweite Zeile sollte am besten mit

```
Tina & weiblich & 22 & Abitur\cr
```

eingegeben werden können. Die oberste Zeile muß natürlich gesondert behandelt werden, aber es soll möglich sein, daß die rechte Ausrichtung der ersten Spalte, die linke der letzten und die Zentrierung der mittleren Spalten automatisch vorgenommen wird. Ebenso wäre es wünschenswert, wenn die Schriftumschaltung in der ersten und der zweiten Spalte automatisch erfolgen würde. Dies ist tatsächlich möglich mit folgender Schablone

```
\halign{\hfil\bf#&\quad\hfil\it#\hfil&\quad\hfil#\hfil&\quad#\hfil\cr
```

Jetzt brauchen sie nur noch die Zeilen, entsprechend der 'Tina'-Zeile einzugeben, und sie erhalten das gewünschte Ergebnis. In der Titelzeile ist allerdings bei jedem Eintrag die Angabe der Schriftform notwendig, um die Vorgaben zu überschreiben.

Die Tabelle kann übrigens mit dem Befehl '`\openup`' etwas auseinandergezogen werden. Mit der Angabe '`\openup2pt`' entsteht

Name	*Geschlecht*	*Alter*	*Schul-bildung*
Karl	*männlich*	20	Hauptschule
Tina	*weiblich*	22	Abitur
Otto	*männlich*	21	Mittl. Reife
Jupp	*männlich*	20	Abitur

Der zusätzliche Zwischenraum zwischen der Titelzeile und dem Rest der Tabelle wurde mit der Angabe von

```
\noalign{\smallskip}
```

erreicht. Grundsätzlich ist es immer möglich mit diesem Befehl vertikale Elemente in eine Tabelle einzufügen, die nicht zu der Tabelle selber gehören. Von dieser Möglichkeit kann man Gebrauch machen, wenn man etwas zwischen zwei Zeilen schreiben möchte. Mit der Eingabe von

```
\halign{#\hfil&\quad#\hfil\cr
Erste & Spalte \cr\noalign{Dazwischen}
Zweite & Spalte\cr}
```

erhalten sie z.B.

Erste	Zeile
Dazwischen	
Zweite	Zeile

10.2.2 Weitere Befehle

Es ist auch möglich, daß sie TEX entscheiden lassen, wieviel Platz zwischen die Tabellenspalten gesetzt werden soll. Sie können also auf den '`\quad`' Befehl zwischen den Spalten verzichten. Statt dessen geben sie an, wieviel `tabskipglue` zwischen den Spalten verwendet werden soll. Wenn sie also statt der obigen Schablone

```
\tabskip=1em plus 2em minus.5em
\halign{\hfil\bf#&\hfil\it#\hfil&\hfil#\hfil&#\hfil\cr
```

schreiben, dann erhalten sie trotzdem die Tabelle in der Form, wie sie sie schon gesehen haben, als

Name	*Geschlecht*	*Alter*	*Schul-bildung*
Karl	*männlich*	20	Hauptschule
Tina	*weiblich*	22	Abitur
Otto	*männlich*	21	Mittl. Reife
Jupp	*männlich*	20	Abitur

Der Tabskip-Leim wird auch vor sie erste und hinter die letzte Spalte gesetzt, so daß sie mit

```
\halign to...        \halign spread
```

genauso verfahren können, wie bei den entsprechenden Boxbefehlen.

Die Befehlsfolge '`\halign to \hsize`' wird allerdings nur dann richtig verstanden, wenn der Tabskip-Leim auch die nötige Fähigkeit zur Streckung besitzt. Falls dies nicht der Fall ist wird eine Fehlermeldung auf dem Bildschirm und im Logfile ausgegeben. Das geschieht auch, wenn ein Eintrag so groß oder klein ist, daß die angegebene Größe des Leims nicht ausreicht um eine passende Spalte daraus zu machen.

Die Werte für den Tabskip-Leim können auch in der Schablone geändert werden. Es müssen also nicht für alle Spalten die gleichen Werte verwendet werden. Damit können sie einzelne Spalten von den anderen absetzen.

TEX übernimmt die Schablone 1 : 1, d.h. daß nichts in der Schablone expandiert wird. Die einzigen Befehle, die TEX innerhalb der Schablone erkennt sind

```
&  #  \cr  \span  \tabskip
```

Der Befehl '`\span`' sorgt dafür, daß das nächste Token in der Schablone *doch* expandiert wird.[3] Dies führt zu einem besonderen Verhalten aller '`\if`' Befehle innerhalb von Tabellen. Verwenden sie diese Befehle besser nicht innerhalb von Tabellen, wenn sie Probleme vermeiden wollen.

Es kann vorkommen, daß alle Spalten, bis auf eine oder zwei eine ähnliche Form haben, so daß sich entsprechende Muster in der Schablone empfehlen. Für die Einträge, die dieser Form nicht entsprechen, gibt es den Befehl '`\omit`'. Wenn er als erstes im Eintrag steht, wird das Muster aus der Schablone durch ein einfaches '`#`' ersetzt.

10.2.3 Besondere Tabellen

Dezimalpunkte

Wie geht man vor, wenn man eine Tabelle schreiben will, die z.B. Geldwerte enthalten soll, bei denen natürlich der Dezimalpunkt übereinander stehen soll. Man kann natürlich aus jeder Zahl drei Spalten machen, und die Zwischenräume so setzen, daß die mittlere Spalte den Dezimalpunkt enthält und trotzdem noch gut aussieht. Es gibt aber auch eine andere Möglichkeit. Der Trick besteht darin, daß alle Ziffernzeichen dieselbe Breite haben. Man kann nun ein Zeichen, das sonst nicht gebraucht wird zu einem aktiven Zeichen machen und durch dieses Zeichen einen Leerraum ausgeben lassen, der genau der Breite eines Ziffernzeichens entspricht. Wenn man mit der Befehlsfolge

```
\newdimen\digitwidth
\setbox0=\hbox{\rm0}
\digitwidth=\wd0
\catcode`?=\active
\def?{\kern\digitwidth}
```

Das Zeichen '`?`' zu einem aktiven Zeichen gemacht hat, dann kann man die Zahlen 123.45 und 6.789 als '`123.45?`' und '`??6.789`' in dem Eintrag eingeben, und sie werden beide genau mit dem Dezimalpunkt übereinander gesetzt.

Lange Tabellen

Was ist mit Tabellen, wie dieser

n=0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
$\mathcal{G}(n)$=1	2	4	3	6	7	8	16	18	25	32	11	64	31	...

Mit Sicherheit wäre es höchst mühsam für diese Tabelle eine Schablone zu schreiben, insbesondere, da die Tabelle nur zwei Zeilen hat. Es gibt aber einen einfacheren Weg. Wenn sie an einer Stelle der Schablone ein doppeltes '`&`' schreiben, dann setzt TEX die folgenden Einträge bis zum '`\cr`' beliebig oft fort.

$t_1 \,\&\, t_2 \,\&\, t_3 \,\&\&\, t_4 \,\&\, t_5$ `\cr` wird verstanden als $t_1 \,\&\, t_2 \,\&\, t_3 \,\&\, t_4 \,\&\, t_5 \,\&\, t_4 \,\&\, t_5 \,\&\, t_4 \,\& \cdots$
und
$\&t_1 \,\&\, t_2 \,\&\, t_3 \,\&\, t_4 \,\&\, t_5$ `\cr` wird verstanden als $t_1 \,\&\, t_2 \,\&\, t_3 \,\&\, t_4 \,\&\, t_5 \,\&\, t_1 \,\&\, t_2 \,\&\, t_3 \,\& \cdots .$

Der evtl. gesetzte Tabskip-Leim wird in die Kopie übernommen. Somit läßt sich die Schablone der obigen Tabelle als

```
$\hfil#$ =&&\ \hfil#\hfil\cr
```

schreiben.

10.2.4 Breite Einträge

Mitunter ist es nötig, daß sich ein Eintrag einer Tabelle über mehr als eine Spalte erstreckt. Für diesen Fall bieten sich zwei Möglichkeiten an.

[3] *Innerhalb* der Tabelle hat der Befehl eine andere Wirkung, die später erklärt wird.

Verstecken der Breite

In *Plain* TeX wird der Befehl '`\hidewidth`' als

```
\hskip-1000pt plus 1fill
```

definiert. Er hat also eine extreme negative, natürliche Breite, kann aber beliebig gestreckt werden. Wenn sie diesen Befehl rechts hinter einen Spalteneintrag schreiben, erscheint es für TeX, als habe er keine Breite, und der Eintrag ragt rechts über seine Box hinaus. Analog arbeitet der Befehl, wenn er links vor einen Eintrag geschrieben wird. Später wird noch eine nützliche Eigenschaft dieses Befehls gezeigt, wenn er rechts *und* links vom Eintrag steht.

Einträge über mehrere Spalten

Statt der obigen Methode können sie auch den Befehl '`\span`' anstelle des '`&`' verwenden.[4] TeX behandelt die Einträge vor und hinter diesem Befehl genauso, als wenn dort ein '`&`' stünde, schreibt sie aber anschließend in eine Box, die sich über die Breite zweier Einträge erstreckt.

Sie können den Befehl zusammen mit '`\omit`' verwenden, um eventuell auftretende automatische Einträge zu unterbinden. Schließlich gibt es noch den Befehl '`\multispan`', dem eine Nummer folgen muß, die angibt, wieviele Spalten zusammengefaßt werden sollen. Sollen mehr als 9 Spalten zusammengefaßt werden, muß die Zahl in geschweiften Klammern angegeben werden.

Sie können sich die Wirkung der Befehle noch einmal am folgenden Beispiel verdeutlichen.

<table>
<tr><td>Eins</td><td>Zwei</td><td>Drei</td></tr>
<tr><td colspan="2">Eins-Zwei</td><td>Drei</td></tr>
<tr><td>Eins</td><td colspan="2">Zwei-Drei</td></tr>
<tr><td colspan="3">Eins-Zwei-Drei</td></tr>
</table>

wurde mit

```
$$\tabskip=3em
\vbox{\halign{&\hfil#\hfil\cr
Eins&Zwei&Drei\cr
Eins-\span Zwei&Drei\cr
Eins&Zwei-\span Drei\cr
\multispan3 Eins-Zwei-Drei\cr}}$$
```

erzeugt.

10.2.5 Gerahmte Tabellen

D. Knuth spricht davon, daß die Erstellung von gerahmten Tabellen mit zum Schwierigsten gehört, was in TeX realisiert werden kann. Tatsächlich ergeben sich *einige* Schwierigkeiten, wenn man sich genauer mit diesem Thema beschäftigt.[5]

Zunächst müssen sie den Zwischenraum zwischen den Zeilen ausschalten, und den nötigen Zeilenabstand durch 'Struts' in der Schablone aufbauen, da sonst die senkrechten Linien nicht aneinander stoßen würden. Dies geschieht mit dem Befehl '`\offinterlineskip`'. Die Höhe der senkrechte Linie braucht nicht spezifiziert zu werden, da TeX die Boxhöhe vorgibt und sich der unspezifizierte Befehl an die umgebende Box anpaßt. Allerdings muß jeder senkrechte Strich in eine eigene Spalte geschrieben werden. Im TeXBook findet sich folgendes Beispiel

---

[4] Hier hat dieser Befehl eine *ganz* andere Wirkung, als in der Schablone!

[5] Die Schwierigkeiten sind so groß, daß ich jedem nur empfehlen kann, nur im äussersten Notfall *nicht* die Tabular-Umgebung von LaTeX zu benutzen.

```
\vbox{\offinterlineskip
\hrule
\halign{&\vrule#&
  \strut\quad\hfil#\quad\cr
\multispan5\hrulefill\cr
height2pt&\omit&&\omit&\cr
&Year\hfil&&World Population&\cr
height2pt&\omit&&\omit&\cr
\multispan5\hrulefill\cr
height2pt&\omit&&\omit&\cr
&8000\BC&&5,000,000&\cr
&50\AD&&200,000,000&\cr
&1650\AD&&500,000,000&\cr
&1850\AD&&1,000,000,000&\cr
&1945\AD&&2,300,000,000&\cr
&1980\AD&&4,400,000,000&\cr
height2pt&\omit&&\omit&\cr}
\hrule}
```

An diesem Beispiel können sie sehen, wie vergleichsweise kümmerlich die Befehle für gerahmte Tabellen in *Plain* TEX tatsächlich ist, denn das Ergebnis ist gerade mal

Year	World Population
8000 B.C.	5,000,000
50 A.D.	200,000,000
1650 A.D.	500,000,000
1850 A.D.	1,000,000,000
1945 A.D.	2,300,000,000
1980 A.D.	4,400,000,000

Etwas verwirrend könnten noch die Zeilen

```
height2pt&\omit&&\omit&\cr
```

sein. Sie bewirken, daß auch der Anschluß zu den waagerechten Zeilen geschafft wird.

10.2.6 Hilfen bei Tabellen

Wenn bei der Konstruktion einer Tabelle etwas schief geht, empfiehlt sich folgendes Vorgehen. Fügen sie an einer Stelle, an der sie den Fehler vermuten eine unbekannte Kontrollsequenz ein, ich benutze für diese Zwecke immer '`\fritz`'. Dann können sie mit dem Befehl '`\showlists`' eine genauer Information darüber bekommen, was TEX an dieser Stelle gerade macht.

Eine weitere Hilfe interessiert wohl mehr die Makrodesigner. Jede Zeile eines Alignments *muß* mit dem Befehl '`\cr`' abgeschlossen werden. Sie können durch das Primitv '`\crcr`' dem Fall vorbeugen, daß ein Anwender diesen '`\cr`' Befehl vergißt. Die Wirkung dieses Befehls ist folgende. Steht er unmittelbar hinter einem '`\cr`', dann wirkt er gar nicht, steht er woanders, dann wirkt er wie ein '`\cr`' Befehl.

Wenn ihnen selber die andauernde Tipperei der '`\cr`' Befehle zuviel wird, dann gibt es noch eine Möglichkeit. Schließen sie ihre Tabelle in eine eigene Gruppe ein und stellen sie dann innerhalb der Gruppe der Tabelle die Befehle

```
\let\par=\cr
\obeylines
```

voran. Durch den '`\obeylines`' Befehl wird jede Zeile wie ein eigener Absatz behandelt. Das automatisch eingefügte '`\par`' wird als '`\cr`' interpretiert. Diese Vorgehensweise empfiehlt sich aber nur bei längeren Tabellen, die sie aber sowieso mit '`\+`' ausführen sollten. Dort funktioniert dieser Trick aber leider nicht.

Kapitel 11

Ausgaberoutinen

In einem vorhergehenden Kapitel haben sie schon gesehen, wie TEX aus der langen Reihe von Token zunächst Paragraphen und dann Seiten macht. Diese Seiten bestehen zunächst allerdings nur aus dem eigentlichen Seitentext und den Einfügungen, z.B. Fußnoten. Was noch fehlt sind die Kopf- und Fußzeilen.[1] Diese Ergänzungen werden von einer sogenannten Outputroutine noch zu der Seite hinzugefügt.

11.1 Die Outputroutine von *Plain* TEX

Normalerweise ist eine Seite von *Plain* TEX $8\frac{1}{2}$ Inch breit und 11 Inch hoch, dazu ein Rand von einem Inch an allen vier Seiten. Diese Werte können natürlich verändert werden. Dazu dienen die Befehle '`\hsize`' und '`\vsize`'. *Plain* TEX setzt '`\hsize=6.5in`' und erhält damit eine Seitenbreite von $8\frac{1}{2}$ Inch (denken sie an die beiden Ränder von einem Inch und '`\vsize=8.9in`'. Nicht, wie erwartet '`9in`', da *Plain* TEX keinen Platz für die Seitennummer reserviert.

Sie sollten diese beiden Werte nicht zu oft ändern. Am besten nur ganz zu Beginn eines Textes, oder wenn sie sicher sind, daß TEX alle Seiten ausgegeben hat.

Sie können darüber hinaus auch noch die gesamte Seite auf dem Papier verschieben und zwar mit den Befehlen '`\hoffset`' und '`\voffset`'. Die Angabe

```
\hoffset=.5in \voffset=-1in
```

verschiebt ihre Seite um ein halbes Inch nach rechts und ein Inch nach oben. Sie sollten aber auf die physikalischen Grenzen ihres Mediums, meist Papiers, achten, da TEX dafür keine Kontrolle bereitstellt.

Die einfachste Art die Outputroutine von *Plain* TEX zu ändern besteht in der Angabe von

```
\nopagenumbers
```

Mit diesem Befehl verhindern sie die Ausgabe von Seitennummern, was vor allem bei kurzen Texten recht hilfreich sein dürfte. Dies ist aber nur ein Spezialfall. Generell können die Kopf- und die Fußzeile einer Seite mit den Befehlen

```
\headline={<Kopfzeile>}
\footline={<Fusszeile>}
```

gesetzt werden. Mit '`\headline={\hrulefill}`' würde z.B. vor jede Seite ein waagerechter Strich gezogen. Die generelle Idee ist es, daß die Outputroutine von *Plain* TEX vor die Seite ein

```
\line{\the\headline}
```

[1] Man beachte den Unterschied zwischen Fußnoten und Fußzeilen. In letzteren stehen üblicherweise die Seitennummern o.ä.

und hinter die Seite ein

```
\line{\the\footline}
```

schreibt, wobei die Kopf- bzw. Fußzeile etwas von der eigentlichen Seite abgesetzt wird. Das Makro '`\nopagenumbers`' ist also einfach eine Abkürzung für

```
\footline={\hfil}
```

Normalerweise ist '`\footline={\hss\tenrm\folio\hss}`' definiert, wobei '`\folio`' die Seitennummer ausgibt. Die Umschaltung auf die Schrift ist notwendig, da sonst immer die aktuelle Schrift verwendet würde, was nicht immer wünschenswert ist. Das Makro leistet aber noch mehr. Wenn die Seitenzahl negativ ist, wird sie in römischen Ziffern ausgegeben, ist sie positiv, dann wird sie in normalen lateinischen Ziffern ausgegeben. Die Definition lautet:

```
\ifnum\pageno<0 \romannumeral-\pageno \else\number\pageno \fi
```

Hier nun mal ein Beispiel einer Seitennummerierung aus dem TEXbook, bei dem die Seitennummern in der Kopfzeile ausgegeben werden, und zwar bei ungeraden und geraden Seiten unterschiedlich.

```
\nopagenumbers % suppress footlines
\headline={\ifodd\pageno\rightheadline \else\leftheadline\fi}
\def\rightheadline{\tenrm\hfil RIGHT RUNNING HEAD\hfil\folio}
\def\leftheadline{\tenrm\folio\hfil LEFT RUNNING HEAD\hfil}
\voffset=2\baselineskip
```

Mit "Running Head"[2] werden in englischen Büchern Kopfzeilen bezeichnet, die auf mehreren Seiten erscheinen. Die Angabe von '`\voffset`' ist nötig, damit auch weiterhin ein Abstand zur eigentlichen Seiten bestehen bleibt.

Wenn sie '`\vsize`' nicht ändern, dann wird sich auch die Position der Kopf- bzw. Fußzeilen nicht ändern. Wenn sie z.B. den Befehl '`\raggedbottom`' verwenden, so daß nicht mehr alle Seiten die gleiche Höhe haben, dann ändert sich dennoch die Position der Fußzeile nicht!

11.2 Die Interna der Outputroutinen

Beim Aufruf einer Ausgaberoutine ist in der Box255 immer die aktuelle, zur Ausgabe bereite Seite gespeichert. Der Befehl '`\output`' wird wie ein normaler Befehl, so wie '`\everypar`' o.ä. bearbeitet, mit der Ausnahme, daß immer eine Gruppierungsklammer um den Befehl geschrieben wird. Dies dient der Vermeidung von Seiteneffekten. Die Ausgaberoutinen ändern z.B. oft den Baselineskip, wobei die Änderung natürlich nicht in den weiteren Text fließen soll.

Wenn keine spezielle Outputroutine installiert wird ('`\output={...}`'), dann verwendet TEX seine eigene, minimale Ausgaberoutine

```
\output={\shipout\box255}
```

Der Befehl '`\shipout`' schreibt das Argument in den derzeit gültigen Dvi-File, zeigt den Inhalt der Zahlregister 0 bis 9 an und führt die anstehenden '`\write`'-Befehle aus. Genauere Informationen können auch hier wieder mit dem Befehl '`\tracingoutput`' erhalten werden, wenn der Wert dieser Variablen größer als Null ist. Achten sie darauf, daß alle Makros während eines Shipout genau definiert sind, da sie zu der dann gültigen Version expandiert werden.

Neben dem 255ten Boxregister können auch noch andere Registerwerte Einfluß auf die Ausgaberoutine haben und von dieser ausgewertet werden. TEX verwendet z.B. die Werte

[2] Laufender Kopf.

```
\insertpenalties
\outputpenalty
```

in folgender Weise. Der erste Wert enthält die Anzahl der Einfügungen, die noch zurückgehalten werden, der zweite Wert ist gleich den Strafpunkten dieses Seitenumbruchs. Die Ausgaberoutine von *Plain* TEX erkennt z.B. ein '`\supereject`' um alle noch ausstehenden Einfügungen an den Dvi-File auszugeben. Dies geschieht, indem der zweite Wert auf -20000 gesetzt wird und der zweite Wert ausgewertet wird um zu entscheiden, wieviele Einfügungen noch vorhanden sind.

Die voreingestellte Ausgaberoutine von *Plain* TEX, die einfach alles ausgibt hat natürlich auch ein Pendant, eine Ausgaberoutine, die nichts ausgibt. Diese ist definiert als

```
\output={\unvbox255 \penalty=\outputpenalty}
```

Mit dem ersten Befehl wird die Seite wieder zurückgegeben, mit dem zweiten wieder die richtige Anzahl an Strafpunkten gesetzt. Jetzt versucht TEX wieder einen Seitenumbruch zu finden. Da sich nichts geändert hat, wird auch wieder derselbe Seitenumbruch gefunden, allerdings viel schneller, als beim ersten Mal, da z.B. die Paragraphen nicht umgebrochen werden brauchen. Somit befindet sich TEX in einer Endlosschleife.

Um derartige Endlosschleifen zu verhindern gibt es in TEX einen Mechanismus. Bei jedem Aufruf einer Ausgaberoutine wird der Zahler '`\deadcycles`' um eins erhöht und erst bei einer tatsächlichen Ausgabe wieder auf Null zurückgestellt. Wenn der Wert dieser Variablen gleich oder größer als '`\maxdeadcycles`' wird, dann wird die voreingestellte Ausgaberoutine aufgerufen.

Nach dem Aufruf einer Ausgaberoutine muß die Box255 leer sein, da TEX dort die neue Seite aufbauen will. Sollte das Register nicht leer sein, dann geht TEX davon aus, daß sie das Register fehlerhaft benutzen und zerstört den Inhalt.

11.3 Beispiele für Outputroutinen

11.3.1 Die Routine von *Plain* TEX

Die Ausgaberoutine von *Plain* TEX ist definiert als

```
\output={\plainoutput}
```

hierbei ist '`\plainoutput`' eine Abkürzung für

```
\shipout\vbox{\makeheadline
  \pagebody
  \makefootline}
\advancepageno
\ifnum\outputpenalty>-20000 \else\dosupereject\fi
```

Wie im TEXbook sollte dieses "Programm" Zeile für Zeile betrachtet werden, um einen Einblick in die Wirkungsweise dieser Ausgaberoutine zu gewinnen.

Zunächst das Makro '`\makeheadline`'. Dieses Makro erzeugt eine Box der Höhe und Tiefe Null, das die Kopfzeile enthält. Definiert ist es folgendermaßen

```
\vbox to 0pt{\vskip-22.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}
\nointerlineskip
```

Der seltsame Wert von -22.5pt sorgt dafür, daß die Referenzlinie der Kopfzeile genau 24pt oberhalb der Referenzlinie der obersten Textzeile der Seite liegt. Sie setzt sich zusammen aus

$$\text{topskip} - \text{Höhe eines Strut} - 2\text{Baselineskip}$$

das ist

$$10\text{pt} - 8.5\text{pt} - 24\text{pt}$$

Das Makro '`\pagebody`' ist eine Abkürzung für

```
\vbox to\vsize{\boxmaxdepth=\maxdepth \pagecontents}
```

Der Wert von '`\boxmaxdepth`' wird auf '`\maxdepth`' gesetzt, wobei die Routine annimmt, daß TEX die Box255 für die Seite verwendet hat. Das Makro '`\pagecontents`' fügt nun die Seite, inkl. aller Einfügungen am Anfang und am Ende ein. Es ist definiert als

```
\ifvoid\topins \else\unvbox\topins\fi
\dimen0=\dp255 \unvbox255
\ifvoid\footins\else % footnote info is present
  \vskip\skip\footins
  \footnoterule
  \unvbox\footins\fi
\ifraggedbottom \kern-\dimen0 \vfil \fi
```

'`\topins`' und '`\footins`' sind die Klassennummern für die Kopf- und Fußeinfügungen. Sollten auch noch andere Einfügungsklassen verwendet werden, ist diese Routine entsprechend abzuändern. Mit '`\footnoterule`' wird ein waagerechter Strich zur Abgrenzung des Textes von den Fußnoten ausgegeben. In der letzten Zeile wird zusätzlicher Leim eingefügt, falls die Seiten unterschiedliche Länge haben sollen, um den '`\topskip`' nicht überzustrapazieren.

Im "Hauptprogramm" folgt nun der Aufruf von '`\makefootline`'. Dieser Aufruf setzt die Fußzeile der Seite

```
\baselineskip=24pt
\line{\the\footline}
```

Das Makro '`\advancepageno`' erhöht nun die Seitenzahl, falls diese positiv ist, anderenfalls wird sie erniedrigt.[3] Definiert ist dieses Makro als

```
\ifnum\pageno<0 \global\advance\pageno by-1
\else \global\advance\pageno by 1 \fi
```

Schließlich werden *alle* Einfügungen ausgegeben, wenn nur die Strafpunkte den entsprechenden Wert haben. Die Definition sieht folgendermaßen aus

```
\ifnum\insertpenalties>0
  \line{} \kern-\topskip \nobreak
  \vfill\supereject\fi
```

Der Kernbefehl löscht einen evtl. vorhandenen Topskip, der zwischen den Einfügungen nur störend wäre.

11.3.2 Zweispaltige Ausgabe

Der Trick bei einer zweispaltigen Ausgabe besteht darin, daß man TEX einfach sagt, daß die Seite nur noch halb so breit sein darf, und dann die Ausgaberoutine nur jede zweite Seite etwas ausgeben läßt. Insofern stellt also jede einzelne Spalte eine eigenständige Seite mit eigenene Einfügungen und allen anderen Einzelheiten dar. Nur die Kopf- und Fußzeile teilt sie mit der Nachbarspalte.

Die Spalten sollen z.B. 3.2 Inch breit sein, der Zwischenraum soll 0.1 Inch betragen. Für die Kopf- und Fußzeile brauchen wir nun eine neue Größe

[3] Daher erfolgt die Nummerierung der *negativen* Seiten richtig, also: i, ii, iii, iv, ...

```
\newdimen\fullhsize
\fullhsize=6.5in \hsize=3.2in
\def\fullline{\hbox to\fullhsize}
```

Die Makros '`\makeheadline`' und '`\makefootline`' sollten nun '`\fullline`' statt '`\line`' benutzen.

Die Ausgaberoutine unterscheidet nun einen neuen Kontrollwert '`\lr`', der die Werte 'L' oder 'R' annehmen kann. Bekommt die Routine eine *linke* Spalte, dann wird diese einfach in einem Register gespeichert, bekommt die Routine eine rechte Spalte, dann wird sie zusammen mit der linken ausgegeben und die Seitennummer erhöht. Die Definition sieht folgendermaßen aus

```
\let\lr=L \newbox\leftcolumn
\output={\if L\lr
    \global\setbox\leftcolumn=\columnbox \global\let\lr=R
  \else \doubleformat \global\let\lr=L\fi
  \ifnum\outputpenalty>-20000 \else\dosupereject\fi}
\def\doubleformat{\shipout\vbox{\makeheadline
    \fullline{\box\leftcolumn\hfil\columnbox}
    \makefootline}
  \advancepageno}
\def\columnbox{\leftline{\pagebody}}
```

Das '`\columnbox`' Makro stellt sicher, daß die Seite auch in eine Spalte paßt. Es könnte ja sein, daß im Originaltext '`\hsize`' geändert wurde. Es kann nun passieren, daß ein Text mit einer gefüllten linken, aber leeren rechten Spalte endet. Dieser Text würde bei der obigen Routine nicht ausgegeben. Hier hilft die Anweisung

```
\supereject
\if R\lr \null\vfill\eject\fi
```

11.3.3 Markierungen

Bei der Erstellung eines Textes gibt es die unterschiedlichsten Anforderungen, die eine Kopfzeile erfüllen muß. Denken sie alleine einmal an den Unterschied zwischen einem Buch, in dessen Kopfzeile die aktuelle Kapitelüberschrift steht, gegenüber einem Wörterbuch, bei dem auf der linken Seite die ersten Buchstaben des ersten Eintrags der Seite stehen sollen, auf der rechten Seite aber die Anfangsbuchstaben des letzten Eintrags der Seite. Die besondere Art von TeX Seiten an eine Ausgaberoutine zu übergeben macht es nahezu unmöglich im Vorhinein festzustellen, an welcher Stelle TeX sich befinden wird, wenn es eine Seite umbricht.

TeX bietet nun die Möglichkeit innerhalb des Textes *Markierungen* anzubringen. Generell funktioniert das so, daß sie innerhalb eines Textes

```
\mark{<Markierungstext>}
```

schreiben, wobei der '`<Markierungstext>`' expandiert wird, wie bei einer '`\edef`' Definition. TeX schreibt diese Markierung in die vertikale Liste, wie schon früher erwähnt.

Früher haben wir uns auch schon vorgestellt, daß der Text für TeX eine lange Liste von Token ist. Stellen sie sich nun weiter vor diese lange Liste wird in Seiten gebrochen und an eine Ausgaberoutine übergeben. Dann werden vorher drei Register gesetzt

1. `\botmark` Ist der Markierungstext, der am weitesten unten auf der gerade bearbeiteten Seite gefunden wurde.

2. `\firstmark` Ist der Markierungstext, der am weitesten oben auf der gerade bearbeiteten Seite gefunden wurde.

3. `\topmark` Hat den Inhalt von `\botmark`, *bevor* die aktuelle Seite angefangen wurde.

Angenommen sie haben einen Text geschrieben mit vier Markierungen und der Text wird so gebrochen, daß Markierung α auf Seite 2, Markierung β und γ auf Seite 4 und Markierung δ auf Seite 5 steht. Dann sie die Register im Verlaufe der Abarbeitung folgendermaßen gesetzt.

Auf Seite	`\topmark` ist	`\firstmark` ist	`\botmark` ist
1	leer	leer	leer
2	leer	α	α
3	α	α	α
4	α	β	γ
5	γ	δ	δ
6	δ	δ	δ

Im vertikalen Modus werden Markierungen einfach an der Stelle, an der sie auftreten eingefügt, im horizontalen Modus werden sie an das Ende des Paragraphen angehängt. Im internen vertikalen Modus und im beschränkten horizontalen Modus können Markierungen verschwinden und sollten deshalb dort nicht verwendet werden. Der '`\vsplit`' Befehl behandelt die Markierungen übrigens entsprechend, allerdings mit einem Unterschied. Die obere bzw. untere Markierung wird nun in

```
\splitfirstmark
\splitbotmark
```

gefunden.

Eine Outputroutine kann nun die Markierungen nutzen um die entsprechend gewünschte Kopfzeile zu erstellen. Bei Wörterbüchern genügt die Verwendung von '`\firstmark`' bzw. '`\botmark`' völlig, um den Anforderungen genüge zu tun. Anders sieht das bei Büchern aus, die andere Kopfzeilen erfordern. Hier ist es meist erwünscht, daß in der Kopfzeile *das* Kapitel benannt werden soll, welches zu Beginn einer Seite gültig ist. Sollte auf der Seite ein neues Kapitel beginnen, dann soll dessen Kopfzeile erst auf der nächsten Seite stehen.

Angenommen in einem Buch beginnt auf einer Seite ein neuer Abschnitt, z.B. der 'Abschnitt 3. Neues aus TEXLand'. Dieser Abschnitt soll mit dem Befehl

```
\beginsection 3. Neues aus \TeX Land
```

eingeleitet werden. Wie ist dann der Befehl '`\beginsection`' am besten zu definieren? Hier eine Möglichkeit

```
\def\beginsection #1. #2.
  {\sectionbreak
  \leftline{\sectionfont #1. #2}
  \mark{#1}
  \nobreak\smallskip\noindent}
```

Der '`\sectionbreak`' Befehl soll dafür sorgen, daß entweder ein genügend großer Zwischenraum zum vorherigen Abschnitt gelassen wird, oder daß die letzte Seite aufgefüllt wird und eine neue Seite begonnen wird. Eine mögliche Realisation wäre

```
\penalty-200 \vskip18pt plus4pt minus6pt
```

Der letzte Befehl sorgt dafür, daß der erste Paragraph des Textes nicht eingerückt wird, uns interessiert hier aber die Markierung. Die hier vorgestellte Lösung erfüllt aber nicht die Erwartungen. Das Ergebnis ist nämlich folgendermaßen. '`\firstmark`' wird '3.' sein und '`\topmark`' '2.' unabhängig davon, ob der Absatz auf der Seite beginnt oder nicht. Die Lösung bestünde darin die Markierung *vor* den Befehl zu Beginn des Absatzes zu setzen ('`\sectionbreak`'), statt an die Stelle, an der er jetzt steht. Dann würde '`\topmark`' immer den Absatz bezeichnen, der an Beginn der Seite noch gültig ist.

Es gibt aber noch eine bessere Lösung. Diese sorgt dafür, daß sich '`\topmark`' immer auf den Anfang der Seite, und '`\botmark`' immer auf das Ende der Seite bezieht. Damit ist eine noch genauere Kopfgestaltung, z.B. bezogen auf rechte und linke Seiten möglich. Eine Definition könnte folgendermaßen aussehen

```
\def\beginsection #1. #2.
  {\mark{\currentsection \noexpand\else #1}
  \sectionbreak
  \leftline{\sectionfont #1. #2}
  \mark{#1\noexpand\else #1} \def\currentsection{#1}
  \nobreak\smallskip\noindent}
\def\currentsection{} % the current section number
```

Die Idee hier ist folgende: Es werden zwei Markierungen gesetzt, eine vor und eine hinter dem Absatzabstand. Außerdem bestehen die Markierungen aus dem Text '`2.\else 3.`' für '`\topmark`' und '`3.\else 3.`' für '`\botmark`'. Die linke Komponente von '`\botmark`' ist somit für den unteren Seitenrand, die rechte Komponente von '`\topmark`' für den oberen Seitenrand zuständig.

Im obigen Beispiel wurde zwischen zwei Markierungskomponenten mittels einer '`if`' Abfrage ausgewählt. Völlig analog kann natürlich auch ein Markierungstext aus

```
a1 \or a2 \or a3 \or ...
```

bestehen und dann aus mehreren Alternativen ausgewählt werden.

Kapitel 12

Umgang mit Fehlern

In einem früheren Kapitel haben sie schon gesehen, wie TEX darauf reagiert, wenn eine unbekannte Kontrollsequenz auftaucht, meist ein Fehler in der Schreibweise. TEX kennt aber wesentlich mehr Fehler, die sie aber vermutlich nie alle kennenlernen werden, da manche dieser Fehler nur schwer zu produzieren sind. In diesem Kapitel werden nun einige weitere Fehlersituationen behandelt.

12.1 Schreibfehler

Normalerweise treten Fehler als Schreibfehler auf. Was passiert z.B. wenn man eine Maßeinheit falsch schreibt? Gehen wir man davon aus, daß sie statt '`\hsize=4in`' '`\hsize=4im`' geschrieben hätten. Die Fehlermeldung, die sie erhalten wird dann ungefähr so aussehen

```
! Illegal unit of measure (pt inserted).
<to be read again>
                   i
<to be read again>
                   m
<*> \hsize=4im
               \input story
?
```

Sie sehen, ich verwende wieder das Beispiel von früher. Ein einfaches Return würde hier dafür sorgen, daß die Buchstaben 'i' und 'm' einen neuen Paragraphen ausmachen würden, besser wäre es eine '2' einzugeben, um die Buchstaben zu löschen. Danach würde TEX anhalten um ihnen die Gelegenheit zu geben das weitere Vorgehen zu überdenken. Die zugehörige Meldung sähe folgendermaßen aus

```
<recently read> m
|indent
<*> \hsize=4im
               \input story
?
```

O.k. das 'i' und das 'm' sind nun weg. Würden sie jetzt versuchen TEX mit einem einfachen Return zur Weiterarbeit zu veranlassen, dann würde TEX versuchen den ganzen Text in einer Breite von 4pt zu setzen. Die Anzahl von Fehlermeldungen wg. übervollen `hbox`en können sie sich sicher selber vorstellen. Besser ist es die gewünschte Breite erneut einzufügen.

```
I\hsize=4in
```

Normalerweise versucht TEX selber Fehler zu beseitigen. Das geschieht dadurch, daß Texte eingefügt werden, die offensichtlich ausgelassen wurden, oder aber Textteile entfernt werden, die offensichtliche fehlen. Auch hierzu ein Beispiel

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   ^
l.11 Die Tatsache 32768=2^
                           {15} ist hier uninteressant
? H
I've inserted a begin-math/end-math symbol since I think
you left one out. Proceed, with fingers crossed.
```

Der Unterschied zum obigen Beispiel liegt darin, daß hier der eingefügte Text ('$') noch nicht wirklich eingefügt wurde. Sie haben also die Chance den Einfügungstext zu entfernen, bevor TEX ihn tatsächlich zu sehen bekommt.

Wie sollte man in einem solchen Falle vorgehen. Die Sequenz '32768=2' ist schon gesetzt, und sie haben keine Chance den falschen Leerraum um das Gleichheitszeichen nachträglich zu ändern. Ihr Ziel sollte sein den Rest des Textes möglichst fehlerfrei bearbeitet zu bekommen, um eventuell auftretende weitere Fehler noch gemeldet zu bekommen.

Ein Return würde den Rest des Paragraphen in Italic-Schrift setzen, was mit Sicherheit nicht wünschenswert ist. Es gibt zwei Alternativen zu reagieren

1. Mit '6' löschen sie das '`$^{15}`' und fügen dann mit '`I$^{15}$`' den richtigen Text ein, oder

2. sie geben '2' ein und erhalten $\ldots 2^{15}$, und der Rest des Absatzes wird richtig gesetzt.

12.2 Schlimme Fehler

Richtig ärgerlich sind allerdings die Fehler, die z.B. in Zeile 10 eines Absatzes verursacht werden, aber erst in Zeile 73 desselben Absatzes bemerkt werden.[1] Derartige Fehler haben meist Folgefehler, dennoch ist es meist möglich in einem Lauf von TEX alle Fehler zu finden, wenn man nur auf die Fehlermeldungen entsprechend reagiert. Am schlimmsten sind aber Fehler, bei denen kein Schreibfehler vorliegt, sondern ein Bedeutungsfehler. Wenn sie z.B. mit

```
\def\box{...}
```

eine Definition einrichten und so alle Boxbefehle von TEX unsinnig werden lassen, dann wird es schwer diesen Fehler zu beheben. Am besten lesen sie dieses Buch von Anfang bis Ende und vermeiden solche Fehler.

Auch bei der Makrodefinition können solche Fehler auftreten. Das TEXBook stellt einen solchen Fehler dar. Die Definition lautet

```
\newcount\serialnumber
\def\firstnumber{\serialnumber=0 }
\def\nextnumber{\advance \serialnumber by 1
  \number\serialnumber)\nobreak\hskip.2em }
```

Mit dem Aufruf von

```
\firstnumber
\nextnumber xx, \nextnumber yy, and \nextnumber zz
```

würde TEX dann folgendes setzen: 1) xx, 2) yy und 3) zz. Ein anderer Benutzer könnte nun folgende Fehlermeldung erhalten

[1] Meistens handelt es sich um eine fehlende Gruppenklammer, es ist also ein Editor empfehlenswert, der zueinander passende Klammerpaare findet.

```
! Missing number, treated as zero.
<to be read again>
                     c
l.107 \nextnumber minusc
                             ule chances of error
?
```

Der Fehler läge darin, daß das Wort '`minus`' ein Schlüsselwort von TEX ist und natürlich eine Zahl erwartet würde. Der Fehler kann einfach dadurch behoben werden, daß die Makrodefinition mit einem '`\relax`' abgeschlossen wird.

12.3 Fehler, die nicht zur Unterbrechung führen

Auch wenn ihr Text vollständig und ohne Fehlermeldung übersetzt wird, kann es noch zu Fehlern kommen. Eine Möglichkeit ist die, daß ein von ihnen verwendeter Zeichensatz nicht zur Verfügung steht und von ihrem Dvi-Treiber durch einen anderen ersetzt wird. Dieser Ersatzzeichensatz kann allerdings ziemlich schlecht aussehen und ihren Ansprüchen nicht genügen. In diesem Falle müssen sie schon per Hand zur Fehlerkorrektur greifen, da kann ihnen TEX nicht weiterhelfen.[2]

12.4 Der Uralttrick

Wenn sie wirklich einmal nicht mehr weiter wissen und absolut nicht darauf kommen, wieso TEX an einer Stelle einen Fehler macht, dann gehen sie folgendermaßen vor. Kürzen sie den Text solange, bis er praktisch nur noch aus dem Fehler besteht. Sie haben eine wesentliche bessere Chance den Fehler zu finden.

Sehr beliebt sind in diesem Zusammenhang auch fehlende Leerzeichen. Denken sie daran, daß TEX in zwei Fällen Leereichen als Trenner ließt und nicht ausgibt

1. Hinter Kontrollsequenzen, und

2. hinter Maßangaben für TEX Primitive.

Die Suche nach derartigen Fehlern gestaltet sich besonders bei der Verwendung von Makros besonders schwierig. Setzen sie ggf.

```
\tracingcommands=1
```

um hinter das Problem zu kommen.

12.5 Fatale Fehler

In manchen Fällen weiß auch das gute, alte TEX nicht mehr weiter. Z.B. im '`\batchmode`' wenn TEX eine Eingabe von der Tastatur benötigt. TEX macht dann einen Nothalt, bevor es für immer den Geist aufgibt. Hier eine Auswahl der Fehlermeldungen, die sie u.U. zu sehen bekommen.

```
Fatal format file error; I'm stymied.
```

Der geladene Formatfile kann nicht benutzt werden, da er für eine andere Version von TEX hergestellt wurde.

```
That makes 100 errors; please try again.
```

TEX hat hundert Fehler gefunden, seit der letzte Paragraph beendet wurde. Vermutlich befindet es sich in einer Endlosschleife.

---

[2] Am besten besorgen sie sich den benötigten Zeichensatz.

`Interwoven alignment preambles are not allowed.`

Wenn sie diese Meldung erhalten, dann werden sie auch verstehen, was sie bedeutet, und sie werden sie nicht lieben.

`I can't go on meeting you like this.`

Wegen eines früheren Fehlers kann TEX nicht weiter arbeiten, sie müssen erst den vorigen Fehler beheben.

`This can't happen.`

Irgendetwas ist mit ihrem TEX nicht in Ordnung.

12.6 Speicherprobleme

Mitunter kann auch folgende Meldung auftreten

`TeX capacity exceeded, sorry.`

Dies ist immer dann der Fall, wenn für irgendein Teil von TEX mehr Speicher benötigt wird, als auf ihrem Rechner vorhanden ist. Zusätzlich wird auch angegeben, um welche Art von Speicher es sich handelt. Es gibt vierzehn Möglichkeiten:

Meldung	*Gegenstand der Bearbeitung*
`number of strings`	Namen der Kontrollsequenzen und der Files
`pool size`	Die Zeichen in derartigen Namen
`main memory size`	Boxen, Leim, Umbruchpunkte, Tokenlisten, Zeichen, etc.
`hash size`	Namen der Kontrollsequenzen
`font memory`	Zeichensatzdaten
`exception dictionary`	Trennungsausnahmen
`input stack size`	Parallele Eingabedateien
`semantic nest size`	Bearbeitung einer unfertigen Liste
`parameter stack size`	Makroparameter
`buffer size`	Zeichen in einer Eingabezeile
`save size`	Werte, die nach einem Gruppenende wieder hergestellt werden müssen
`text input levels`	Inputfiles und Einfügungen
`grouping levels`	Unvollständige Gruppen
`pattern memory`	Trennungsmuster

Mit '`\tracingstats=1`' können natürlich wieder alle Speicherverwendungen angesehen werden. Bei jedem Shipout wird dann der Speicherbedarf angegeben, und zwar in der Form 'xxx&xxxx'. Dabei meint die Zahl vor dem Kaufmannsund wieviele (Speicher)worte für große Dinge, wie Boxen etc. verwendet wurden, und die zweite Zahl wieviele für kleine Dinge, wie Zeichen oder Token verwendet wurden.

Was macht man nun, wenn ein derartiger Fehler aufgetreten ist, und man sonst keine Möglichkeit hat für mehr Speicher zu sorgen. Es gibt zwei Möglichkeiten. Zum einen können sie nach großen Paragraphen, Tabellen oder Makrodefinitionen suchen und diese verkleinern.[3] Sie können aber auch TEX anders konfigurieren. Sie müssten dann für die Speicherart, bei der das Problem auftrat, mehr physikalischen Speicher bereitstellen, den sie bei anderen Arten, die evtl. nicht so ausgelastet sind, einschränken.

Meist handelt es sich aber um einen Fehler in einer Definition, der leicht zu beheben ist. Bei der Definition von

[3]Tricks dazu stehen in diesem Text ja reichlich.

```
\def\rekurs{(\rekurs)}
```

erhalten sie in kürzester Zeit die Fehlermeldung

```
! TeX capacity exceeded, sorry [input stack size=80].
\recurse ->(\recurse
                    )
\recurse ->(\recurse
                    )
...
```

was nicht weiter verwundern dürfte.

12.7 Eine letzte Hilfe

Wenn ihnen sonst nichts mehr einfällt, um einen bestimmten Fehler zu beheben, dann können sie immer noch ein '`\pausing=1`' einfügen, was TEX dazu veranlaßt die Zeilen von der Tastatur und nicht mehr vom Eingabefile einzulesen. Sie können dann einen der vielen '`\show...`' Befehle einfügen, oder sonst Befehle eingeben, die ihnen vielleicht weiterhelfen.

Einen kleinen Wehrmutstropfen zum Schluß. Die ganzen Tracingbefehle sind leider nicht auf allen TEX Implementationen verfügbar.

Anhang A

Die Plain TEX Befehle

A.1 Die Konventionen

A.1.1 Spezielle Buchstaben

Für besonderen Gebrauch sind die Zeichen

```
\ { } $ & # % ^ _ ~
```

bestimmt.

A.1.2 Die Schriftarten

\rm roman	{\sl geneigt}	{\bf fett}	{\it Italic\/} Schrift
roman	*geneigt*	**fett**	*Italic* Schrift

A.1.3 Sonderzeichen

``	''	--	---	?`	!`	\$	\#	\&	\%
“	”	–	—	¿	¡	$	#	&	%
\ae	\AE	\oe	\OE	\aa	\AA	\ss	\o	\O	\`a
æ	Æ	œ	Œ	å	Å	ß	ø	Ø	à
\'e	\^o	\\"u	\=y	\~n	\.p	\u\i	\v s	\H\j	\t\i u
é	ô								
ü	ȳ	ñ	ṗ	ĭ	š	ȷ̋	i͡u		
\b k	\c c	\d h	\l	\L	\dag	\ddag	\S	\P	
ḵ	ç	ḥ	ł	Ł	†	‡	§	¶	

Und außerdem noch die Zeichen

{\it\$	\&}	\copyright	\TeX	\dots
$	*&*	©	TEX	. . .

A.1.4 Zeilenumbruchbefehle

Die *normalen* Befehle

```
\break \nobreak \allowbreak \hbox{Nicht umbrechbar}
```

und die speziellen Befehle

```
dis\-cre\-tion\-ary Tren\-nung
Erlaubte\slash Trennung
```

A.1.5 Horizontale Abstände

Mit erlaubtem Zeilenumbruch

`\␣`	Normaler Wortabstand
`\enskip`	So viel
`\quad`	So viel
`\qquad`	So viel
`\hskip`	Angebbarer Platz

Ohne Erlaubnis zum Zeilenumbruch

`~`	Normaler Wortabstand
`\enspace`	So viel
`\thinspace`	So viel
`\negthinspace`	Soviel
`\kern`	Angebbarer Platz

A.1.6 Vertikale Abstände

`\smallskip` `\medskip` `\bigskip`

A.1.7 Seitenumbruch

Seitenumbruchkontrolle ohne vertikalen Platz erlauben die Befehle

`\eject` `\supereject` `\nobreak` `\goodbreak` `\filbreak`

Mit vertikalem Platz verwendet man

`\smallbreak` `\medbreak` `\bigbreak`

A.1.8 Tabellenbefehle

Zum einen gibt es die Tabellenbefehle

`\settabs` `\columns` `\+` `\cr`

und außerdem die etwas generelleren Befehle

`\halign` `\valign` `\omit` `\span` `\multispan`

Zum Auffüllen der Zwischenräume in Tabellen dienen die Befehle

`\hrulefill`	`\dotfill`
`\leftarrowfill`	`\rightarrowfill`
`\upbracefill`	`\downbracefill`

A.2 Die Makros von plain.tex

Die Makros aus dem File `plain.tex` sollen ab hier erklärt werden. Tritt dabei ein Befehl auf, der bisher noch nicht behandelt wurde, dann wird er auch für normale Anwender erklärt, ansonsten richtet sich die Beschreibung eher an die Designer von Makros.[1]

[1] Ich habe die Kommentare im Source in englischer Sprache gehalten, die Bedeutung sollte aber sowieso klar sein.

A.2.1 Die Kodetabellen

Zunächst müssen die Kategoriekodes mancher Zeichen geändert werden, damit überhaupt weitere Definitionen möglich sind. Die ersten Zeilen lauten daher

```
\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\$=3 % dollar sign is math shift
\catcode`\&=4 % ampersand is alignment tab
\catcode`\#=6 % hash mark is macro parameter character
\catcode`\^=7 \catcode`\^^K=7 % circumflex and uparrow for superscripts
\catcode`\_=8 \catcode`\^^A=8 % underline and downarrow for subscripts
\catcode`\^^I=10 % ASCII tab is treated as a blank space
\chardef\active=13 \catcode`\~=\active % tilde is active
\catcode`\^^L=\active \outer\def^^L{\par} % ASCII form-feed is \outer\par

\message{Preloading the plain format: codes,}
```

Die Zuweisungen für '`^^K`' und '`^^A`' dienen für die Verwendung von anderen Tastaturen. Alle Zeichen außer den Buchstaben erhalten die Kategorie *anders* (12), die Buchstaben die Kategorie 11. Weiterhin werden die folgenden Zeichenkodes gesetzt

```
\catcode `\\ =0   \catcode`\   =10   \catcode `\%=14
\catcode`\^^@=9   \catcode`\^^M=5   \catcode`\^^?=15
```

Der '`\message`' Befehl gibt eine Meldung über den Stand der Dinge bis zu diesem Zeitpunkt auf dem Bildschirm aus.[2] Wenn sie außer den Befehlen von `plain.tex` weitere Befehle einbinden wollen, dann schreiben sie nach der Aufforderung von INITEX einfach '`&plain meine`', wenn sie ihre Definitionen in einer Datei mit dem Namen `meine.tex` geschrieben haben. Die *Plain* TEX Befehle werden dann auf dem schnelleren, vorkompilierten Wege eingelesen und ihre Definitionen angefügt.

Schließlich wird das '`@`' Zeichen vorübergehend zu einem normalen Zeichen gemacht. Auf diese Art ist es möglich, daß *Plain* TEX seine eigenen Befehle kennt, die sie als Anwender nicht so einfach benutzen können. Später wird diesem Zeichen wieder eine andere Kategorie zugewiesen. Jetzt kommen die Tabellen der mathematischen Zeichen

```
\mathcode`\^^@=\"2201 \mathcode`\^^A=\"3223 \mathcode`\^^B=\"010B
\mathcode`\^^C=\"010C \mathcode`\^^D=\"225E \mathcode`\^^E=\"023A
\mathcode`\^^F=\"3232 \mathcode`\^^G=\"0119 \mathcode`\^^H=\"0115
\mathcode`\^^I=\"010D \mathcode`\^^J=\"010E \mathcode`\^^K=\"3222
\mathcode`\^^L=\"2206 \mathcode`\^^M=\"2208 \mathcode`\^^N=\"0231
\mathcode`\^^O=\"0140 \mathcode`\^^P=\"321A \mathcode`\^^Q=\"321B
\mathcode`\^^R=\"225C \mathcode`\^^S=\"225B \mathcode`\^^T=\"0238
\mathcode`\^^U=\"0239 \mathcode`\^^V=\"220A \mathcode`\^^W=\"3224
\mathcode`\^^X=\"3220 \mathcode`\^^Y=\"3221 \mathcode`\^^Z=\"8000
\mathcode`\^^[=\"2205 \mathcode`\^^\=\"3214 \mathcode`\^^]=\"3215
\mathcode`\^^^=\"3211 \mathcode`\^^_=\"225F \mathcode`\^^?=\"1273
\mathcode`\ =\"8000   \mathcode`\!=\"5021   \mathcode`\'=\"8000
\mathcode`\(=\"4028   \mathcode`\)=\"5029   \mathcode`\*=\"2203
\mathcode`\+=\"202B   \mathcode`\,=\"613B   \mathcode`\-=\"2200
\mathcode`\.=\"013A   \mathcode`\/=\"013D   \mathcode`\:=\"303A
\mathcode`\;=\"603B   \mathcode`\<=\"313C   \mathcode`\==\"303D
\mathcode`\>=\"313E   \mathcode`\?=\"503F   \mathcode`\[=\"405B
\mathcode`\\=\"026E   \mathcode`\]=\"505D   \mathcode`\_=\"8000
\mathcode`\{=\"4266   \mathcode`\|=\"026A   \mathcode`\}=\"5267
```

[2] Diese Meldungen erscheinen *immer* bei der Bearbeitung mit INITEX.

Es folgen die Setzungen für

```
\uccode  \lccode
\sfcode  \delcode
```

Abschließend noch einige Definitionen

```
\chardef\@ne=1 \chardef\tw@=2 \chardef\thr@@=3
\chardef\sixt@@n=16 \chardef\@cclv=255
\mathchardef\@cclvi=256 \mathchardef\@m=1000
\mathchardef\@M=10000 \mathchardef\@MM=20000
```

Diese Definitionen sorgen dafür, daß TEX bei manchen Anwendungen schneller läuft und weniger Speicher benötigt. In diesem Anhang werden aber dennoch immer die Zahlkonstanten verwendet, da so die Programme besser lesbar sind.

A.2.2 Die Register

Der zweite Teil von `plain.tex` beschäftigt sich mit der Alloziierung von Registern, so daß später die einzelnen Makros miteinander arbeiten können, ohne sich gegenseitig die Register umzubelegen. Hier die zugehörigen Regeln

1. Die Register mit den Nummern 0 bis 9 sind immer für den temporären Gebrauch frei. Jedes Makro beendet seine Benutzung eines dieser Register nach seiner Benutzung.[3]
2. Die Register `\count255`, `\dimen255` und `\skip255` stehen auch nur für den temporären Gebrauch zur Verfügung.
3. Alle Zuweisungen zu ungeraden Registern sollten lokal, alle zu geraden Registern sollten global sein.[4]
4. Register dürfen natürlich innerhalb einer Gruppe frei belegt werden, wenn sichergestellt ist, daß sie am Ende der Gruppe wieder hergestellt werden können und wenn keine anderen Makros globale Zuweisungen auf dieses Register vornehmen.
5. Register, die von vielen Makros, oder für eine längere Zeit beansprucht werden, sollten mit den `\new...` Befehle alloziiert werden.

Einige Abkürzungen dienen dem besseren Zugriff auf die Temporalregister

```
\countdef\count@=255  \toksdef\toks@=0       \skipdef\skip@=0
\dimendef\dimen@=0    \dimendef\dimen@i=1    \dimendef\dimen@ii=2
```

Es folgen nun einige Zuweisungen zu Registern, mit denen später festgehalten wird, welche Zahl denn nun mit einem `\new` Befehl alloziiert wurde. Diese Zahl kann auch von einem alloziierenden Makro verwendet werden.

```
\count10=22 % this counter allocates \count registers 23, 24, 25, ...
\count11=9  % this counter allocates \dimen registers 10, 11, 12, ...
\count12=9  % this counter allocates \skip registers 10, 11, 12, ...
\count13=9  % this counter allocates \muskip registers 10, 11, 12, ...
\count14=9  % this counter allocates \box registers 10, 11, 12, ...
\count15=9  % this counter allocates \toks registers 10, 11, 12, ...
\count16=-1 % this counter allocates input streams 0, 1, 2, ...
```

[3] Die Zählregister `\count0` bis `\count9` sind allerdings schon von TEX vorbelegt. Diese Reservierung gilt also nur für alle anderen Register.

[4] Damit verhindert man, daß zuviele Register am Ende einer Gruppe wieder hergestellt werden müssen.

```
\count17=-1 % this counter allocates output streams 0, 1, 2, ...
\count18=3  % this counter allocates math families 4, 5, 6, ...
\count19=0  % this counter allocates language codes 1, 2, 3, ...
\count20=255 % this counter allocates insertions 254, 253, 252, ...
\countdef\insc@unt=20 % nickname for the insertion counter
\countdef\allocationnumber=21 % the most recent allocation
\countdef\m@ne=22 \m@ne=-1 % a handy constant
\def\wlog{\immediate\write-1} % this will write on log file (only)

\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\outer\def\newskip{\alloc@2\skip\skipdef\insc@unt}
\outer\def\newmuskip{\alloc@3\muskip\muskipdef\@cclvi}
\outer\def\newbox{\alloc@4\box\chardef\insc@unt}
\let\newtoks=\relax % this allows plain.tex to be read in twice
\outer\def\newhelp#1#2{\newtoks#1#1=\expandafter{\csname#2\endcsname}}
\outer\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
\outer\def\newread{\alloc@6\read\chardef\sixt@@n}
\outer\def\newwrite{\alloc@7\write\chardef\sixt@@n}
\outer\def\newfam{\alloc@8\fam\chardef\sixt@@n}
\outer\def\newlanguage{\alloc@9\language\chardef\@cclvi}

\def\alloc@#1#2#3#4#5{\global\advance\count1#1 by 1
  \ch@ck#1#4#2% make sure there's still room
  \allocationnumber=\count1#1   \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}

\outer\def\newinsert#1{\global\advance\insc@unt by-1
  \ch@ck0\insc@unt\count \ch@ck1\insc@unt\dimen
  \ch@ck2\insc@unt\skip  \ch@ck4\insc@unt\box
  \allocationnumber=\insc@unt
  \global\chardef#1=\allocationnumber
  \wlog{\string#1=\string\insert\the\allocationnumber}}

\def\ch@ck#1#2#3{\ifnum\count1#1<#2%
  \else\errmessage{No room for a new #3}\fi}
```

Das Makro '`\alloc@`' erledigt hier die meiste Arbeit, inkl. der Meldung in den Logfile etc. Außerdem wurde ein Befehl '`\newhelp`' eingeführt, der es erlaubt relativ einfach eigene Hilfstexte zu bearbeiten. Mit der Folge

```
\newhelp\helpout{Hilfstext}
\errhelp=\helpout
```

vor dem Aufruf von '`\errmessage`' wird dieser Text verfügbar gemacht.[5]

Nun werden die wichtigen Konstanten alloziiert.

```
\newdimen\maxdimen \maxdimen=16383.99999pt
\newskip\hideskip \hideskip=-1000pt plus1fill
\newskip\centering \centering=0pt plus 1000pt minus 1000pt
\newdimen\p@ \p@=1pt % this saves macro space and time
\newdimen\z@ \z@=0pt % likewis
\newskip\z@skip \z@skip=0pt plus0pt minus0pt
\newbox\voidb@x % permanently void box register
```

[5] Dies dürfte aber wirklich *nur* Makrodesigner interessieren.

Die Kontrollsequenz '`\maxdimen`' steht für die größte zur Verfügung stehende Dimension. Die Befehle '`\hideskip`' und '`\centering`' sollten niemals geändert werden, von ihnen wird später noch zu lesen sein. Die konstanten Zuweisungen dienen wieder einmal der Optimierung.

Abschließend noch einige andere Alloziierungen

```
\outer\def\newif#1{\count@=\escapechar \escapechar=-1\parbreak%
  \expandafter\expandafter\expandafter\parbreak%
   \edef\@if#1{true}{\let\noexpand#1=\noexpand\iftrue}%\parbreak%
  \expandafter\expandafter\expandafter\parbreak%
   \edef\@if#1{false}{\let\noexpand#1=\noexpand\iffalse}%\parbreak%
  \@if#1{false}\escapechar=\count@} % the condition starts out false
\def\@if#1#2{\csname\expandafter\if@\string#1#2\endcsname}\parbreak%
{\uccode`1=`i \uccode`2=`f \uppercase{\gdef\if@12{}}} % `if' is required
```

A.2.3 Parameter

INITEX setzt die meisten Parameter auf Null, leert alle Box- und sonstigen Register, es gibt aber auch einige Ausnahmen

Parameter	*voreingestellter Wert*
`\mag`	1000
`\tolerance`	10000
`\maxdeadcycles`	25
`\hangafter`	1
`\escapechar`	`` `\\ ``
`\endlinechar`	`` `\^^M ``

Darüber hinaus setzt *Plain* TEX noch weitere Werte

```
\pretolerance=100 \tolerance=200 \hbadness=1000 \vbadness=1000
\linepenalty=10   \hyphenpenalty=50 \exhyphenpenalty=50
\binoppenalty=700 \relpenalty=500
\clubpenalty=150 \widowpenalty=150 \displaywidowpenalty=50
\brokenpenalty=100 \predisplaypenalty=10000
\doublehyphendemerits=10000 \finalhyphendemerits=5000 \adjdemerits=10000
\tracinglostchars=1 \uchyph=1 \delimiterfactor=901
\defaulthyphenchar=`\- \defaultskewchar=-1 \newlinechar=-1
\showboxbreadth=5 \showboxdepth=3 \errorcontextlines=5


\hfuzz=0.1pt \vfuzz=0.1pt \overfullrule=5pt
\hsize=6.5in \vsize=8.9in \parindent=20pt
\maxdepth=4pt \splitmaxdepth=\maxdimen \boxmaxdepth=\maxdimen
\delimitershortfall=5pt \nulldelimiterspace=1.2pt \scriptspace=0.5pt

\parskip=0pt plus 1pt
\abovedisplayskip=12pt plus 3pt minus 9pt
\abovedisplayshortskip=0pt plus 3pt
\belowdisplayskip=12pt plus 3pt minus 9pt
\belowdisplayshortskip=7pt plus 3pt minus 4pt
\topskip=10pt \splittopskip=10pt
\parfillskip=0pt plus 1fil

\thinmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
```

Einige Parameter werden erst zur Laufzeit initialisiert. Die Parameter

```
\time \day \month \year
```

werden erst zu Beginn einer Bearbeitung gesetzt. Der Parameter '`\outputpenalty`' wird von der Ausgaberoutine gesetzt. Die Parameter

```
\predisplaysize \displaywidth \displayindet
```

erhalten ihre Werte, unmittelbar bevor eine abgesetzte Formel erstellt wird, und die Parameter

```
\looseness=0
\hangindent=0pt
\hangafter=1
\parshape=0
```

werden am Ende jedes Paragraphen gesetzt, wenn TEX in den vertikalen Modus übergeht. Auch die Parameter

```
\baselineskip \lineskip \lineskiplimit
```

werden erst durch ein Makro initialisiert, das erst später vorgestellt wird.

Jetzt kommen einige "Pseudoparameter", sie verhalten sich genauso, wie die Parameter von TEX, gehören aber eher zum Sprachumfang von *Plain* TEX, als zu den Primitiven.

```
\newskip\smallskipamount % the amount of a \smallskip
  \smallskipamount=3pt plus1pt minus1pt
\newskip\medskipamount % the amount of a \medskip
  \medskipamount=6pt plus2pt minus2pt
\newskip\bigskipamount % the amount of a \bigskip
  \bigskipamount=12pt plus4pt minus4pt
\newskip\normalbaselineskip % normal value of \baselineskip
  \normalbaselineskip=12pt
\newskip\normallineskip % normal value of \lineskip
  \normallineskip=1pt
\newdimen\normallineskiplimit % normal value of \lineskiplimit
  \normallineskiplimit=0pt
\newdimen\jot % unit of measure for opening up displays
  \jot=3pt
\newcount\interdisplaylinepenalty % interline penalty in \displaylines
  \interdisplaylinepenalty=100
\newcount\interfootnotelinepenalty % interline penalty in footnotes
  \interfootnotelinepenalty=100
```

A.2.4 Fontinformationen

Hier kommen nun die Befehle, die TEX braucht um sich in den ganzen Zeichensätzen zurechtfinden zu können. Zunächst das `magstep` Makro

```
\def\magstephalf{1095 }
\def\magstep#1{\ifcase#1 1000\or
  1200\or 1440\or 1728\or 2074\or 2488\fi\relax}
```

Bemerkenswert ist vielleicht, daß eine Zahl zurück gegeben wird und nicht mit dem Multiplikationsbefehl gearbeitet wird. Der Grund dafür liegt darin, daß der Multiplikationsbefehl eine Zuweisung bedeutet, und Zuweisungen werden innerhalb von Makros nicht ausgeführt.

Jetzt kommen die Zeichensätze, die schon einmal vorab geladen werden sollen.

```
\font\tenrm=cmr10         \font\preloaded=cmr9      \font\preloaded=cmr8
\font\sevenrm=cmr7        \font\preloaded=cmr6      \font\fiverm=cmr5

\font\teni=cmmi10         \font\preloaded=cmmi9     \font\preloaded=cmmi8
\font\seveni=cmmi7        \font\preloaded=cmmi6     \font\fivei=cmmi5

\font\tensy=cmsy10        \font\preloaded=cmsy9     \font\preloaded=cmsy8
\font\sevensy=cmsy7       \font\preloaded=cmsy6     \font\fivesy=cmsy5

\font\tenex=cmex10

\font\tenbf=cmbx10        \font\preloaded=cmbx9     \font\preloaded=cmbx8
\font\sevenbf=cmbx7       \font\preloaded=cmbx6     \font\fivebf=cmbx5

\font\tensl=cmsl10        \font\preloaded=cmsl9     \font\preloaded=cmsl8
\font\tentt=cmtt10        \font\preloaded=cmtt9     \font\preloaded=cmtt8
\font\tenit=cmti10        \font\preloaded=cmti9     \font\preloaded=cmti8
\font\preloaded=cmss10    \font\preloaded=cmssq8
\font\preloaded=cmssi10   \font\preloaded=cmssqi8

\font\preloaded=cmr7 scaled \magstep4 % for titles
\font\preloaded=cmtt10 scaled \magstep2
\font\preloaded=cmssbx10 scaled \magstep2

% Additional \preloaded fonts can be specified here.
% (And those that were \preloaded above can be eliminated.)
\let\preloaded=\undefined % preloaded fonts must be declared anew later.
```

Mit dem '`\preloaded`' Befehl werden die Zeichensätze nicht *tatsaechlich* verfuegbar gemacht. Der Befehl bleibt undefiniert. Der Grund dafür liegt darin, daß *Plain* TEX ein Standardformat ist, das nicht zu viele Zeichensätze enthalten soll. Dennoch werden die Zeichensatzinformationen hier in TEX's Speicher geladen und nicht erst beim Aufruf mit '`\font`'. Das spart später eine Menge Zeit.

Die meisten Zeichensätze haben ihre eigenen Werte für '`\/`', Trennungszeichen u.a. Mathematische Zeichen- und Symbolsätze haben das oft nicht, sie müssen erst definiert werden, das geschieht hier.

```
\skewchar\teni='177 \skewchar\seveni='177 \skewchar\fivei='177
\skewchar\tensy='60 \skewchar\sevensy='60 \skewchar\fivesy='60
```

Nachdem nun die Zeichensätze geladen sind, müssen sie noch in Familien, für den Gebrauch in mathematischen Sequenzen eingeordnet werden, und es fehlen noch die Abkürzungen, wie '`\it`' oder '`\rm`'

```
\textfont0=\tenrm \scriptfont0=\sevenrm \scriptscriptfont0=\fiverm
\def\rm{\fam0 \tenrm}
\textfont1=\teni  \scriptfont1=\seveni  \scriptscriptfont1=\fivei
\def\mit{\fam1 } \def\oldstyle{\fam1 \teni}
\textfont2=\tensy \scriptfont2=\sevensy \scriptscriptfont2=\fivesy
\def\cal{\fam2 }
\textfont3=\tenex \scriptfont3=\tenex   \scriptscriptfont3=\tenex

\newfam\itfam \def\it{\fam\itfam\tenit} \textfont\itfam=\tenit
\newfam\slfam \def\sl{\fam\slfam\tensl} \textfont\slfam=\tensl
\newfam\bffam \def\bf{\fam\bffam\tenbf} \textfont\bffam=\tenbf
  \scriptfont\bffam=\sevenbf   \scriptscriptfont\bffam=\fivebf
\newfam\ttfam \def\tt{\fam\ttfam\tentt} \textfont\ttfam=\tentt
```

A.2.5 Textmakros

Hier kommen nun die ersten Makros, zunächst die, die nichts mit dem mathematischen Zeichensatz zu tun haben.

```
\def\frenchspacing{\sfcode`\.=1000 \sfcode`\?=1000 \sfcode`\!=1000
  \sfcode`\:=1000 \sfcode`\;=1000 \sfcode`\,=1000 }
\def\nonfrenchspacing{\sfcode`\.=3000 \sfcode`\?=3000 \sfcode`\!=3000
  \sfcode`\:=2000 \sfcode`\;=1500 \sfcode`\,=1250 }
\def\normalbaselines{\lineskip=\normallineskip
  \baselineskip=\normalbaselineskip \lineskiplimit=\normallineskiplimit}
```

Die nun folgenden Makros sind zwar einfach, aber sie werden oft eingesetzt. Zunächst werden '`<TAB>`' und '`<RETURN>`' so umdefiniert, daß sie als normales Leerzeichen verstanden werden. Dann kommen Makros, für Leute, die sich schwer tun Anführungszeichen und eckige Klammern zu schreiben. Die Definitionen von '`\endgraf`' und '`\endline`' sind nötig, da es oft sinnvoll ist '`\par`' und '`\cr`' selber umzudefinieren. Der Rest erklärt sich von selber.

```
\def\^^I{\ } \def\^^M{\ }
\def\lq{`} \def\rq{'} \def\lbrack{[} \def\rbrack{]}
\let\endgraf=\par \let\endline=\cr
\def\space{ } \def\empty{} \def\null{\hbox{}}
\let\bgroup={ \let\egroup=}
```

Nun folgt die Definition des trickreichen Makros '`\obeylines`' und '`\obeyspaces`'

```
\def\obeyspaces{\catcode`\ =\active}
{\obeyspaces\global\let =\space}
{\catcode`\^^M=\active % these lines must end with `%'
  \gdef\obeylines{\catcode`\^^M=\active \let^^M=\par}%
  \global\let^^M=\par} % this is in case ^^M appears in a \write
```

Die Definition benutzt '`\let`' anstelle von '`\def`' um später mehr Flexibilität zu erreichen.[6] Jetzt folgen die *Schleifen*makros.

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body \let\next=\iterate \else\let\next=\relax\fi \next}
\let\repeat=\fi % this makes \loop...\if...\repeat skippable
```

Nun kommen die Definitionen für Leerräume. Neben den bekannten Befehlen mit ihren Eigenschaften werden hier auch '`\hglue\<Leim>`' und '`\vglue\<Leim>`' definiert, die einen Zwischenraum schaffen, der *nie* entfernt wird.

Die Makros '`\nointerlineskip`' und '`\offinterlineskip`' bewirken beide, daß zwischen den nächsten Zeilen kein Zeilenabstand eingefügt wird. Der zweite Befehl ist dabei der weitergehende. Er bewirkt, daß zwar Zwischenraum da ist, dieser aber die Ausdehnung Null hat. Der erste Befehl wirkt nur auf die nächste Zeile.

```
\def\enskip{\hskip.5em\relax}     \def\enspace{\kern.5em }
\def\quad{\hskip1em\relax}        \def\qquad{\hskip2em\relax}
\def\thinspace{\kern .16667em }   \def\negthinspace{\kern-.16667em }

\def\hglue{\afterassignment\hgl@\skip@=}
\def\hgl@{\leavevmode \count@=\spacefactor \vrule width0pt
  \nobreak\hskip\skip@ \spacefactor=\count@}
```

[6] Beispiele dazu im Kapitel über Tabellen.

```
\def\vglue{\afterassignment\vgl@\skip@=}
\def\vgl@{\par \dimen@=\prevdepth \hrule height0pt
  \nobreak\vskip\skip@ \prevdepth=\dimen@}
\def\topglue{\nointerlineskip \vglue-\topskip \vglue} % for top of page

\def\nointerlineskip{\prevdepth=-1000pt }
\def\offinterlineskip{\baselineskip=-1000pt
  \lineskip=0pt \lineskiplimit=\maxdimen}


\def\smallskip{\vskip\smallskipamount}
\def\medskip{\vskip\medskipamount}
\def\bigskip{\vskip\bigskipamount}
```

Nun kommen die Befehle zum Zeilenumbruch u.a.

```
\def\break{\penalty-10000 } \def\nobreak{\penalty10000 }
\def\allowbreak{\penalty0 }
\def~{\penalty10000\ }
\def\slash{/\penalty\exhyphenpenalty}

\def\filbreak{\par\vfil\penalty-200\vfilneg}
\def\goodbreak{\par\penalty-500 }
\def\eject{\par\penalty-10000 }
\def\supereject{\par\penalty-20000 }

\def\removelastskip{\ifdim\lastskip=0pt \else\vskip-\lastskip\fi}
\def\smallbreak{\par \ifdim\lastskip<\smallskipamount
  \removelastskip \penalty-50 \smallskip \fi}
\def\medbreak{\par \ifdim\lastskip<\medskipamount
  \removelastskip \penalty-100 \medskip \fi}
\def\bigbreak{\par \ifdim\lastskip<\bigskipamount
  \removelastskip \penalty-200 \bigskip \fi}
```

Als nächstes die Boxbefehle

```
\def\line{\hbox to\hsize}
\def\leftline#1{\line{#1\hss}} \def\rightline#1{\line{\hss#1}}
\def\centerline#1{\line{\hss#1\hss}}

\def\llap#1{\hbox to 0pt{\hss#1}} \def\rlap#1{\hbox to 0pt{#1\hss}}

\def\m@th{\mathsurround=0pt }
\def\underbar#1{$\setbox0=\hbox{#1} \dp0=0pt
  \m@th \underline{\box0}$}
```

Beachten sie, daß TeX in den mathematischen Modus umschaltet, um eine Unterstreichung vorzunehmen. Dieser Gebrauch vom mathematischen Modus kommt noch mehrmals vor.

Zunächst aber mal der '`\strut`' Befehl in allen vorkommenden Feinheiten

```
\newbox\strutbox
\setbox\strutbox=\hbox{\vrule height8.5pt depth3.5pt width0pt}
\def\strut{\relax\ifmmode\copy\strutbox\else\unhcopy\strutbox\fi}
```

Der '`\relax`' Befehl wird hier verwendet, um Seiteneffekte zu vermeiden.

Und noch einige Tabellenbefehle

```
\def\ialign{\everycr={}\tabskip=0pt \halign} % initialized \halign
\def\hidewidth{\hskip\hideskip}

\newcount\mscount
\def\multispan#1{\omit \mscount=#1 \loop\ifnum\mscount>1 \sp@n\repeat}
\def\sp@n{\span\omit \advance\mscount by -1 }
```

Die nun folgenden Befehle betreffen alle den Tabellenaufbau. Am besten sehen sie sich die Wirkung mit '`\tracingall`' an einem einfachen Beispiel an.

```
\newif\ifus@ \newif\if@cr
\newbox\tabs \newbox\tabsyet \newbox\tabsdone

\def\cleartabs{\global\setbox\tabsyet=\null \setbox\tabs=\null}
\def\settabs{\setbox\tabs=\null \futurelet\next\sett@b}
\let\+=\relax % in case this file is being read in twice
\def\sett@b{\ifx\next\+ \let\next=\relax % turn off \outerness
    \def\next{\afterassignment\s@tt@b\let\next}%
  \else\let\next=\s@tcols\fi\next}
\def\s@tt@b{\let\next=\relax \us@false\m@ketabbox}
\outer\def\+{\tabalign} \def\tabalign{\us@true \m@ketabbox}
\def\s@tcols#1\columns{\count@=#1 \dimen@=\hsize
  \loop \ifnum\count@>0 \@nother \repeat}
\def\@nother{\dimen@ii=\dimen@ \divide\dimen@ii by\count@
  \setbox\tabs=\hbox{\hbox to\dimen@ii{}\unhbox\tabs}%
  \advance\dimen@ by-\dimen@ii \advance\count@ by -1 }

\def\m@ketabbox{\begingroup
  \global\setbox\tabsyet=\copy\tabs \global\setbox\tabsdone=\null
  \def\cr{\@crtrue\crcr\egroup\egroup
    \ifus@ \unvbox0 \lastbox\fi \endgroup
    \setbox\tabs=\hbox{\unhbox\tabsyet\unhbox\tabsdone}}%
  \setbox0=\vbox\bgroup\@crfalse \ialign\bgroup&\t@bbox##\t@bb@x\crcr}

\def\t@bbox{\setbox0=\hbox\bgroup}
\def\t@bb@x{\if@cr\egroup % now \box0 holds the column
  \else\hss\egroup \global\setbox\tabsyet=\hbox{\unhbox\tabsyet
      \global\setbox1=\lastbox}% now \box1 holds its size
    \ifvoid1 \global\setbox1=\hbox to\wd0{}%
    \else\setbox0=\hbox to\wd1{\unhbox0}\fi
    \global\setbox\tabsdone=\hbox{\box1\unhbox\tabsdone}\fi
  \box0}
```

Es folgen die Makros für die Aufzählungen

```
\def\hang{\hangindent\parindent}
\def\item{\par\hang\textindent}
\def\itemitem{\par\indent \hangindent2\parindent \textindent}
\def\textindent#1{\indent\llap{#1\enspace}\ignorespaces}
\def\narrower{\advance\leftskip by\parindent
  \advance\rightskip by\parindent}
```

Das folgende Makro dient dazu einen neuen Abschnitt in einem Dokument einzuleiten. Es beginnt eine neue Seite, wenn die letzte nahezu voll ist, dann wird ein '`\bigskip`' eingefügt, und der Abschnittstitel in einer eigenen Zeile in Fettschrift ausgegeben. Dieser Titel erscheint auch bei der Ausgabe auf dem Bildschirm. Außerdem wird bei dem ersten Paragraphen keine Einrückung vorgenommen.

```
\outer\def\beginsection#1\par{\vskip0pt plus.3\vsize\penalty-250
  \vskip0pt plus-.3\vsize\bigskip\vskip\parskip
  \message{#1}\leftline{\bf#1}\nobreak\smallskip\noindent}
```

Zum Setzen von Lemmata, Sätzen u.a. dient das folgende Makro

```
\outer\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2\par}%
  \ifdim\lastskip<\medskipamount \removelastskip\penalty55\medskip\fi}
```

Hier nun die Definition der Makros für den Flattersatz. Das zweite Makro dient für die Ausgabe in der Teletyperschrift, da dort die Wortabstände größer sind.

```
\def\raggedright{\rightskip=0pt plus2em
  \spaceskip=.3333em \xspaceskip=.5em\relax}
\def\ttraggedright{\tt\rightskip=0pt plus2em\relax}
```

Schließlich die Akzente, die ja auch via Makro realisiert werden.

```
\chardef\%=`\% \chardef\&=`\& \chardef\#=`\# \chardef\$=`\$
\chardef\ss=\"19
\chardef\ae=\"1A \chardef\oe=\"1B \chardef\o=\"1C
\chardef\AE=\"1D \chardef\OE=\"1E \chardef\O=\"1F
\chardef\i=\"10 \chardef\j=\"11 % dotless letters
\def\aa{\accent'27a} \def\l{\char'40l}

\def\leavevmode{\unhbox\voidb@x} % begins a paragraph, if necessary
\def\_{\leavevmode \kern.06em \vbox{\hrule width0.3em}}
\def\L{\leavevmode\setbox0=\hbox{L}\hbox to\wd0{\hss\char'40L}}
\def\AA{\leavevmode\setbox0=\hbox{h}\dimen@=\ht0
  \advance\dimen@ by-1ex
  \rlap{\raise.67\dimen@\hbox{\char'27}}A}

\def\mathhexbox#1#2#3{\leavevmode
  \hbox{$\m@th \mathchar\"#1#2#3$}}
\def\dag{\mathhexbox279}  \def\ddag{\mathhexbox27A}
\def\S{\mathhexbox278} \def\P{\mathhexbox27B}

\def\oalign#1{\leavevmode\vtop{\baselineskip0pt \lineskip.25ex
  \ialign{##\crcr#1\crcr}}} % put characters over each other
\def\ooalign{\lineskiplimit-\maxdimen \oalign}
\def\d#1{\oalign{#1\crcr\hidewidth.\hidewidth}}
\def\b#1{\oalign{#1\crcr\hidewidth
    \vbox to.2ex{\hbox{\char'26}\vss}\hidewidth}}
\def\c#1{\setbox0=\hbox{#1}\ifdim\ht0=1ex \accent'30 #1%
  \else{\ooalign{\hidewidth\char'30\hidewidth\crcr\unhbox0}}\fi}
\def\copyright{{\ooalign
    {\hfil\raise.07ex\hbox{c}\hfil\crcr\mathhexbox20D}}}


\def\dots{\relax\ifmmode\ldots\else$\m@th \ldots\,$\fi}
\def\TeX{T\kern-.1667em \lower.5ex\hbox{E}\kern-.125em X}
```

```
\def\`#1{{\accent\"12 #1}} \def\'#1{{\accent\"13 #1}}
\def\v#1{{\accent\"14 #1}} \def\u#1{{\accent\"15 #1}}
\def\=#1{{\accent\"16 #1}} \def\^#1{{\accent\"5E #1}}
\def\.#1{{\accent\"5F #1}} \def\H#1{{\accent\"7D #1}}
\def\~#1{{\accent\"7E #1}} \def\\"#1{{\accent\"7F #1}}
\def\t#1{{\edef\next{\the\font}\the\textfont1\accent\"7F\next#1}}
```

Einige spezielle Befehle für die glücklichen Menschen, die eine bessere Tastatur haben, als Normalsterbliche.

```
\let\^^_=\v        \let\^^S=\u        \let\^^D=\^
```

Jetzt jede Menge Möglichkeiten Zwischenraum aufzufüllen.

```
\def\hrulefill{\leaders\hrule\hfill}
\def\dotfill{\cleaders\hbox{$\m@th \mkern1.5mu . \mkern1.5mu$}\hfill}
\def\rightarrowfill{$\m@th \mathord- \mkern-6mu
  \cleaders\hbox{$\mkern-2mu \mathord- \mkern-2mu$}\hfill
  \mkern-6mu \mathord\rightarrow$}
\def\leftarrowfill
  {$\m@th \mathord\leftarrow \mkern-6mu
  \cleaders\hbox{$\mkern-2mu \mathord- \mkern-2mu$}\hfill
  \mkern-6mu \mathord-$}

\mathchardef\braceld=\"37A \mathchardef\bracerd=\"37B
\mathchardef\bracelu=\"37C \mathchardef\braceru=\"37D
\def\upbracefill{$\m@th
  \bracelu\leaders\vrule\hfill\bracerd
  \braceld\leaders\vrule\hfill\braceru$}
\def\downbracefill{$\m@th
  \braceld\leaders\vrule\hfill\braceru
  \bracelu\leaders\vrule\hfill\bracerd$}
```

Die Befehle '`\upbracefill`' und '`\downbracefill`' fürfen nur innerhalb einer '`\hbox`' auftreten, oder in einer Tabelle. Nun kommt noch die Definition von '`\bye`'

```
\outer\def\bye{\par\vfill\supereject\end} % the recommended way to stop
```

A.2.6 Makros für die Mathematik

Viele der in diesem Abschnitt vertretenen Makros dienen nur der Darstellung von Sonderzeichen, sie werden hier nicht gesondert aufgeführt. Zunächst einmal einige einfache Makros, für Leute, die kein '`_`' oder '`^`' auf ihrer Tastatur haben gibt es die Befehle '`\sp`' und '`\sb`'. Dann kommen noch die Makros für die Zwischenräume, und andere Kleinigkeiten

```
\let\sp=^ \let\sb=_         {\catcode`\_=\active \global\let_=\_}
\def\,{\mskip\thinmuskip}  \def\!{\mskip-\thinmuskip}
\def\>{\mskip\medmuskip}   \def\;{\mskip\thickmuskip}
\def\*{\discretionary{\thinspace\the\textfont2\char2}{}{}}
{\catcode`\^^Z=\active \gdef^^Z{\not=}} % ^^Z is like \ne in math

{\catcode`\'=\active \gdef'{^\bgroup\prim@s}}
\def\prim@s{\prime\futurelet\next\pr@m@s}
\def\pr@m@s{\ifx'\next\let\nxt\pr@@@s
   \else\ifx^\next\let\nxt\pr@@@t
  \else\let\nxt\egroup\fi\fi \nxt}
\def\pr@@@s#1{\prim@s} \def\pr@@@t#1#2{#2\egroup}
```

Jetzt kommen die Definitionen für griechische Buchstaben. Um die Geschichte nicht allzu lang werden zu lassen werden Folgen von Befehlen durch drei Punkte angezeigt.

```
\mathchardef\alpha=\"010B ... \mathchardef\omega=\"0121
\mathchardef\Gamma=\"7000 ... \mathchardef\Omega=\"700A
\mathchardef\aleph=\"0240 ... \mathchardef\spadesuit=\"027F
\def\hbar{{\mathchar'26\mkern-9muh}}
\def\surd{{\mathchar\"1270}}
\def\angle{{\vbox{\ialign{$\m@th\scriptstyle##$\crcr
      \not\mathrel{\mkern14mu}\crcr \noalign{\nointerlineskip}
      \mkern2.5mu\leaders\hrule height.34pt\hfill\mkern2.5mu\crcr}}}}
```

Nun die großen Symbole

```
\mathchardef\smallint=\"1273
\mathchardef\sum=\"1350 ... \mathchardef\biguplus=\"1355
\mathchardef\intop=\"1352  \def\int{\intop\nolimits}
\mathchardef\ointop=\"1348 \def\oint{\ointop\nolimits}
```

Auch nichts verwunderliches bei den binären Operatoren

```
\mathchardef\pm=\"2206 ... \mathchardef\amalg=\"2271
```

Auch die Relationen werden einfach linear definiert, mit Ausnahme der Relationen, die aus verschiedenen Zeichen zusammengesetzt sind. Dazu gehört '`\mapsto`' und '`\longmapsto`'

```
\mathchardef\leq=\"3214 ... \mathchardef\perp=\"323F
\def\joinrel{\mathrel{\mkern-3mu}}
\def\relbar{\mathrel{\smash-}} \def\Relbar{\mathrel=}
\def\longrightarrow{\relbar\joinrel\rightarrow}
\def\Longrightarrow{\Relbar\joinrel\Rightarrow}
\def\longleftarrow{\leftarrow\joinrel\relbar}
\def\Longleftarrow{\Leftarrow\joinrel\Relbar}
\def\longleftrightarrow{\leftarrow\joinrel\rightarrow}
\def\Longleftrightarrow{\Leftarrow\joinrel\Rightarrow}
\mathchardef\mapstochar=\"322F \def\mapsto{\mapstochar\rightarrow}
\def\longmapsto{\mapstochar\longrightarrow}
\mathchardef\lhook=\"312C \def\hookrightarrow{\lhook\joinrel\rightarrow}
\mathchardef\rhook=\"312D \def\hookleftarrow{\leftarrow\joinrel\rhook}

\def\neq{\not=}   \def\models{\mathrel|\joinrel=}
\def\bowtie{\mathrel\triangleright\joinrel\mathrel\triangleleft}
```

Nach der Definition von '`\ldotp`' und '`\cdotp`' ist die weitere Definition von den im mathematischen Modus gebräuchlichen Punkten keine Schwierigkeit mehr.

```
\mathchardef\ldotp=\"613A\mathchardef\cdotp=\"6201\mathchardef\colon=\"603A
\def\ldots{\mathinner{\ldotp\ldotp\ldotp}}
\def\cdots{\mathinner{\cdotp\cdotp\cdotp}}
\def\vdots{\vbox{\baselineskip=4pt \lineskiplimit=0pt
    \kern6pt \hbox{.}\hbox{.}\hbox{.}}}
\def\ddots{\mathinner{\mkern1mu\raise7pt\vbox{\kern7pt\hbox{.}}\mkern2mu
    \raise4pt\hbox{.}\mkern2mu\raise1pt\hbox{.}\mkern1mu}}
```

Die meisten mathematischen Akzente werden mit '`\mathaccent`' realisiert, die mit variabler Breite müssen einzeln bereitgestellt werden

```
\def\acute{\mathaccent\"7013 } ... \def\ddot{\mathaccent\"707F }
\def\widetilde{\mathaccent\"0365 } \def\widehat{\mathaccent\"0362 }
\def\overrightarrow#1{\vbox{\ialign{##\crcr
     \rightarrowfill\crcr\noalign{\kern-1pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overleftarrow#1{\vbox{\ialign{##\crcr
     \leftarrowfill\crcr\noalign{\kern-1pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}
\def\overbrace#1{\mathop{\vbox{\ialign{##\crcr\noalign{\kern3pt}
     \downbracefill\crcr\noalign{\kern3pt\nointerlineskip}
     $\hfil\displaystyle{#1}\hfil$\crcr}}}\limits}
\def\underbrace#1{\mathop{\vtop{\ialign{##\crcr
     $\hfil\displaystyle{#1}\hfil$\crcr \noalign{\kern3pt\nointerlineskip}
     \upbracefill\crcr\noalign{\kern3pt}}}}\limits}
\def\skew#1#2#3{{#2{#3\mkern#1mu}\mkern-#1mu}{}}
```

Nun die 24 Klammern, die in der Größe variabel sind

```
\def\langle{\delimiter\"426830A }       \def\rangle{\delimiter\"526930B }
\def\lbrace{\delimiter\"4266308 }       \def\rbrace{\delimiter\"5267309 }
\def\lceil{\delimiter\"4264306 }        \def\rceil{\delimiter\"5265307 }
\def\lfloor{\delimiter\"4262304 }       \def\rfloor{\delimiter\"5263305 }
\def\lgroup{\delimiter\"400033A }       \def\rgroup{\delimiter\"500033B }
\def\lmoustache{\delimiter\"4000340 }  \def\rmoustache{\delimiter\"5000341 }
\def\uparrow{\delimiter\"3222378 }      \def\Uparrow{\delimiter\"322A37E }
\def\downarrow{\delimiter\"3223379 }    \def\Downarrow{\delimiter\"322B37F }
\def\updownarrow{\delimiter\"326C33F } \def\arrowvert{\delimiter\"033C000 }
\def\Updownarrow{\delimiter\"326D377 } \def\Arrowvert{\delimiter\"033D000 }
\def\vert{\delimiter\"026A30C }         \def\Vert{\delimiter\"026B30D }
\def\backslash{\delimiter\"026E30F }    \def\bracevert{\delimiter\"033E000 }
```

Jetzt können auch die '`\big...`' Befehle bereitgestellt werden

```
\def\bigl{\mathopen\big} \def\bigm{\mathrel\big} \def\bigr{\mathclose\big}
\def\Bigl{\mathopen\Big} \def\Bigm{\mathrel\Big} \def\Bigr{\mathclose\Big}
\def\biggl{\mathopen\bigg}             \def\Biggl{\mathopen\Bigg}
\def\biggm{\mathrel\bigg}              \def\Biggm{\mathrel\Bigg}
\def\biggr{\mathclose\bigg}            \def\Biggr{\mathclose\Bigg}
\def\big#1{{\hbox{$\left#1\vbox to 8.5pt{}\right.\n@space$}}}
\def\Big#1{{\hbox{$\left#1\vbox to 11.5pt{}\right.\n@space$}}}
\def\bigg#1{{\hbox{$\left#1\vbox to 14.5pt{}\right.\n@space$}}}
\def\Bigg#1{{\hbox{$\left#1\vbox to 17.5pt{}\right.\n@space$}}}
\def\n@space{\nulldelimiterspace=0pt \m@th}
```

Und jetzt noch ein paar Abkürzungen, die sich auf Klammern beziehen

```
\def\choose{\atopwithdelims()}
\def\brack{\atopwithdelims[]}
\def\brace{\atopwithdelims\{\}}
\def\sqrt{\radical\"270370 }
```

Nun wird es wieder etwas interessanter. Der '`\mathpalette`' Befehl reagiert unterschiedlich, je nach Stil, in dem er aufgerufen wird. Er ist maßgeblich für weitere Befehle. Diese Definition kann auch als Muster dafür dienen, wie andere Definitionen auf die verschiedenen Stile reagieren kann.

```
\def\mathpalette#1#2{\mathchoice{#1\displaystyle{#2}}
  {#1\textstyle{#2}}{#1\scriptstyle{#2}}{#1\scriptscriptstyle{#2}}}

\newbox\rootbox
\def\root#1\of{\setbox\rootbox=
  \hbox{$\m@th \scriptscriptstyle{#1}$}
  \mathpalette\r@@t}
\def\r@@t#1#2{\setbox0=\hbox{$\m@th #1\sqrt{#2}$}
  \dimen@=\ht0 \advance\dimen@ by-\dp0
  \mkern5mu \raise.6\dimen@\copy\rootbox \mkern-10mu \box0}

\newif\ifv@ \newif\ifh@
\def\vphantom{\v@true\h@false\ph@nt}
\def\hphantom{\v@false\h@true\ph@nt}
\def\phantom{\v@true\h@true\ph@nt}
\def\ph@nt{\ifmmode\def\next{\mathpalette\mathph@nt}%
  \else\let\next=\makeph@nt\fi \next}
\def\makeph@nt#1{\setbox0=\hbox{#1}\finph@nt}
\def\mathph@nt#1#2{\setbox0=\hbox{$\m@th#1{#2}$}\finph@nt}
\def\finph@nt{\setbox2=\null \ifv@ \ht2=\ht0 \dp2=\dp0 \fi
  \ifh@ \wd2=\wd0 \fi \box2 }
\def\mathstrut{\vphantom(}

\def\smash{\relax % \relax, in case this comes first in \halign
  \ifmmode\def\next{\mathpalette\mathsm@sh}\else\let\next\makesm@sh
  \fi \next}
\def\makesm@sh#1{\setbox0=\hbox{#1}\finsm@sh}
\def\mathsm@sh#1#2{\setbox0=\hbox{$\m@th#1{#2}$}\finsm@sh}
\def\finsm@sh{\ht0=0pt \dp0=0pt \box0 }

\def\cong{\mathrel{\mathpalette\@vereq\sim}} % \sim over =
\def\@vereq#1#2{\lower.5pt\vbox{\baselineskip0pt \lineskip-.5pt
    \ialign{$\m@th#1\hfil##\hfil$\crcr#2\crcr=\crcr}}}
\def\notin{\mathrel{\mathpalette\c@ncel\in}}
\def\c@ncel#1#2{\ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
\def\rightleftharpoons{\mathrel{\mathpalette\rlh@{}}}
\def\rlh@#1{\vcenter{\hbox{\ooalign{\raise2pt
          \hbox{$#1\rightharpoonup$}\crcr $#1\leftharpoondown$}}}}
\def\buildrel#1\over#2{\mathrel{\mathop{\kern0pt #2}\limits^{#1}}}
\def\doteq{\buildrel\textstyle.\over=}
```

Hier einige Definitionen für alternative Namen

```
\let\ne=\neq         \let\le=\leq         \let\ge=\geq
\let\{=\lbrace       \let\|=\Vert         \let\}=\rbrace
\let\to=\rightarrow  \let\gets=\leftarrow \let\owns=\ni
\let\land=\wedge     \let\lor=\vee        \let\lnot=\neg
\def\iff{\;\Longleftrightarrow\;}
```

Die meisten Funktionsnamen, die in der Schriftart Roman gesetzt werden sind schon bekannt, hier folgen nur einige, die noch nicht vorgestellt wurden

```
\def\arccos{\mathop{\rm arccos}\nolimits}
        ... \def\tanh{\mathop{\rm tanh}\nolimits}
\def\det{\mathop{\rm det}} ... \def\sup{\mathop{\rm sup}}
\def\liminf{\mathop{\rm lim\,inf}} \def\limsup{\mathop{\rm lim\,sup}}
```

```
\def\bmod{\mskip-\medmuskip \mkern5mu
  \mathbin{\rm mod} \penalty900 \mkern5mu \mskip-\medmuskip}
\def\pmod#1{\allowbreak \mkern18mu ({\rm mod}\,\,#1)}
```

Die Definitionen von '`\matrix`' und '`\bordermatrix`' werden hier bereitgestellt.

```
\def\matrix#1{\null\,\vcenter{\normalbaselines\m@th
    \ialign{\hfil$##$\hfil&&\quad\hfil$##$\hfil\crcr
      \mathstrut\crcr\noalign{\kern-\baselineskip}
      #1\crcr\mathstrut\crcr\noalign{\kern-\baselineskip}}}\,}


\newdimen\p@renwd \setbox0=\hbox{\tenex B} \p@renwd=\wd0
\def\bordermatrix#1{\begingroup \m@th
  \setbox0=\vbox{\def\cr{\crcr\noalign{\kern2pt\global\let\cr=\endline}}
      \ialign{$##$\hfil\kern2pt\kern\p@renwd&\thinspace\hfil$##$\hfil
        &&\quad\hfil$##$\hfil\crcr
        \omit\strut\hfil\crcr\noalign{\kern-\baselineskip}
        #1\crcr\omit\strut\cr}}
  \setbox2=\vbox{\unvcopy0 \global\setbox1=\lastbox}
  \setbox2=\hbox{\unhbox1 \unskip \global\setbox1=\lastbox}
  \setbox2=\hbox{$\kern\wd1\kern-\p@renwd \left( \kern-\wd1
    \global\setbox1=\vbox{\box1\kern2pt}
    \vcenter{\kern-\ht1 \unvbox0 \kern-\baselineskip} \,\right)$}
  \null\;\vbox{\kern\ht1\box2}\endgroup}
```

Die nächsten Definitionen sind wieder etwas einfacher

```
\def\cases#1{\left\{\,\vcenter{\normalbaselines\m@th
    \ialign{$##\hfil$&\quad##\hfil\crcr#1\crcr}}\right.}
\def\pmatrix#1{\left( \matrix{#1} \right)}
```

Zum guten Schluß noch die Makros für die abgesetzten Formeln

```
\def\openup{\afterassignment\@penup\dimen@=}
\def\@penup{\advance\lineskip\dimen@
  \advance\baselineskip\dimen@ \advance\lineskiplimit\dimen@}
\def\eqalign#1{\null\,\vcenter{\openup1\jot \m@th
  \ialign{\strut\hfil$\displaystyle{##}$&$\displaystyle{{}##}$\hfil
      \crcr#1\crcr}}\,}

\newif\ifdt@p
\def\displ@y{\global\dt@ptrue \openup1\jot \m@th
  \everycr{\noalign{\ifdt@p \global\dt@pfalse
      \vskip-\lineskiplimit \vskip\normallineskiplimit
      \else \penalty\interdisplaylinepenalty \fi}}}
\def\@lign{\tabskip=0pt\everycr={}} % restore inside \displ@y
\def\displaylines#1{\displ@y
  \halign{\hbox to\displaywidth{$\hfil\@lign\displaystyle##\hfil$}\crcr
    #1\crcr}}

\def\eqalignno#1{\displ@y \tabskip=\centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip=0pt
    &$\@lign\displaystyle{{}##}$\hfil\tabskip=\centering
    &\llap{$\@lign##$}\tabskip=0pt\crcr
```

```
    #1\crcr}}
\def\leqalignno#1{\displ@y \tabskip=\centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip=0pt
    &$\@lign\displaystyle{{}##}$\hfil\tabskip=\centering
    &\kern-\displaywidth\rlap{$\@lign##$}\tabskip=\displaywidth\crcr
    #1\crcr}}
```

Der Wert von '`\lineskiplimt`' wird dabei als '`\normallineskiplimit`' angenommen, nur verändert durch die "Öffnung".

A.2.7 Makros für die Ausgaberoutine

Zuerst werden einige Dinge bzgl. Seitennummern, Kopf- und Fußzeilen geregelt

```
\countdef\pageno=0 \pageno=1 % first page is number 1
\newtoks\headline \headline={\hfil} % headline is normally blank
\newtoks\footline \footline={\hss\tenrm\folio\hss}
  % footline is normally a centered page number in font \tenrm
\def\folio{\ifnum\pageno<0 \romannumeral-\pageno \else\number\pageno \fi}
\def\nopagenumbers{\footline={\hfil}} % blank out the footline
\def\advancepageno{\ifnum\pageno<0 \global\advance\pageno by -1
  \else\global\advance\pageno by 1 \fi} % increase |pageno|

\newif\ifr@ggedbottom
\def\raggedbottom{\topskip10pt plus60pt \r@ggedbottomtrue}
\def\normalbottom{\topskip10pt \r@ggedbottomfalse} % undoes \raggedbottom
```

Das Fußnoten Makro benutzt '`\futurelet`' und '`\aftergroup`', damit die Fußnote nicht als Argument an ein anderes Makro übergeben werden muß.

```
\newinsert\footins
\def\footnote#1{\let\@sf=\empty % parameter #2 (the text) is read later
  \ifhmode\edef\@sf{\spacefactor=\the\spacefactor}\/\fi
  #1\@sf\vfootnote{#1}}
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty=\interfootnotelinepenalty
  \splittopskip=\ht\strutbox % top baseline for broken footnotes
  \splitmaxdepth=\dp\strutbox \floatingpenalty=20000
  \leftskip=0pt \rightskip=0pt \spaceskip=0pt \xspaceskip=0pt
  \textindent{#1}\footstrut\futurelet\next\fo@t}
\def\fo@t{\ifcat\bgroup\noexpand\next \let\next\f@@t
  \else\let\next\f@t\fi \next}
\def\f@@t{\bgroup\aftergroup\@foot\let\next}
\def\f@t#1{#1\@foot}
\def\@foot{\strut\egroup}
\def\footstrut{\vbox to\splittopskip{}}
\skip\footins=\bigskipamount % space added when footnote is present
\count\footins=1000 % footnote magnification factor (1 to 1)
\dimen\footins=8in % maximum footnotes per page
```

Hier steht nun, wie *fließende* Einfügungen realisiert werden

```
\newinsert\topins \newif\ifp@ge \newif\if@mid
\def\topinsert{\@midfalse\p@gefalse\@ins}
```

```
\def\midinsert{\@midtrue\@ins}
\def\pageinsert{\@midfalse\p@getrue\@ins}
\skip\topins=0pt % no space added when a topinsert is present
\count\topins=1000 % magnification factor (1 to 1)
\dimen\topins=\maxdimen % no limit per page
\def\@ins{\par\begingroup\setbox0=\vbox\bgroup} % start a \vbox
\def\endinsert{\egroup % finish the \vbox
  \if@mid \dimen@=\ht0 \advance\dimen@ by\dp\z@ \advance\dimen@ by12\p@
    \advance\dimen@ by\pagetotal \advance\dimen@ by-\pageshrink
    \ifdim\dimen@>\pagegoal \@midfalse\p@gefalse\fi\fi
  \if@mid \bigskip \box0 \bigbreak
  \else\insert\topins{\penalty100 % floating insertion
    \splittopskip=0pt \splitmaxdepth=\maxdimen \floatingpenalty=0
    \ifp@ge \dimen@=\dp0
     \vbox to\vsize{\unvbox0 \kern-\dimen@} % depth is zero
    \else \box0 \nobreak\bigskip\fi}\fi\endgroup}
```

Hier die vollständige Ausgaberoutine

```
\output={\plainoutput}
\def\plainoutput{\shipout\vbox{\makeheadline\pagebody\makefootline}%
  \advancepageno
  \ifnum\outputpenalty>-20000 \else\dosupereject\fi}
\def\pagebody{\vbox to\vsize{\boxmaxdepth=\maxdepth \pagecontents}}
\def\makeheadline{\vbox to 0pt{\vskip-22.5pt
    \line{\vbox to8.5pt{}\the\headline}\vss}\nointerlineskip}
\def\makefootline{\baselineskip=24pt \line{\the\footline}}
\def\dosupereject{\ifnum\insertpenalties>0 % something is being held over
  \line{}\kern-\topskip\nobreak\vfill\supereject\fi}


\def\pagecontents{\ifvoid\topins\else\unvbox\topins\fi
  \dimen@=\dp255 \unvbox255
  \ifvoid\footins\else % footnote info is present
    \vskip\skip\footins \footnoterule \unvbox\footins\fi
  \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
\def\footnoterule{\kern-3pt
  \hrule width 2truein \kern 2.6pt} % the \hrule is .4pt high
```

A.2.8 Trennung und dergleichen

Der letzte Teil von `plain.tex` ließt die Trennungsmuster aus dem File `hyphen.tex` und auch die Ausnahmen

```
\lefthyphenmin=2 \righthyphenmin=3 % disallow x- or -xx breaks
\input hyphen % the hyphenation patterns and exceptions

\def\magnification{\afterassignment\m@g\count@}
\def\m@g{\mag=\count@
  \hsize6.5truein\vsize8.9truein\dimen\footins8truein}

\def\tracingall{\tracingonline=1 \tracingcommands=2 \tracingstats=2
  \tracingpages=1 \tracingoutput=1 \tracinglostchars=1
  \tracingmacros=2 \tracingparagraphs=1 \tracingrestores=1
  \showboxbreadth=\maxdimen \showboxdepth=\maxdimen \errorstopmode}
```

```
\def\showhyphens#1{\setbox0=\vbox{\parfillskip0pt \hsize=\maxdimen \tenrm
  \pretolerance=-1 \tolerance=-1 \hbadness=0 \showboxdepth=0 \ #1}}


\normalbaselines\rm % select roman font
\nonfrenchspacing % punctuation affects the spacing
\catcode`@=12 % at signs are no longer letters


\def\fmtname{plain}\def\fmtversion{3.0} % identifies the current format
```

Anhang B

Trennungen

Es ist sicher besser ein Wort zu trennen, als die Zwischenräume zwischen einzelnen Wörtern zu weit zu strecken. Computer sind normalerweise nicht sehr gut bei der Trennung von Worten. Als die ersten automatischen Trennungen in Zeitungen auftauchten, machten auch schnell die ersten Witze die Runde.[1]

Woher die Schwierigkeiten kommen dürfte klar sein. So wird z.B. das Wort 'record' einmal als 're-cord' und einmal als 'rec-ord' getrennt, je nachdem, wie es verwendet wird. Auch das englische Wort für Trennung: 'hy-phen-a-tion' ist nicht ganz einfach zu trennen, so wird bei dem ähnlichen Wort 'con-cat-e-na-tion' das 'n' mit zu dem 'a' gezogen, im Gegensatz zu dem ersten Beispiel.

Eine gute Lösung für das Trennungproblem fand Frank M. Liang zwischen 1980 und 1982 und TEX übernahm die Methode. Diese Methode findet zu fast allen Wörtern die legitimen Trennstellen, macht nur selten Fehler, ist schnell, verbraucht nur wenig Speicherplatz und ist flexibel genug um schnell auf andere Sprachen angewendet zu werden, ja, sie kann sogar simultan auf zwei Sprachen angewendet werden.

Um ein Wort zu trennen schaut TEX zuerst in ein Ausnahmenregister, um das Wort eventuell dort zu finden. Wenn das Wort dort nicht vorkommt, dann sucht TEX nach Mustern in dem Wort, und das ist der Kernpunkt der Methode von Liang. Hier nun die Vorgehensweise am Beispiel des Wortes 'hyphenation', vorausgesetzt, TEX arbeitet mit den englischen Trennmustern.

Zunächst wird dem Wort ein Zeichen vor- und eins nachgestellt, die als Markierung dienen sollen.

```
.hyphenation.
```

Hier bei wurde das '`.`' als Markierung verwendet. Dieses Wort hat die *Unter*wörter

```
. h y p h e n a t i o n .
```

der Länge eins, und die Unterwörter der Länge zwei

```
.h hy yp ph he en na at ti io on n.
```

und schließlich die der Länge drei

```
.hy hyp yph phe hen ena nat ati tio ion on.
```

und so weiter. Jedes Unterwort der Länge k definiert $k + 1$ kleine Zahlwerte, die die Fähigkeit zur Trennung zwischen zwei Buchstaben ausdrücken. In den folgenden Beispielen werden diese Wert als Indizes angegeben.

'$_0$ h$_0$ e$_2$ n$_0$' meint z.B., daß bei dem Unterwort '`hen`' alle Werte Nul sind mit der Ausnahme zwischen den Buchstaben '`e`' und '`n`', deren Wert zwei ist. Die Werte sind alle Null, es sei denn

[1] Im Gegensatz zu den englischen Beispielen aus dem TEXbook erinnere ich hier an die unvergessenen Beispiele aus der CONTEXT-Dokumentation, wie Klo-ster, oder noch besser Urin-stinkt. Im weiteren werde ich mich aber an die Originalbeispiele halten.

TEX findet in seiner Musterbibliotek ein Muster. Bei unserem Beispiel werden die folgenden Muster gefunden

$_0 h_0 y_3 p_0 h_0$
$_0 h_0 e_2 n_0$
$_0 h_0 e_0 n_0 a_4$
$_0 h_0 e_0 n_5 a_0 t_0$
$_1 n_0 a_0$
$_0 n_2 a_0 t_0$
$_1 t_0 i_0 o_0$
$_2 i_0 o_0$
$_0 o_2 n_0$

TEX berechnet nun das Maximum an Werten, das zwischen zwei Buchstaben vorkommt. Zwischen '`e`' und '`n`' treten z.B. vier signifikante Werte auf. Das Ergebnis dieser Berechnung ist

$._0 h_0 y_3 p_0 h_0 e_2 n_5 a_4 t_2 i_0 o_2 n_0$.

Jetzt kommt der letzte Schritt. Eine Trennung ist akzeptabel an einer Stelle, an der ein ungerader Wert berechnet wurde. Somit ergeben sich zwei mögliche Trennungen: '`hy-phen-ation`'. Das Wort '`concatenation`' ergibt die Muster

$_0 o_2 n_0$
$_0 o_0 n_1 c_0$
$_1 c_0 a_0$
$_1 n_0 a_0$
$_0 n_2 a_0 t_0$
$_1 t_0 i_0 o_0$
$_2 i_0 o_0$
$_0 o_2 n_0$

und das Ergebnis

$_0 c_0 o_2 n_1 c_0 a_0 t_0 e_1 n_2 a_1 t_2 i_0 o_2 n_0$

und damit die möglichen Trennungen '`con-cate-na-tion`'. Weitere Worte, wie das Beispiel aus dem TEXbook 'supercalifragilisticexpialidocious' erspare ich mir an dieser Stelle.

Plain TEX lädt 4447 Muster in seinen Speicher. Angefangen mit '$_0 ._0 a_0 c_0 h_4$' bis '$_4 z_1 z_2$' und '$_0 z_4 z_0 y_0$'. Die Werte zwischen den Buchstaben haben dabei die Werte 0 bis 5. Ein großer ungerader Wert, wie 5 fördert natürlich eine Trennung an dieser Stelle, ein großer gerader Wert, wie 4 verhindert praktisch eine Trennung an dieser Stelle.

Liang fand diese Muster, indem er ein spezielles Wörterbuch bearbeitete und auch solche Ausnahmen wie 'Af-ghan-i-stan' mit aufnahm. Diese Trennmuster reichen aus für nahezu alle vorkommenden Trennungen, solange es sich um englischsprachige Wörter handelt. Für andere Sprachen muß natürlich eine eigene Bibliothek von Wörtern gefunden werden. Normalerweise findet diese Methode nur gültige Trennungen, es werden aber nicht unbedingt alle möglichen Trennungen gefunden, wie man auch an den obigen Beispielen sehen kann.

In seltenen Fällen kann es natürlich vorkommen, daß TEX ein Wort nicht richtig trennt. Dann haben sie noch die Möglichkeit die Trennung explizit anzugeben. Sie schaffen damit einen weiteren Ausnahmeeintrag, der oben schon angesprochen wurde. Das TEXbook wurde mit den Ausnahmen

```
\hyphenation{man-u-script man-u-scripts ap-pen-dix}
```

geschrieben, um auch die Worte 'manuscript', 'manuscripts' und 'appendix' richtig zu trennen. Sie sehen, daß auch der Plural des Wortes angegeben ist. Das hängt damit zusammen, daß das Wort vollständig in der Ausnahmeliste gefunden werden muß. Es handelt sich also tatsächlich um zwei Einträge.

Wenn sie alle Trennungen sehen wollen, die TEX bei einem Text findet, dann können sie mit der Eingabe

```
\showhyphens{Text}
```

eine hbox ausgeben lassen, die alle Wörter des Textes in getrennter Form enthält. Diese Box ist allerdings immer zu klein, so daß sie immer eine Fehlermeldung erhalten werden, kümmern sie sich nicht darum.

Um ein Wort in die Ausnahmeliste aufzunehmen muß der Befehl

```
\hyphenation{<Worte>}
```

am besten zu Beginn des Textes angegeben werden. '`<Worte>`' ist dabei eine Liste von Worten, die durch Leerzeichen getrent sind. Diese Worte enthalten das Zeichen '`-`' als Trennzeichen an den Stellen, an denen es getrennt werden darf. TEX wandelt alle Worte zunächst in Kleinbuchstaben um, bevor die Gleichheit mit einem aufgefundenen Wort geprüft wird. Trennungen werden nicht nach dem ersten Buchstaben und vor dem letzten Buchstaben akzeptiert. Ein Trennungseintrag kann auch keine Trennung enthalten, dann wird TEX das Wort überhaupt nicht mehr trennen.

Die Einträge in die Ausnahmeliste sind global, sie verschwinden nicht am Ende einer Gruppe. Wir ein Wort mehrfach angegeben, dann gilt immer die letzte Angabe.

Völlig analog funktioniert die Angabe der Trennmuster, allerdings mit einem schwerwiegenden Unterschied. Der Befehl

```
\patterns{<Muster>}
```

darf nur für INITEX auftreten. Beim Lauf von TEX können die Muster nicht mehr geändert werden.

Index